Market Driven Systems
Laboratory Exercise 1
Implementation of a Batch Process Control System

Department of Automatic Control
Lund University
Last update: April 2016

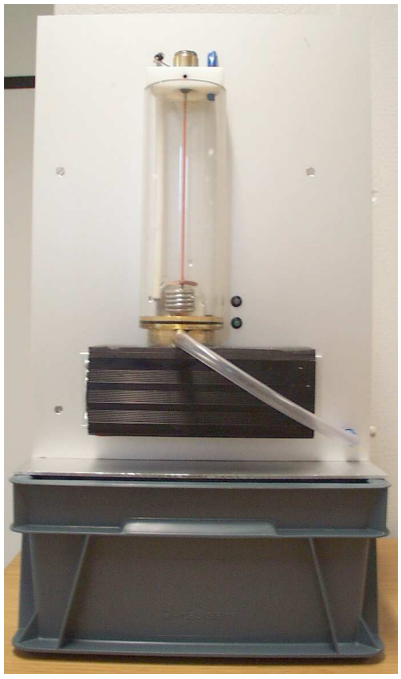


Figure 1 The batch reactor.

1. Introduction

In this laboratory exercise you will implement a sequential control system for a batch reactor process, see Figure 1. The control system will be implemented in the graphical programming language Grafchart, an extension to Grafcet. The control system will first be tested against a simulation and then evaluated against the real process.

The process to be controlled is a batch reactor, which is to be run in the following way: First an amount of reactant A is added with the use of a pump. Then the reactor is heated and an endothermic reaction starts, which turns A into the product B. When the reaction is ready the reactor is emptied using another pump. Once the reactor is empty it needs to be cleaned before the next batch can be made. In the laboratory exercise water is used as both reactant, product, and for cleaning and an electric cooler simulates the endothermic reaction.

Preparations

Before the laboratory exercise you should have read this manual and solved the *preparatory* exercises. Make sure to study and understand the introduction to JGrafchart in appendix A, prior to attending the lab.

2. Getting started

Log in as `lab_batch` (no password required).

Open a terminal window and start JGrafchart with the command `JGrafchart`.

Rather than starting from scratch, open `start.xml` in JGrafchart, it contains some definitions and structure. You should now have the same windows as in Figure 2.

If you close the simulation window you can restart it again from a terminal window with `./labstart`.

Finally, ensure that the process is connected to a 230 V socket and to the lab PC via serial cable. There is a switch on the side of the process. Make sure it is switched on. A LED at the front of the process is lit when the process is on. If the LED is green everything is OK. If it is red, press the reset button on the side of the process.

3. Laboratory Equipment

The process consists of a small tank with an electrical heater. It has a number of measurement and control signals, which are tied to variables in the `Top` workspace of the JGrafchart application.

The electrical heater is controlled with the real variable `PID.u`. At the bottom of the tank there is a cooler which is on when the Boolean variable `Cool` is 1. In addition to cooling on demand, the cooler is also used to simulate the endotermic reaction. An agitator in the tank makes sure there are no temperature gradients in the liquid. The agitator is started using the Boolean variable `Agitator`. There is one pump for filling and one for emptying the tank. They are controlled using the Boolean variables `InPump` and `OutPump` respectively.

There are two real variables connected to sensors in the process. The `Level` variable is in the range 0–1, corresponding to 0–10 V from the level sensor. A larger number corresponds to higher water level. `Temp` is in the range 0–100, which corresponds to 0–100 °C.

The process has some interlocks built in, e.g. it is not possible to heat the tank if the level is too low. In case something prohibited takes place the process sets `Error` to 1 and the LED on the front of the process turns red. If this happens, try to resolve the cause and then press the reset button at the side of the process.

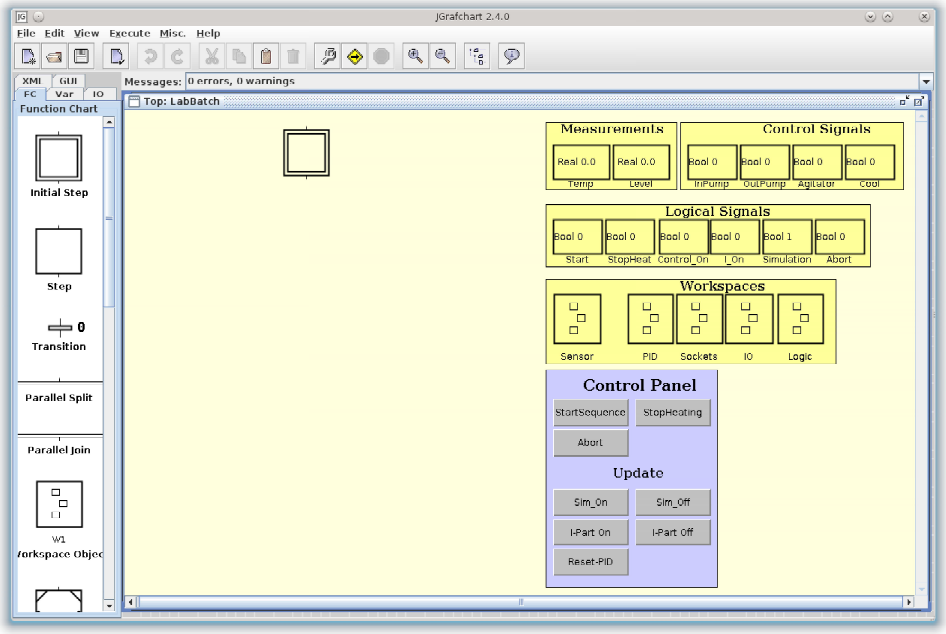
4. Modeling and Simulation

We will control two states in the batch reactor: the water level and the temperature of the water.

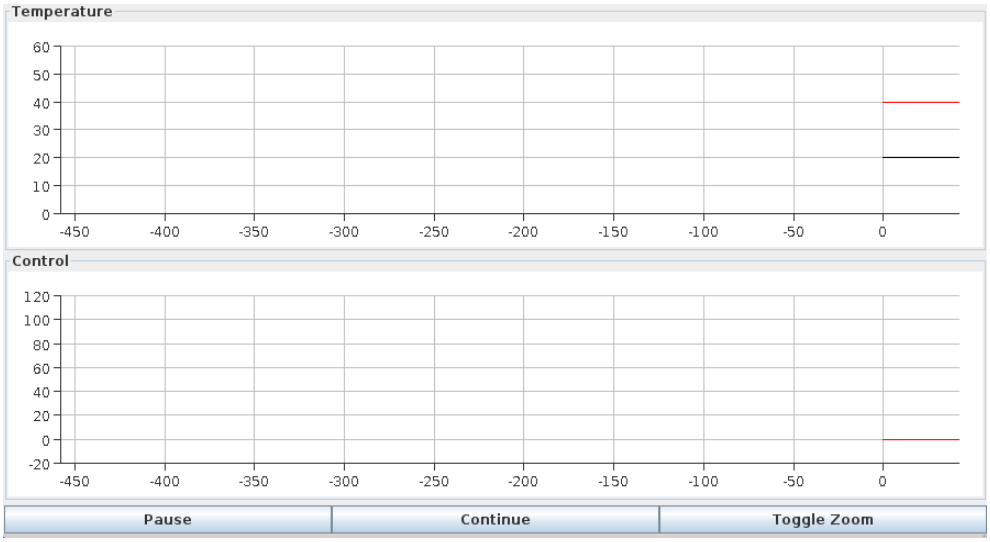
4.1 Water Level Dynamics

Let us denote the water level as a function of time with $h(t)$ in meters. Mass conservation gives the differential equation

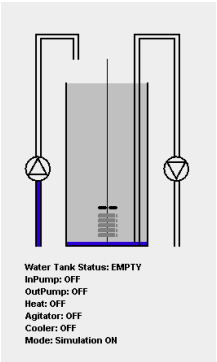
$$A\frac{dh}{dt} = q_{in} - q_{out} \quad [\mathrm{m}^3/\mathrm{s}]. \tag{1}$$



(a) JGrafchart with the Top workspace of the application showing.



(b) Plotter



(c) Tank animation

Figure 2 Windows of the graphical user interface.

Here q_{in} [m^3/s] is the flow of water from the in-pump and q_{out} [m^3/s] is the flow from the out-pump. The tank cross-section area is $A = 0.06^2\pi$ m^2. As the dynamics are simple there will be no need for advanced control, simple on/off control of the pumps will suffice. The pumps have a maximum capacity of 15 l/min, but this will not be fully exploited. The water level measurement from the process is normalized, $h(t)/h_{max}$, so that it becomes a number between 0 and 1 in JGrafchart.

4.2 Water Temperature Dynamics

Let us denote the temperature with $T(t)$ [°C]. In our simple model we will assume that the temperature is uniform throughout the tank. This is not completely true, but we will use an agitator to make this assumption close to reality. The temperature dynamics are modeled by energy balance, described below. The combined specific heat of the water and the tank are represented by the constant c [J/kg°C] and their combined mass is denoted m [kg]. The heat conduction between the tank and the room is modeled by k [W/°C]. Further, there is a heat source q_{heat} [W] and a heat sink q_{cool} [W]. The balance equation becomes

$$mc\frac{dT}{dt} = k \cdot (T_{room} - T) + q_{heat} - q_{cool}(T_{room} - T) \quad [\mathrm{W}]. \tag{2}$$

The room temperature T_{room} is approximately 20°C. Notice that if the heater and cooling is off the water temperature T will tend to T_{room}, which makes sense. (2) is only valid for temperatures below the boiling point and above the freezing point.

The cooling $q_{cool}(T)$ is provided by a *Peltier* thermoelectric element. Its efficiency depends on the temperature difference between the water and the room. It gets more efficient as the temperature of the water increases. When the temperature difference is zero, the cooling is about 40 W. In the working temperature range, the dependence is well described by a linear relation. Apart from cooling upon request, the cooling is also used to simulate the endothermic chemical reactions.

The heater can deliver a maximum of $q_{heat} = 150$ W. It will be scaled according to

$$q_{heat} = 150\frac{u}{100} \quad [\mathrm{W}],$$

where u is our control signal and is a dimensionless number between 0 and 100. u is the output from the controller we will design in JGrafchart.

As seen in (2) there are many physical parameters to determine. However, for our purpose there is no need to measure each single parameter. Instead, as we know the structure of the model, we can fit the model to some experimental data and get a reasonable result. Step response experiments yield

$$\tau \cdot \dot{T} + T = \kappa_1 + \kappa_2 \cdot u \quad [^\circ\mathrm{C}] \tag{3}$$

when the cooling is on, at normal room temperature, and the level is approximately one cm over the agitator. The open-loop time constant τ is 1075 seconds. The other constants are $\kappa_1 = -4.33$°C and $\kappa_2 = 1.74$°C.

4.3 Simulation

As the temperature dynamics of the real system are relatively slow with a time constant of 1075 s, we will use a simulated model that runs ten times as fast during the design phase of the controller. The animation window, see Figure 2(c), shows the status of the simulated tank. It is also updated when running the real process.

The heat control signal u and the temperature T are plotted in another window, see Figure 2(b). In this window the nominal control signal `v1` and the temperature reference `Tref` are also plotted. The plots should be used to evaluate the controller performance. The plotting can be paused and resumed by clicking `Pause` and `Continue` in the `Plotter`, see Figure 2(b). The y-axis scale of the control signal plotter can also be changed with the `Toogle Zoom` button.

JGrafchart communicates both with the real and the simulated process. If `Simulation` in JGrafchart is 1 the simulated model, running ten times as fast as the real process, is used. If `Simulation` is 0 the real world process is used. The necessary time-scaling of parameters is done automatically.

5. Sequential Control

In this part of the laboratory exercise you will develop a sequence for controlling the batch reactor. The sequence should be described as a Grafcet diagram and then translated to JGrafchart. The sequence will be tested against the simulated process and then evaluated against the real process.

Sequential Control of the Batch Reactor

The reactor is making batches over and over again. The making of one batch is outlined below. All buttons and variables live in the `Top` workspace, unless stated otherwise.

1. The operator starts the batch by pressing the `StartSequence` button, which sets the Boolean variable `Start` to 1.
   Note: `Start` is automatically reset by the `Logic` workspace.
2. Once the start button has been pressed, the reactor should be filled using the in-pump, which is running as long as the Boolean variable `InPump` is 1. The filling should be stopped when the Boolean variable `Sensor.Full` (i.e. the variable `Full` in the `Sensor` workspace) becomes 1.
3. As soon as the in-pump is stopped, the agitator should be started by setting `Agitator` to 1 and the heating controller should be turned on by setting `Control_On` to 1. To simulate the endothermic reaction, cooling should be enabled by setting `Cool` to 1.
4. Heating control and agitation should be stopped when the operator presses the button `StopHeating`, which assigns the Boolean variable `StopHeat` the value 1. It is the responsibility of the operator to wait until a desired temperature is reached before pressing the button, and the endothermic reaction is then assumed to have stopped. When heating and agitation have stopped, the tank should automatically be emptied by means of the out-pump, controlled by the Boolean variable `OutPump`. The out-pump should be turned off once `Sensor.Empty` becomes 1.
5. The tank needs to be automatically Cleaned In Place (CIP) before the batch sequence can be repeated. The CIP procedure consists in flushing and cooling the tank: The tank should be filled until the variable `Sensor.Full` becomes 1. Subsequently, the agitator is started and the cooler is activated by setting `Cool` to 1. When temperature has dropped to 25°C, agitation and cooling should stop. Temperature measurements in units of °C are available through the variable

`Temp`. Next, the out-pump empties the tank until `Sensor.Empty` becomes 1. Finally, the out-pump is turned off. After this, execution should return to the initial state, where the process waits until the operator presses the start button anew.

Note: It might happen in simulation, as well as in the real process, that the temperature is below 25°*C* already at the beginning of the CIP step. If this is the case, the CIP step will only involve flushing of the tank.

Preparation Exercise 5.1 Draw a Grafcet diagram (pen and paper), which describes the sequence above. Use a macro step to implement the CIP. ◇

You need to make some adjustments to be able to use your sequence in JGrafchart.

Preparation Exercise 5.2 Translate your Grafcet sequence in Preparation Exercise 5.1 to the programming language syntax of JGrafchart (still pen and paper). It means that in this exercise you should write (draw) the exact code that is necessary to run the control system. Use the exact names of the variables mentioned above. See appendix A for details of the JGrafchart language. Use the provided skeleton, as shown to the left in Figure 2(a).

Note: Variables in subworkspaces are accessible with the dot-notation, e.g. the variable `Full` in the `Sensor` workspace is accessed with `Sensor.Full`. You can write `Sensor.Full` in the body of the `CIP` macro step too since JGrafchart uses lexical scoping and automatically looks in the enclosing workspace if there is no local match. ◇

Programming and Simulation of the Control Sequence

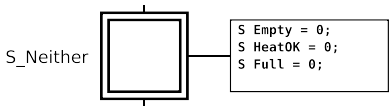
Exercise 5.3 *Implement a discrete level sensor in the* `Sensor` *workspace*. A workspace is opened with `Show/Hide Body` in its context menu. The contents of the `Sensor` workspace template, shown in Figure 3, should now be visible in a new window. Your task is to program a JGrafchart sequence, setting the Boolean variables `Full` and `Empty`, using the real variable `Level`. The `Full` variable should be set to 1 if and only if `Level >= 0.50` and `Empty` should be 1 if and only if `Level <= 0.025`. Ignore `HeatOK` for now. ◇

Exercise 5.4 Implement your sequence from Preparation Exercise 5.2. Test it in simulation. Compile the program by selecting `Compile` from the `Execute` menu (same as the monkey wrench in the toolbar). Start the simulation by selecting `Execute` from the same menu (or the sign with a right-arrow in the toolbar). Don't forget to press the `StartSequence` button to start one batch, and the `StopHeating` button to start CIP.

Note: Execution is stopped by selecting `Stop` from the `Execute` menu.

Note: You have to stop, re-compile, and execute each time you make changes.

Note: The Boolean variable `Simulation`, found in the `Top` workspace, should be set to 1, which is the default. If you have changed it, press the `Sim_On` button when your program is running. ◇



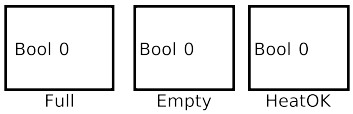


Figure 3 Contents of the `Sensor` workspace template

Evaluation using the Real Process

When the sequence gives a satisfying result in simulation it is time to try it on the real process.

Exercise 5.5 Make sure that the computer and the process are connected and that the LED on the front of the process is green. Set the value of `Simulation` to 0 (by clicking the `Sim_Off` button during execution) to use the real process. You might need to calibrate the level sensor, which you have implemented in the `Sensor` workspace. The `Empty` level should correspond to no (or very little) water in the tank and the `Full` level should correspond to the tank being roughly half full. Evaluate the sequence on the real process. ◇

6. Temperature Control

Until now, the temperature has been controlled by means of a Proportional (P) controller with proportional gain $K = 1000$, making it behave like an on/off controller. We will now investigate how control performance is affected by varying the proportional gain of the P controller. Subsequently, the controller will be extended to a Proportional Integrating (PI) controller in order to achieve better control performance (in terms of tracking, disturbance rejection, and control signal activity).

P control

The P-controller is implemented in the `P_Controller` macro step in the `PID` workspace, see Figure 4 and Figure 5. Make sure you understand its implementation prior to proceeding.

The `PID` workspace also contains the `PI_Controller` macro step and logic to direct execution to either the P or the PI controller. The `PI_Controller` macro step will be handled later.

Controller parameters and other variables related to the PID implementation are also located in the `PID` workspace, e.g. `K` and `Tref`.

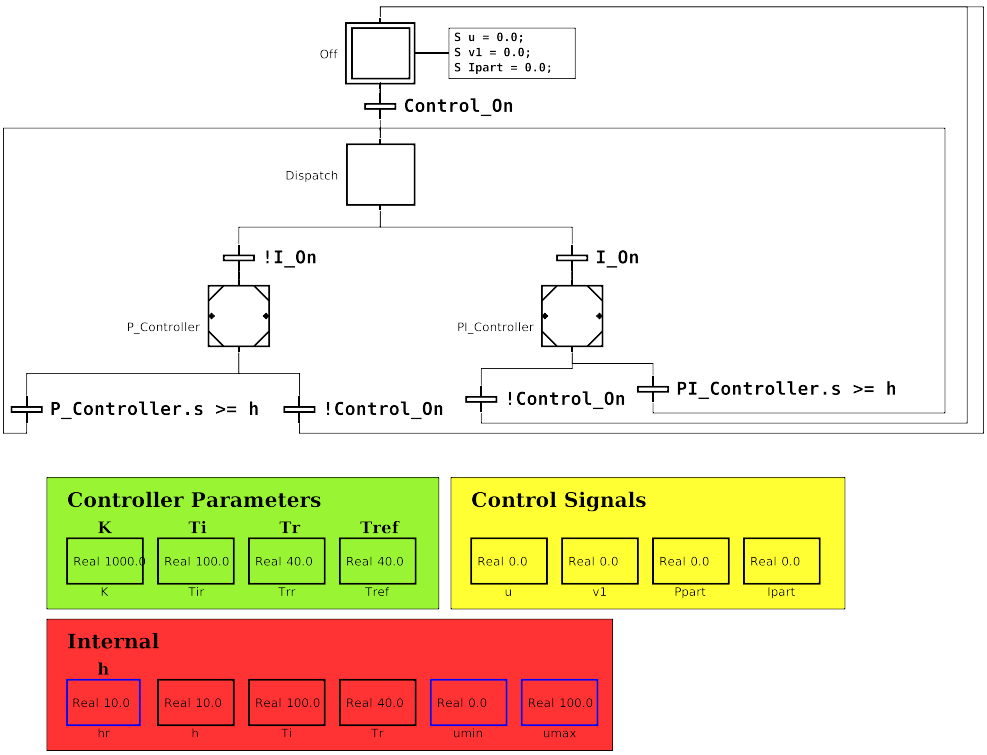


Figure 4 Contents of the `PID` workspace

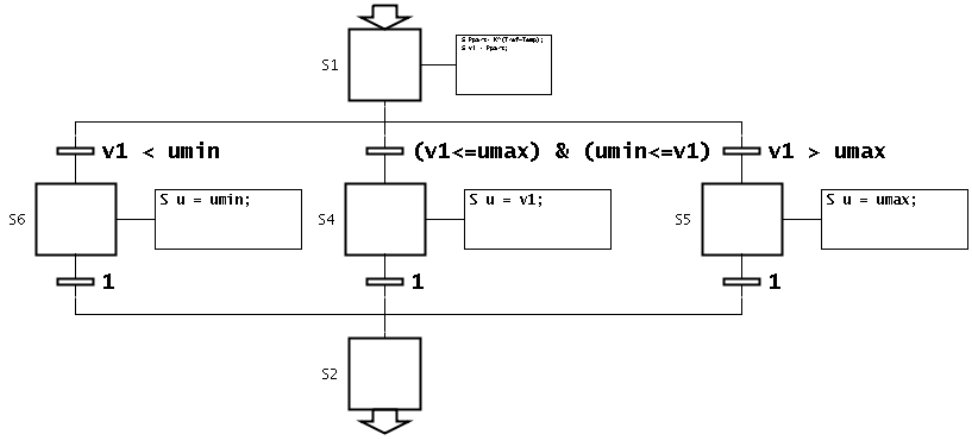


Figure 5 P controller for the `P_Controller` macro step

Exercise 6.1 Turn simulation back on. Study how the gain K of the P controller influences the heating behavior. Use $K = 4,\ 15,\ 200$. Set the reference signal `Tref` to 40 °C. Try to explain the observed behavior.

Note: There are no traces of the D-part in the `PID` workspace since we are not interested in derivative action. Why? ◇

PI control

For processes working around a stationary point corresponding to non-zero input signal, P control results in a stationary control error. This error can be decreased by

increasing the proportional gain. However, this is done at the expense of stability margins and noise suppression. An attractive alternative to increasing the gain is to introduce an integrator in the controller.

A continuous time PI controller has the transfer function:

$$U(s) = K\left(1 + \frac{1}{sT_i}\right)E(s)$$

where $E(s) = Y_r(s) - Y(s)$. The signals and constants are:

- U control signal
- Y process output
- Y_r reference
- E control error
- K proportional gain
- T_i integral time

To be able to implement this controller in a computer, the integrating parts must be replaced by a discrete time approximation. Here this is done by assuming constant control error between sample points. This approximation results in the following pseudo-code implementation (running once per sample period):

```
Ppart = K*(r-y)
v1 = Ppart + Ipart
if v1 <= umin:
  u = umin
else if v1 >= umax:
  u = umax
else:
  u = v1
Ipart = I_Old+(K*h/Ti)*(r-y)
```

where h is the sample time.

In order to avoid integrator windup caused by actuator limitation the PI controller is extended with so called tracking-based anti-windup. This requires the additional constant:

- T_r tracking time constant

With anti-windup the code for the update of the I-part is now modified to

```
...
Ipart = I_Old+(K*h/Ti)*(r-y)+(h/Tr)*(u-v1)
```

The temperature controller will only start if the Boolean variable `Control_On` is 1, and that the PI controller (as opposed to the P controller) is used if the Boolean variable `I_On` is 1. To manipulate `I_On`, use the `I-Part On` and `I-Part Off` buttons.

Exercise 6.2 Make sure that you understand the implementation of the PI controller. Simulate the system with the PI controller. Use the parameters

$$K = 30$$
$$T_i = 100$$
$$T_r = 40$$

Study the behavior of the system. ◇

Exercise 6.3 Run the entire batch sequence with the PI controller on the real process. Use the same controller parameters as for the simulation. How reliable is the model of the system used for simulation? ◇

7. Grafcet with parallel branches

In the previous solution the heating controller was not started until the level had reached the `Full` level. However, if one assumes that the reaction starts as soon as the tank is beginning to be filled then the heat controller should be active also during the filling. However, the physical process contains an interlock that shuts down the process if the heater is on when there is too little water in the tank. Hence, the heat controller may not be turned on until the level has passed the `HeatOK` value.

The new sequence of the the batch now becomes

1. The operator starts the batch by pressing the `StartSequence` button.
2. Once the start button has been pressed, the reactor should be filled using the in-pump.
3. When the level reaches the value `Sensor.HeatOK` the agitator should be started and the heating controller should be turned on. The simulated endothermic reaction should be started here (due to another interlock cooling cannot be with too little water either). The in-pump should still be running.
4. The filling should be stopped when the Boolean variable `Sensor.Full` is true.
5. When the tank is full the heating control and agitation can be stopped when the operator presses the button `StopHeating`, which assigns the Boolean variable `StopHeat` the value 1. It is the responsibility of the operator to wait until a desired temperature is reached, before pressing the button. If the tank is not full, pressing `StopHeat` should have no effect. When heating and agitation have stopped, the tank should automatically be emptied by means of the out-pump, controlled by the Boolean variable `OutPump`. The out-pump should be turned off once `Sensor.Empty` becomes 1.
6. The tank is Cleaned In Place.

Exercise 7.1 Modify the Grafcet sequence according to the new specification. Use the parallel split and join constructs. You also need to modify the Sensor workspace to implement the discrete `HeatOK` sensor. `HeatOK` should be true whenever `Level >= 0.28`. Start by testing using the simulator and then run it on the physical process. The level for `HeatOK` might need to be calibrated, it should correspond to approximately 2 cm above the agitator blades.

Exercise 7.2 Modify the solution so that it supports running a sequence of batches. The number of batches should be determined by a parameter that you need to create. In order to have a better structured solution you should embed the sequence for a single batch into a Macro Step.

Exercise 7.3 Extend the solution so that it is possible to abort the production by clicking the `Abort` button which sets `Abort` to 1. Implement this using an exception transition. Make sure that the tank is emptied, the agitation and cooling is turned off, and that the heating controller is switched off after an exception has occurred.

8. Conclusions

During the laboratory exercise we have developed a small control program for a batch reactor. The program contains both a sequential part and a PI controller for temperature control. This mix of control loops and logic is very common and can be found in all from highly complex industrial processes to electric domestic appliances such as laundry machines.

A. Introduction to JGrafchart

JGrafchart is a freeware function chart editor and execution environment developed at the department.

In this lab only a subset of all elements in JGrafchart are available and only what you might need to know to do the lab is described in this appendix.

A.1 Example

To get an idea of what a JGrafchart application might look like consider a timed lamp which should be on for two minutes after the button has been pressed.

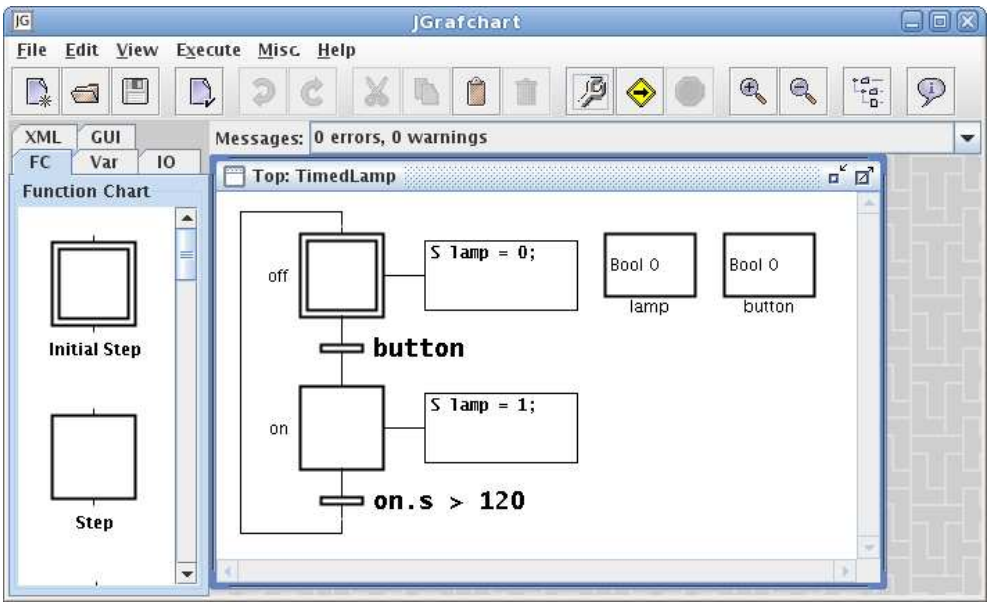


Figure 6 JGrafchart with the example application opened. The palette is to the left, the list of compilation messages is at the top, and the application itself is to the right.

A JGrafchart application for this may look like in Figure 6. The Boolean variable `lamp` control the lamp and the Boolean variable `button` reads if the button is pressed. The steps `off` and `on` correspond to the lamp's off and on state. The execution will start in the initial step, i.e., `off`. We switch from `off` to `on` when `button` is true (1), and from `on` to `off` when `on` has been active for more than 120 seconds. The action associated with `off` (`S lamp = 0;`) sets `lamp` to false (0) when the step is activated. Similarly the action associated with `on` sets `lamp` to true (1) when that step is activated.

A.2 Programming in JGrafchart

Programming in JGrafchart is done by drag-and-drop from the palette to a workspace. The palette contains steps, transitions, variables, and lots of other elements. It is possible to select, delete, copy, cut, and paste objects in the standard fashion. Function chart objects (objects in the FC palette tab) are connected by click-dragging the stubs. The rules of Grafcet apply, for example a step cannot be connected to a step.

A.3 Compilation

Before an application can be executed it must be compiled. This is done with *Compile* in the *Execute* menu or in the toolbar. Compilation errors are indicated by red

text color of the condition expression/step actions. Error messages are written to the messages list.

Two types of problems may arise during compilation, namely syntactic errors and semantic errors.

For example, the condition expression `y OR z` would generate a syntactic error since two consecutive variables, here `y` and `OR`, is not allowed. What the programmer probably meant here was `y | z`.

Semantic errors means that the syntax is correct but what you are trying to do is not possible. One example is trying to assign to an input. Another example is if name lookup fails, for example if there is no variable named `y` in the previous example.

A.4 Execution

JGrafchart applications are executed periodically. Every period (scan cycle) the following operations are performed:

1. Read digital and analog inputs.
2. Determine which transitions that will fire and mark these.
3. Fire all marked transitions, meaning first deactivating the preceding steps (executing their exit actions) and then activating the succeeding steps (executing their enter actions).
4. Update step timers (accessed through <stepName>.t or <stepName>.s), execute periodic actions, and prepare normal action updates.
5. Update variables subject to normal actions.

A.5 Expressions

The expression syntax is similar to Java. One important difference is that the literals 0 and 1 are used both for the Boolean values true and false and for the integer values 0 and 1. It is the context that decides the interpretation.

Supported operators are: +, -, *, /, ! (negation), & (and), | (or), == (equals), != (not equals), <, >, <=, >=.

Expressions may contain name references to variables. JGrafchart uses lexical scoping based on workspaces. For example, a variable named Y on workspace W1 is different from a variable named Y on workspace W2. References between workspaces are expressed using dot-notation. For example, a step action in a step on workspace W1 can refer to the variable Y on workspace W2 with `W2.Y`.

The expression <stepName>.x returns 1 if the step is active and 0 otherwise. The expression <stepName>.t returns the number of scan cycles since the step was last activated, or 0 if it is not active. The expression <stepName>.s returns the time in seconds since the step was last activated, or 0 if it is not active.

A.6 Actions

Associated with each step is a set of actions.

Some different action types are supported:

- **Enter action** (Stored action): The action is executed once when the step becomes active.
  `S <action>;`

- **Periodic action**: The action is executed periodically, once every scan cycle, while the step is active.
  `P <action>;`
- **Exit action**: The action is executed once immediately before the step is deactivated.
  `X <action>;`
- **Abort action**: The action is executed once when the step is aborted due to the firing of an exception transition.
  `A <action>;`
- **Normal action** (Level action): Associates the truth-value of a Boolean variable with the activation status of the step. The variable becomes 1 when the step is activated and 0 when the step is deactivated.
  `N <Boolean variable>;`

where <action> is either an assignment or a call:

- **Assignment**: <variable> = <expression>
- **Call**: <method>()

Note: An action always ends with a semi-colon.

A.7 Conditions

Associated with each transition is a condition which is simply a Boolean expression.

A.8 Grafcet Elements

In this section a selection of Grafchart elements are described.

Steps The step name is shown on the left hand side and step actions are shown to the right if the Action Block is visible.

Step actions are entered as text in the dialog that is opened with *Edit* in the step's context menu.

Initial Steps Initial steps are steps that are actived when the execution of the function chart starts.

Transitions Transitions are associated with conditions or events that should be true in order for the function chart to change state, see Figure 7.

Parallel Split/Join Parallel Split is used to split the execution into two parallel paths and Parallel Join is used to join two parallel execution paths.

Macro Steps A macro step represents a hierarchical abstraction and contains its own (sub-)workspace, see Figure 8. The subworkspace is opened/closed with `Show/Hide Body` in the macro step's context menu. The first step in the macro step is represented by a special enter step. Similarly the final step of the macro step is represented by a special exit step. The enter step and the exit step are otherwise ordinary steps and may for example have actions.

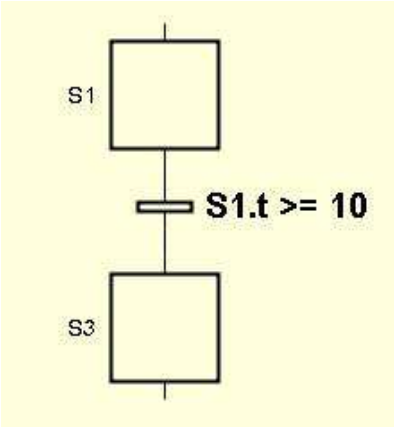


Figure 7 Transition

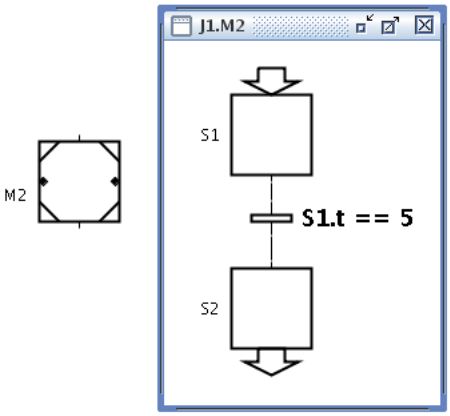


Figure 8 Macro step M2 and its subworkspace containing the enter step S1, the exit step S2, and a transition.

Exception Transitions An exception transition can be connected to the left side of a macro step (the one to the left) and is used to abort the macro step. It has priority over ordinary transitions.

Variables Variables have a value and a name and there is a variable type for each of the primitive data types: real, Boolean, integer, and string.

Action Buttons An action button performs actions when clicked *during execution*.

Workspace Object A Workspace Object contains a subworkspace and is typically used to structure the application.

Text *Text* objects are commonly used as comments.

A.9 Documentation

For more detailed information use the documentation for JGrafchart which can be opened with *Online Help* in the *Help* menu. The documentation for an object can also be consulted interactively with the *Object Help* feature on it, i.e. the speech bubble with an *i* in the toolbar or *Object Help* in the *Help* menu.