Exception Handling in Recipe-Based Batch Control

Rasmus Olsson

Department of Automatic Control
Lund Institute of Technology
Lund, December 2002

Department of Automatic Control
Lund Institute of Technology
Box 118
S-221 00 LUND
Sweden

ISSN 0280–5316
ISRN LUTFD2/TFRT--3230--SE

Contents

Acknowledgments

I would like to thank my supervisor Karl-Erik Årzén for all the help and stimulating discussions during the work, which has led to this thesis. Karl-Erik is always able to make things work on the fly. If you come to him with a problem it will be solved in real-time.

I would like to thank all my colleagues at the Department of Automatic Control for making it a pleasure to go to work every day. Henrik Sandberg, Rolf Braun, and Anders Blomdell for the cooperation and help during the development of the laboratory batch process. Johan Eker for proof reading this thesis.

The cooperation with the Department of Chemical Engineering at Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, has been very fruitful. Thanks to Jordi Canton, Diego Ruiz, Estanislao Musulin and Professor Luis Puigjaner for letting us use the Procel plant.

All the people outside the world of automatic control, who have contributed to my life in different ways, also deserve some credits. A special thanks goes to my Mother, my Father, and my Sister for being there for me my whole life. Props go to Cia, Ylva, the guys in LBWS and all my friends in Skåne, GBG, and the rest of the world.

The development of JGrafchart is funded by LUCAS – Center for Applied Software Research at Lund Institute of Technology and the EC/GROWTH project CHEM aimed at developing advanced decision support systems for different operator-assisted tasks, e.g., process monitoring, data and event analysis, fault detection and isolation, and operations support, within the chemical process industries. The CHEM project is funded by the European Community under the Competitive and Sustainable Growth Programme (1998-2002) under contract G1RD-CT-2001-00466.

This work has been supported by the Center for Chemical Process Design and Control, CPDC, and the Swedish Foundation for Strategic Research, SSF, which I'm very grateful for. The CPDC graduate school has been a good environment for discussions and exchange of ideas between disciplines.

1

Introduction

1.1 Background and Motivation

In a batch process a certain amount of material is transformed, often in several steps, to reach a new state or form. An everyday example is baking bread. A recipe in a cookbook states the following steps for making your own delicious loaf of bread.

- Mix flour, water and yeast in a bowl.
- Form the dough to a loaf.
- Let the dough to rise for thirty minutes.
- Bake the bread in an oven at 225° C for 45 minutes.
- Let the loaf cool down.
- Put the loaf in a plastic bag or eat it right away.

In the same fashion a lot of products can be manufactured in large quantities in factories. A recipe for making a batch of pharmaceuticals looks very much the same as the one for baking bread. Of course there are a lot of more restrictions and regulations, which need to be considered when producing pharmaceuticals.

Batch processes are becoming more and more important in the chemical process industry. Batch processes are used in the manufacture of specialty materials, which are highly profitable. Some examples where batch processes are important are the manufacturing of pharmaceuticals, polymers, and semiconductors. In continuous processes grade and product changes, as well as start-up and shut-down phases can also be seen as batch processes.

The formal definition of a batch process in the batch control standard S88.01 [18] is:

> "A process that leads to the production of finite quantities of material by subjecting quantities of input material to an ordered set of processing activities over a finite period of time using one or more equipment units."

From a control point of view a batch process combines the characteristics of continuous-flow production with those of discrete, part-based production. A batch can be viewed as a discrete entity, which during production moves between different production units. However, during the transition phase from one unit to the next unit and during certain operations within a unit, e.g. fed-batch operations, the batch is more naturally described as a continuous process. The mixed discrete-continuous nature and the recipe-driven production of batch processes make batch control a challenging problem. A batch control system must support a large number of functions in addition to the basic regulatory control. Some examples are production planning, production scheduling, recipe management, resource arbitration and allocation, batch report generation, unit supervision, and exception handling. A graphical approach is convenient to illustrate the different steps of batch processes. Both which actions could or should be taken at a certain instant in time and which history of operations led to the current state of the process can be depicted in a clear way.

The focus of this thesis is exception handling in batch control. Exception handling is a critical element for achieving long-term success in batch production. It is reported to constitute 40-60 percent of the batch control design and implementation effort [8]. Some examples of exception handling situations could be:

- A valve that fails to respond to an `Open` or `Close` command.
- A chemical reaction that does not begin as expected after ingredients have been added.

- A recipe that requires a shared resource which is already in use.
- An equipment unit that is not ready for charging when the preceding vessel expects to transfer material.
- An emergency stop that must be performed due to a potentially hazardous situation.

Correct handling of exceptions of these type is a key element in process safety, consistent product quality, and production cost minimization.

Recently a lot of focus has been put on standardization of the models and terminology used in batch control, e.g. NAMUR [29] and S88 [18]. However, so far very little has been specified in the area of exception handling.

Different discrete-event formalisms has been used for research of the discrete nature of batch control, see e.g. [12, 15, 38]. However, most of this work has been concerned with scheduling, resource allocation and simulation.

1.2 Research Approach

The work presented in this thesis has been performed within the Grafchart group at the Department of Automatic Control, Lund Institute of Technology. The aim of this group is to develop improved domain-specific graphical programming languages for control applications, in particular applications of a discrete and sequential nature. The aim is to extend the languages that are commonly used for this purposes within automation, especially Grafcet/Sequential Function Charts, with better support for structuring, abstraction, encapsulation, re-use, and user interaction. With these extensions it becomes easier to implement complex applications. It is easier for the programmers and process operators to get an overview of the application and the probability of programming errors decreases. The research approach is complementary to the work on formal verification methods for languages of the above type, e.g., [6] [13].

Recipe-based batch control systems is used as the major example domain in the work. The main reason for this is the high complexity of these systems. The research approach is to investigate different language extensions and how these language extensions simplify the development of batch control systems and allow more and more functions in a batch control system to be implemented within the same programming language framework. As far as possible our ambition is to follow the batch control standard S88 (IEC 61512) or to suggest additions to the standard. In previous work [26] the focus has been recipe representation and resource arbitration and allocation. In this thesis the focus is exception handling on both the recipe level and the equipment level.

The work is based on two new language features: MIMO macro steps and step fusion sets. A MIMO macro step is a macro step with multiple input ports and multiple output ports. It can be conveniently used to represent hierarchical states in state-machine based control systems. The functionality is similar to the super-states in Statecharts [16]. Step fusion sets [24] provide a way to have multiple graphical representations, or views, of the same step. Using step fusion sets it is possible to separate the exception-handling logic from the normal operation sequences in a way that improves usability.

1.3 Contributions

In this thesis the previous work on Grafchart for sequential programming, batch process recipe handling and resource allocation [26] is extended to also include exception handling. An internal model approach is proposed where each equipment object in the control system is extended with a state-machine based model that is used on-line to structure and implement the safety interlock logic. The exception detection logic associated with a certain state is only active when the state is active. The internal model provides a safety check to ensure that recipe operations are performed in a correct order. An operation is only allowed to be activated if the state of the equipment unit is within a certain set of allowed states associated with the operation. The thesis treats exception handling both at the unit supervision level and at the recipe level. The goal is to provide a structure, which makes the implementation of exception handling in batch processes easier. The proposed approach uses the MIMO macro step functionality to implement the state machines and step fusion sets to provide separation between the exception handling logic and the logic for normal, fault-free operation.

The exception handling approach has been implemented and tested both in the G2-language implementation of Grafchart and in JGrafchart, a reimplementation of Grafchart in Java. As a test plant Procel, a batch pilot plant, at the Universitat Politècnica de Catalunya (UPC) in Barcelona, Spain, has been used. Procel consists of three tanks with agitators, heaters, and sensors. The tanks are connected in a highly flexible way to be able to run several different types of recipes in the plant.

A good control engineering course should include hands-on experiments. A number of laboratory control processes are available commercially, e.g., inverted pendulums, helicopter processes, level-controlled water tanks, *etc*. These are commonly used in laboratory exercises in different basic and advanced control courses. However, very few processes exist that illustrate the typical problems associated with batch control.

An inexpensive, portable, and flexible laboratory batch process has been developed. The process is used within the chemical engineering education at Lund Institute of Technology. The process can be used for teaching sequential, PID, multi-variable, and recipe-based control. The students implement sequential control and discrete PID control of the batch process in JGrafchart. Since the process is modular, several processes can be connected together to form a low cost batch pilot plant with the same functionality as the Procel pilot plant described above but much more compact.

The main contributions presented in this thesis are:

- A new approach to equipment supervision in recipe-based batch control systems has been proposed. The approach uses hierarchical state machines represented in JGrafchart to model the equipment status and the status of the procedures executing in the equipment.
- A new approach for representing exception-handling logic at the recipe-level has been proposed. The approach gives a clear separation between exception handling logic and the logic for the

normal operation.

- Different possibilities for combining the two above approaches have been suggested.
- The proposed approaches have been implemented in JGrafchart.
- The combined approach has been experimentally verified on a realistic batch pilot plant.
- A new inexpensive and modular batch laboratory process has been developed. The process is used in chemical-engineering control courses as a single-unit process. By connecting several processes a flexible multi-purpose batch cell can be constructed.

Publications

The thesis is based on the following publications:

- Olsson, R. and K-E. Årzén: "Exception Handling in Recipe-Based Batch Control". *Proc. of ADPM2000 The 4th International Conference on Automation of Mixed Processes, Dortmund, Germany, 2000.*
- Olsson, R., H. Sandberg and K-E. Årzén: "Development of a Batch Reactor Laboratory Process", *Reglermötet 2002, Linköping, Sweden, 2002.*
- Olsson R. and K-E. Årzén: "Exception Handling in S88 using Grafchart". *Proc. of World Batch Forum North American Conference 2002, Woodcliff Lake, NJ, USA, 2002.*
- Årzén, K-E., R. Olsson and J. Åkesson: "Grafchart for Procedural Operator Support Tasks". *Proc. of the 15th IFAC World Congress, Barcelona, Spain, 2002.*
- Olsson R. and K-E. Årzén: "A Modular Batch Laboratory Process". Submitted to *International Symposium on Advanced Control of Chemical Processes ADCHEM 2003, Hong Kong, June, 2003.*

1.4 Outline of the thesis

In Chapter 2 batch control in general is discussed. Some contents of the batch control standard S88 are described. Chapter 3 describes the previous work on Grafchart for batch control and the re-implementation of Grafchart in Java. The state-machine based approach to exception handling in recipe-based batch control is described in Chapter 4. The topic of Chapter 5 is the testing and verification of the exception handling approach on the Procel pilot plant. The developed batch laboratory process and its use in an laboratory exercise for students is described in Chapter 6. How the laboratory process can be used as a module for building a plant structure is also described in Chapter 6. In the last chapter of the thesis, Chapter 7, some conclusions and suggestions for future research are made.

2

Batch Control

2.1 Introduction and Motivation

The control of a batch process is quite different from the control of a continuous process. A continuous process usually operates at a certain set point and the control system tries to keep the process at that point. A batch process involves both continuous and sequential parts. For example, when filling a tank, see Fig. 2.1, to a certain level the flow to the tank is continuous. When the desired level is reached the flow is turned of and the next sequential control part can take place, e.g. heating of the tank. The mixed discrete-continuous nature and recipe-driven production of batch processes make batch control a challenging problem. A batch control system must support a large number of functions in addition to the basic regulatory control. The batches need to be scheduled to meet the production plan, recipes should be developed and maintained, shared resources need to be allocated during processing, production reports of batches should be generated and stored, and exceptions need to be detected and taken care of.

Why use automatic control for batch processes at all? On the surface it seems easy to control the sequential part of batch processes manually. The task is just to fill a tank and then start to heat it as in the example above. The procedure is just to wait until a certain goal is reached and then start the next part. However, once there is more than one option of which unit to use, there are common resources to be

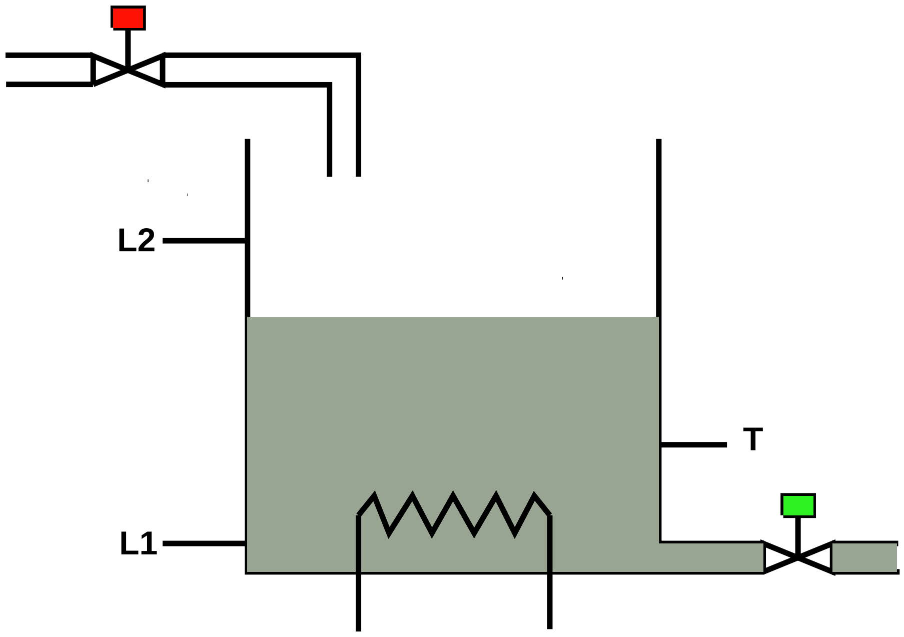


Figure 2.1 Single tank.

shared between different units, or operations need to be synchronized, the task becomes unmanageable.

One of the benefits of automatic control is that the routine operations become fewer for the operators so they can concentrate on more important issues, e.g. exception handling and optimization of the process. Controlling a batch process also leads to repeatability (i.e. consistent product quality batch after batch) and flexibility (i.e. manufacturing of different products and different grades). Another reason are the properties of feedback as such and its capabilities of handling disturbances and uncertainties in the process.

The most complex batch process structure is a network-structured, multi-product plant, see Fig. 2.2. In these kind of plants it is possible to produce multiple products at the same time using a finite set of equipment units. The units are organized in a network structure with a high degree of connectivity. The information about the sequential ordering of the operations, the equipment requirements for the different operations, and the product specific parameters for the manufacturing of a certain product are captured in a *recipe*. In order to execute an operation in an equipment unit, the batch control system must allocate

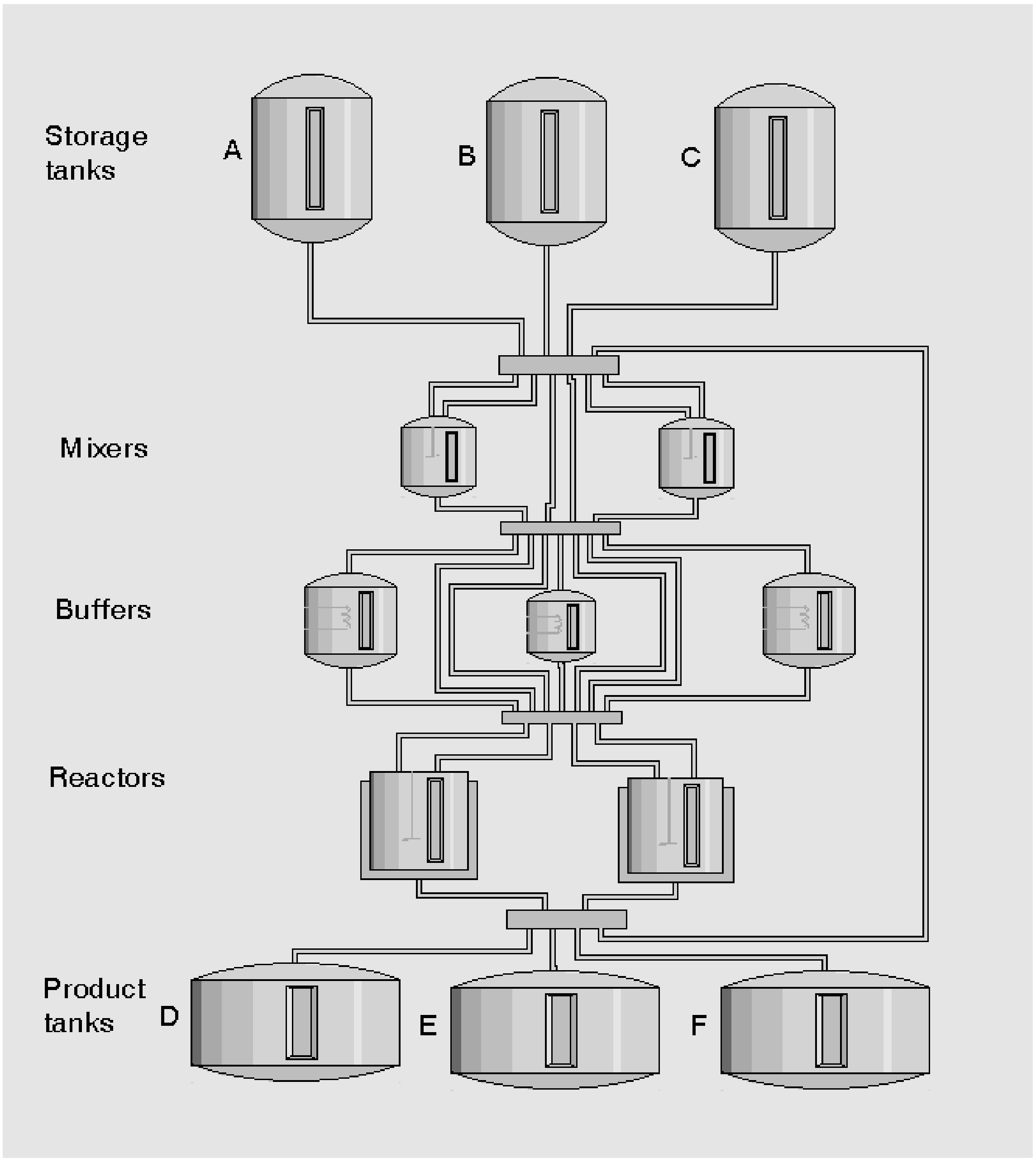


Figure 2.2 Scheme of a network-structured, multi-product plant.

the resource that the equipment unit constitutes. A control system for recipe-based batch production must, hence, include substantially more functions than what is needed in a control system for continuous production. There are a number of books on the subject of control of batch processes, e.g. [11] and [32].

2.2 S88 - Batch Control Standard

During the last decade there have been several initiatives with the aim to standardize batch control systems. The most successful of these attempts is the S88 standard initiated by ISA [22]. The standard is divided into two parts: Part 1 (S88.01) deals with models, terminology and functionality and Part 2 (S88.02) deals with data structures and language guidelines. The ISA S88.01 standard is also known under the names SP88 and IEC 61512-1 [18]. ISA S88.02 is now in the standard IEC 61512-2 [20]. A book that from the user's perspective describes how to implement S88 for an ice-cream factory is [31].

The S88.01 standard describes batch control from two different viewpoints: the process view and the equipment view. The process view corresponds to the view of the chemists, and is modeled by *the process model*. The process model is hierarchically decomposed into the following layers: process, process stage, process operation, and process action. The equipment view corresponds to the view of the production personnel and is represented by *the physical model*. The physical model is also a four-layer hierarchical model containing the following four layers: process cell, unit, equipment module, and control module. A process cell contains one or several units. A unit performs one or several major processing activities. The unit consists of equipment modules and/or control modules. An equipment module carries out a finite number of minor processing activities, i.e. phases (described below). In addition to being a part of a unit, an equipment module can also be a stand-alone, shared or exclusive usage, entity within a process cell. The control module, finally, implements a basic control function, e.g., a PID-controller or the logic handling a valve battery.

The process model and the physical model are linked together by recipes and equipment control. S88 specifies four different recipe types: general, site, master, and control recipes. Here only master recipes and control recipes are considered. Both types of recipes contain four types of information: administrative information, formula information, requirements on the equipment needed, and the procedure that defines how a batch should be produced. The master recipe is targeted towards a specific process cell. It includes almost all information necessary to produce a batch. Each batch is represented by a control recipe. It can be viewed as an instance or copy of the master recipe, that has been com-

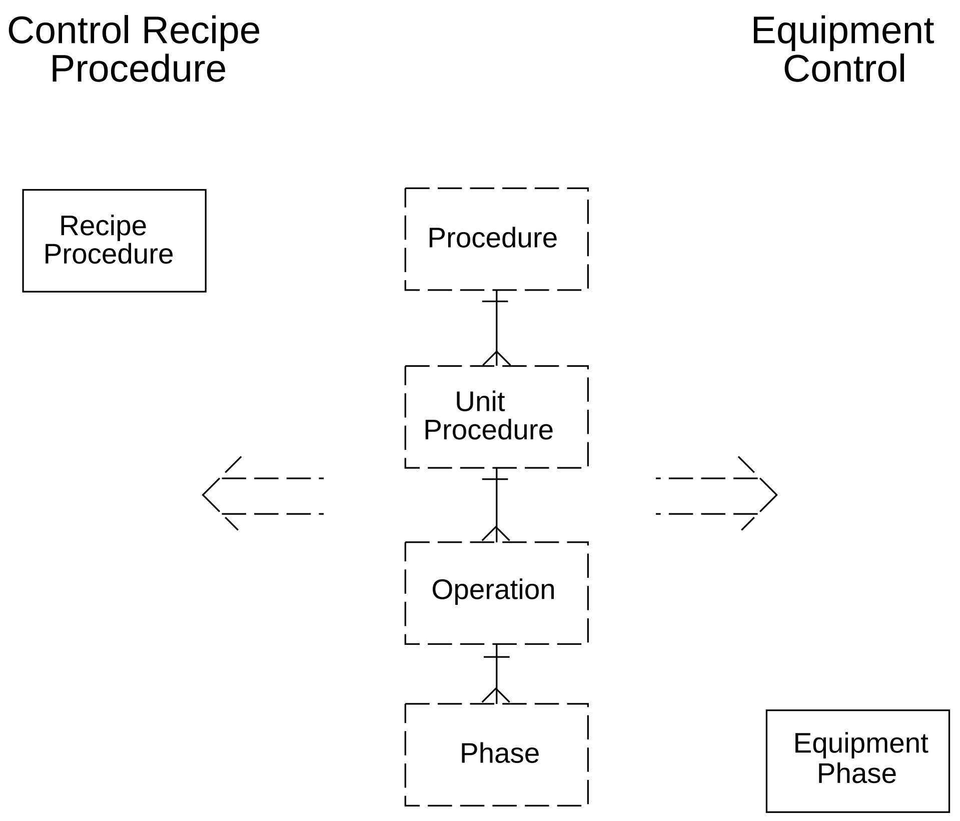


Figure 2.3 Control recipe/Equipment control separation.

pleted and/or modified with scheduling, operational, and equipment information.

The procedure in a recipe is structured according to *the procedural model* in S88. The model is hierarchical with four layers: procedure, unit procedure, operation, and phase. A procedure is decomposed into unit procedures, i.e., sets of related operations that are performed within the same unit. The unit procedures are decomposed into operations, which in turn are decomposed into phases. The phase is the smallest element that can perform a process-oriented task, e.g. open a valve or set an alarm limit.

The procedural model on the recipe level is mirrored by the same model on the equipment control level, see Fig. 2.3. The dashed levels could either be contained in the control recipe or in the equipment control. The linkage between an element in the recipe procedure (unit procedure, operation or phase) and the corresponding element in the equipment control (equipment unit procedure, equipment operation, equipment phase) can be viewed as a method call to the equipment unit object that has been allocated to execute the specific recipe procedure element. S88 offers great flexibility concerning at which level the linkage should take place. It is also possible to collapse one or several levels.

Exception Handling in S88

Surprisingly enough S88 mentions very little about exception handling and how this should be integrated with recipes and equipment control. The only thing that is briefly discussed is that *modes* and *states* may be affected.

Equipment entities and procedural elements may have modes, which determine how they respond to commands and how they operate. Procedural elements have three modes: automatic, semi-automatic, and manual, and equipment entities have two modes: automatic and manual. For procedural elements, in the automatic mode the transitions take place as soon as the transition conditions are fulfilled. In semi-automatic mode, the procedure requires manual approval to proceed after the conditions are fulfilled, and in manual mode the execution of the procedural elements is determined manually. One important issue is how mode changes will propagate. An open question is, e.g., if a unit procedure goes to manual mode should all the lower-level procedural elements within the unit also go to manual mode? The standard does not specify propagation rules. Propagation may be from higher to lower level or vice versa.

Procedural elements and equipment entities may also have states. As an example the following twelve procedural states are mentioned: *idle, running, complete, pausing, paused, holding, held, restarting, stopping, stopped, aborting*, and *aborted*, see Fig. 2.4. If a procedure is in the *idle* state, i.e. it is available, it may be started, and the state will change to *running*. From the *running* state the procedure may be stopped, aborted, held, and paused or the procedure will reach its end and return to the *idle* state through the *complete* state. The state of, e.g., a valve can be: *open, closed*, and *error*. The *error* state would be reached if the valve fails to respond to an open or close command from the control system.

An exception is defined as an event that occurs outside the normal or desired behavior of batch control. A few examples of events that might need exception handling are stated: control equipment malfunction, fire or chemical spills, or unavailability of raw materials or plant equipment. Exception handling may occur at all levels in the control activity model. However, how this should or could be done is not mentioned. The exception handling can be incorporated in any part of the control system.

Although there are no specific safety standards for batch processes, general process industry safety standards, such as IEC 61508 [19] and IEC 61511 [21], also apply to batch processes. A Safety Instrumented System[1] (SIS) [2] of a batch process need a number of additional functionalities, which are different from an SIS of a continuous process. For more info on process safety issues specific to batch reaction systems see [1].

[1]The instrumentation, controls, and interlocks provided for safe operation of the system.

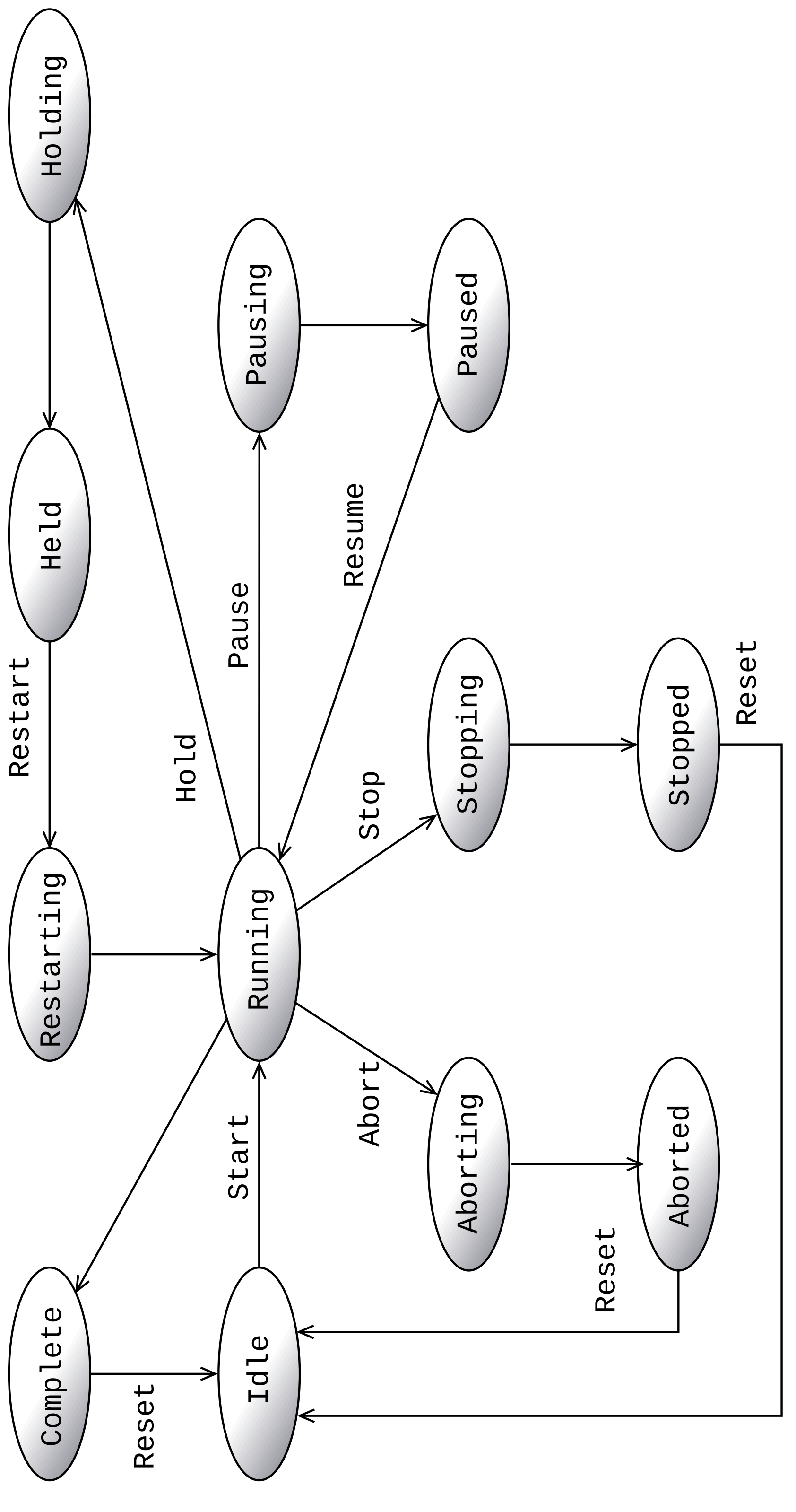


Figure 2.4 The states of a procedural element described in S88.

3

JGrafchart

3.1 Introduction

Grafchart is a graphical programming language for sequential control applications. It is based on Grafcet, or Sequential Function Charts (SFC), together with ideas from Petri nets, Statecharts [16], high-level programming languages and object-oriented programming. A thorough presentation of Grafcet and Petri Nets is given in [9]. Grafcet/SFC is one of the languages specified for PLC (Programmable Logic Controller) programming in the standard IEC 61131-3 [17], and it is widely used as the representation format for the sequential part of supervisory control. Grafchart has been developed at the Department of Automatic Control at Lund Institute of Technology since 1991 [3, 4, 5].

Two different versions of Grafchart exist. The first and ordinary version is based directly on Grafcet whereas the second version, called High-Level Grafchart, also incorporates ideas from high-level Petri nets. Grafchart is described and defined in [26]. Grafchart has a similar syntax to that of Grafcet/SFC, i.e. the basic building blocks are steps, representing states and containing actions, and transitions, representing the change of states. An active step is indicated by the presence of a token in the step. In ordinary Grafchart the tokens are simple boolean indicators, whereas the tokens in High-Level Grafchart are objects that may contain information, e.g. attributes and methods. A step may contain several tokens, of the same or of different classes.

Each step action and each transition condition is associated with a token class. The basic and high-level version of Grafchart has been implemented in G2, a graphical programming environment from Gensym Corp, see [26].

Grafchart contains three hierarchical abstractions: *macro steps, procedures,* and *workspace objects*. Macro steps are used to represent steps that have an internal structure. The internal structure of the macro step is encapsulated within the macro step. A new feature of the macro step is that it may have more than one enter and exit port.

To re-use sequences in a function chart, a sequence can be placed on the sub-workspace of a procedure. Procedures can be stand-alone entities or methods of objects. For example, an object representing a batch reactor could have methods for charging, discharging, agitating, heating, *etc*. A method is called through a procedure step. The method that will be called is determined by an object reference and a method reference. The call to a procedure is represented by a procedure step. The procedure step has an attribute containing the name of the procedure that should be called. The procedure is active as long as the procedure step is active. A process step is similar to a procedure step. The difference is that the procedure is started as a separate execution thread.

To easier organize programs one can use an workspace object. A workspace object contains a sub-workspace that can be used in the same way as a top-level workspace. Using workspace objects it is possible to create complex variables, e.g. structs.

An open problem in Grafcet is how the logic for the normal operating sequence best should be separated from the exception detection and exception handling logic and sequences. Grafchart contains a number of assisting features for this. An *exception transition* is a special type of transition that may only be connected to macro steps and procedure steps. An ordinary transition connected after a macro step will not become active until the execution has reached the last step of the macro step. An exception transition is active all the time that the macro step is active. If the exception transition condition becomes true while the corresponding macro step is executing the execution will be aborted, abortive actions, if any, are executed, and the step following the exception transition will become active. Macro steps "remember" their execution state from the time they were aborted and it is possible to resume them from that state. Exception transitions have proved to be very useful when implementing exception handling. Using *connection posts,* it is possible to break a graphical connection between, e.g., a step and a transition. In this way it is possible to separate a large function chart into several parts that may be stored on different workspaces. This enhances the readability of the chart. The connection post can be used to separate the the normal operating sequence from the exception detection and exception handling logic and sequences. A new language element to separate function charts is the step fusion set. Steps in a step fusion set represents different views of the same step. All the steps in a step fusion set become active when one of the steps becomes active.

3.2 JGrafchart

JGrafchart is an implementation of Grafchart written in Java using Swing graphics, JGo, a class package for graphical object editors from Northwoods Corporation [30], and JavaCC [42], a Java parser generator initially developed by Sun Microsystems. Only the basic version of Grafchart has been implemented so far. JGrafchart consists of an integrated graphical editor and run-time system. In the graphical editor the user creates Grafchart function charts by copying language elements from a palette using drag-and-drop. The language elements are placed on a workspace and connected together graphically. After compilation the function charts are executed by the runtime system within the JGrafchart editor. Storage to file is provided in the form of XML using the Java API for XML Processing (JAXP) [36]. For the complete description of JGrafchart, see `http://www.control.lth.se/~grafchart`, where JGrafchart also will be available for download.

The Graphical Editor

Grafchart function charts are created interactively using drag-and-drop from a palette containing the different Grafchart language elements. The function charts are stored on JGrafchart workspaces, see Fig. 3.1.

Workspaces can be stored to a file and loaded from a file. It is possible to store all top-level workspaces as a single XML file. Workspaces

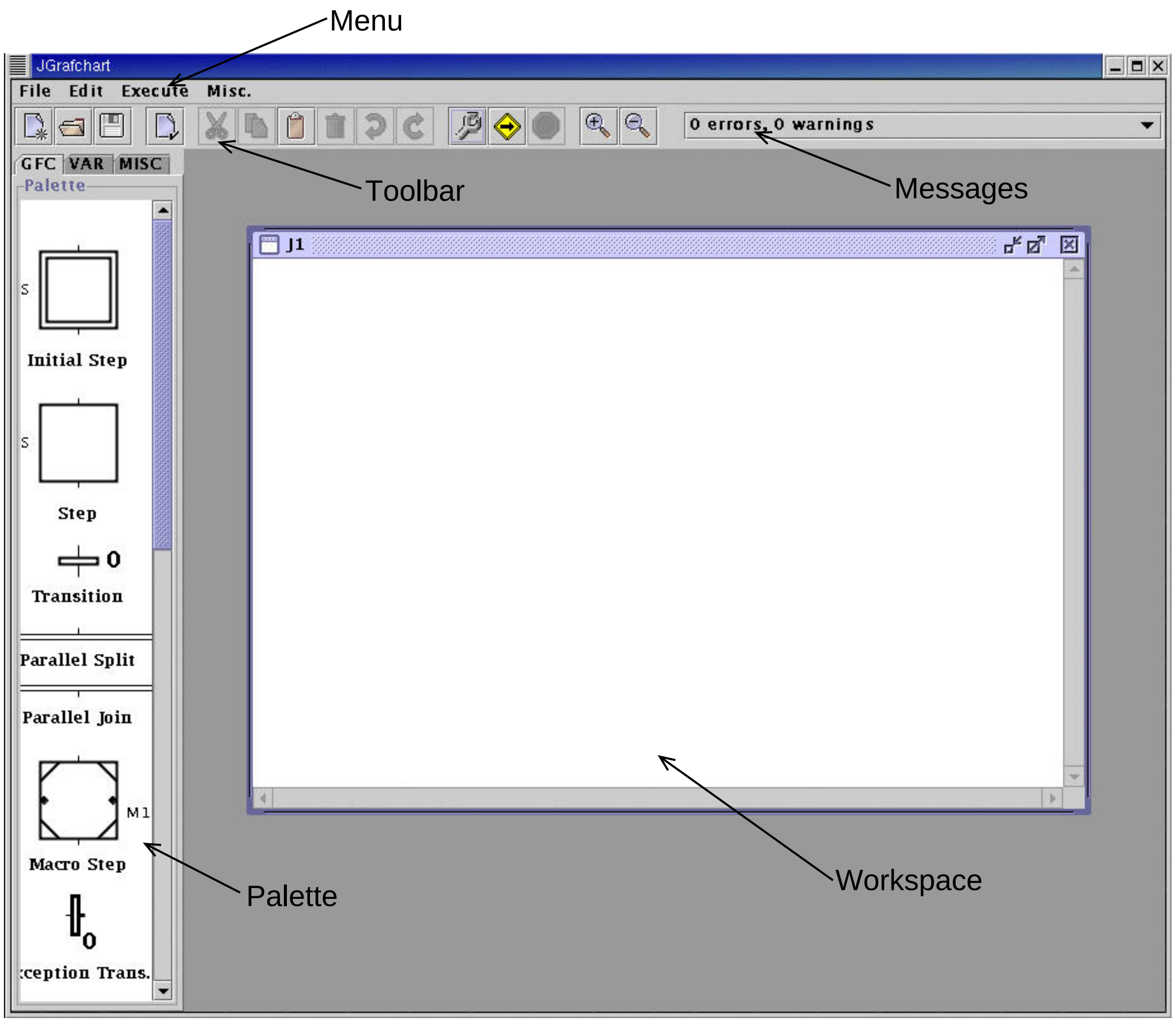


Figure 3.1 JGrafchart, with the palette (left) and a workspace (right).

support scroll, pan, and zoom. It is possible to change their size, to iconize them, *etc*. If multiple workspaces are used, only one of them is the current focus for menu choices. This is indicated through a blue workspace border, rather than the ordinary gray border. The focus is changed by clicking on a workspace. On the workspace it is possible to select an object or an area containing multiple objects in the standard fashion. A selected object can be moved, cut, deleted, or copied to the clipboard. The contents of the clipboard can be pasted to a workspace at the location where the latest mouse-click took place.

Grafchart objects are connected together graphically, to form a function chart, by clicking on a connection port and dragging the connection line to the connecting port of the next Grafchart object. The rules of Grafcet has to be followed: A step cannot be connected to a step, *etc*. The steps and transitions can be edited. Variables and inputs and outputs are defined by dragging and dropping them on a workspace. After the function chart has been built the workspace has to be compiled and if there are no errors the function chart can be executed.

Drawing Capabilities In addition to the Grafchart language elements the editor also supports graphical objects, e.g. rectangles, ellipses, and icons, see Fig. 3.2. The graphical objects can be used to design graphical user interfaces (GUIs). Text comments can be added to a workspace by drag-and-drop of the *Free Text* object on the palette. By single-clicking on the object the text string can be edited. There is also a plotter object, which can be used to plot different variables. Which variables to be plotted and the properties of the axis can be set in the edit menu.

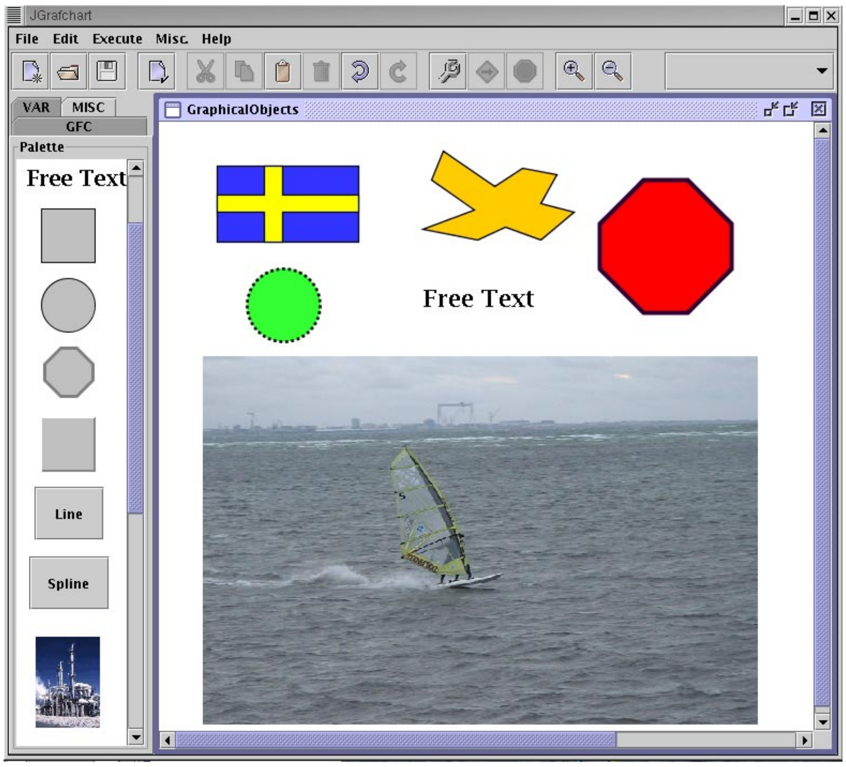


Figure 3.2 Graphical objects[1].

[1]At the bottom the author can be seen windsurfing at Lomma Beach.

Grafchart Elements

JGrafchart supports the following Grafchart elements: steps, initial steps, transitions, parallel splits, parallel joins, macro steps, procedure steps, process steps, enter steps, exit steps, exception transitions, connection posts, workspace objects, step fusion sets, internal variables (real, boolean, string, and integer), and action buttons.

Steps Grafchart steps have action blocks, where the actions of the step is implemented. The action block can be made visible or hidden, see Fig. 3.3. Step actions are entered as text strings. The text string should be ended with a semi-colon. Multiple step actions are separated by semi-colons. Five different action types are supported:

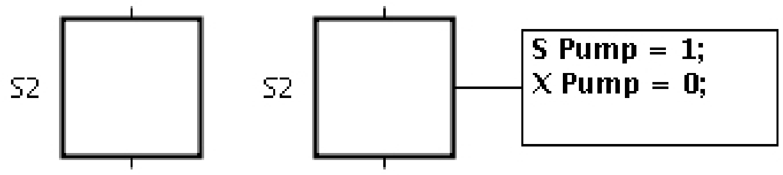


Figure 3.3 A step with the action block hidden and shown.

- **Stored actions** (impulse actions) are executed when the step is activated. A stored action text string starts with 'S'.
- **Periodic actions** (always actions) are executed periodically, once every scan cycle, while the step is active. A periodic action text string starts with 'P'.
- **Exit actions** (finally actions) are executed once, immediately before the step is deactivated. An exit action text string starts with 'E'.
- **Normal actions** (level actions) associate the truth value of a digital output or a boolean variable with the activation status of the step. A normal action text string starts with 'N'.
- **Abortive actions** are executed once when the step is aborted. An abortive action text string starts with 'A'.

Actions can be variable assignments or method calls to objects, such as setting and getting the location, height, and width. Basic math functions such as sin, cos, abs, *etc.* are also available. Basically any Java method can with a small effort be made available in JGrafchart. For the full list of implemented functions consult the documentation of JGrafchart [34] or the *On-Line Help*.

The expression syntax follows the ordinary Java syntax, with some minor exceptions. One important exception is that the literal 0 (1) is used both to represent the boolean literal false (true) and the integer literal 0 (1). The context decides the interpretation.

Expressions may contain references to inputs, outputs, and variables. JGrafchart uses lexical scoping based on workspaces. For example, a variable named `X` on workspace `W1` is different from a variable named `X` 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 `X` on workspace `W2` using `W2.X`.

The abstract grammar for actions is:

action → actionStmt **‘;’** (actionStmt **‘;’**)*

actionStmt → storeStmt | periodicStmt | exitStmt |
        normalStmt | abortStmt

storeStmt → **‘S’** (assignment | methodCall)

periodicStmt → **‘P’** (assignment | methodCall)

exitStmt → **‘X’** (assignment | methodCall)

normalStmt → **‘N’** id

abortStmt → **‘A’** (assignment | methodCall)

assignment → id **‘=’** exp

```
methodCall → id ‘(’ [ argumentList ] ‘)’

argumentList → exp ( ‘,’ exp ) *

exp → exp ‘&’ exp |
        exp ‘|’ exp |
        exp ‘==’ exp |
        exp ‘!=’ exp |
        exp ‘<’ exp |
        exp ‘>’ exp |
        exp ‘<=’ exp |
        exp ‘>=’ exp |
        exp ‘+’ exp |
        exp ‘-’ exp |
        exp ‘*’ exp |
        exp ‘/’ exp |
        ‘-’ exp |
        ‘!’ exp |
        string |
        number |
        methodCall |
        id |
        ‘(’ exp ‘)’

id → ( ‘a’ - ‘z’ | ‘A’ - ‘Z’ ) ( ‘a’ - ‘z’ | ‘A’ - ‘Z’ | ‘_’ | ‘.’ | ‘^’ | ‘0’ - ‘9’ ) *

string → ‘ arbitrary characters ’

number → digit ( digit )*

digit → ‘0’ - ‘9’ | ‘.’
```

Clarifications to the abstract grammar:

- $*$ stands for 0 or multiple occurrences.
- (. . . | . . .) means OR.

- [. . .] means OPTIONAL.

Steps receive names as they are created from the palette. The default name is `'S'+<integer>` (i.e. `S1`, `S2`, *etc*). In most cases the names are unique within the workspace. However, in certain cases, especially involving paste from the clipboard, non-unique names can be obtained. The name can, however, always be changed by click-and-edit.

Initial Steps Initial steps are ordinary steps that become active when the execution of the function chart starts, see Fig. 3.4. Initial steps may have actions in the same way as ordinary steps.

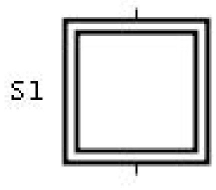


Figure 3.4 Initial step.

Transitions Transitions represent conditions that should be true in order for the Grafchart to change state. The transition expression is represented by a text string associated with the transition, see Fig. 3.5.

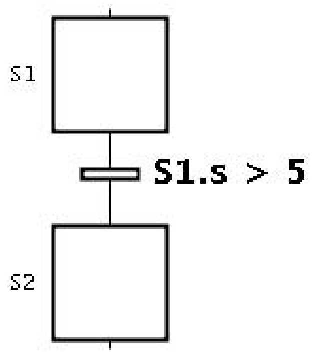


Figure 3.5 Transition with its condition.

The transition expression should return a boolean value. The operators supported in conditions are similar to the operators supported in actions.

Three special step attributes can be used in a transition expression:

- `stepName + ‘.x’`
- `stepName + ‘.t’`
- `stepName + ‘.s’`

The expression `S1.x` returns true if the step `S1` is active and false otherwise. The expression `S1.t` returns the number of scan cycles and `S1.s` returns the absolute time, in seconds, since the step `S1` last was activated. The expression `/a` represents a positive trigger event of a digital input. It is true if the value of the digital input `a` was false in the previous scan cycle and is true in the current cycle. Similarly, the expression `\a` represents a negative trigger event of a digital input. For example, the expression `(/a | \a)` is true whenever the digital input `a` changes its value.

The abstract grammar for transitions is:

condition $\rightarrow$ condexp

condexp $\rightarrow$ condexp **‘&’** condexp |
        condexp **‘|’** condexp |
        condexp **‘==’** condexp |
        condexp **‘!=’** condexp |
        condexp **‘<’** condexp |
        condexp **‘>’** condexp |
        condexp **‘<=’** condexp |
        condexp **‘>=’** condexp |
        condexp **‘+’** condexp |
        condexp **‘-’** condexp |
        condexp **‘*’** condexp |
        condexp **‘/’** condexp |
        ‘-’ condexp |
        ‘!’ condexp |
        ‘\’ id |
        ‘/’ id |

string |
number |
methodCall |
id |
'(' condexp ')'

Parallel Splits and Joins Parallel branches are created and terminated with parallel splits and parallel joins. The parallel objects only allow two parallel branches. If more branches are needed, the parallel elements can be connected in series, see Fig. 3.6.

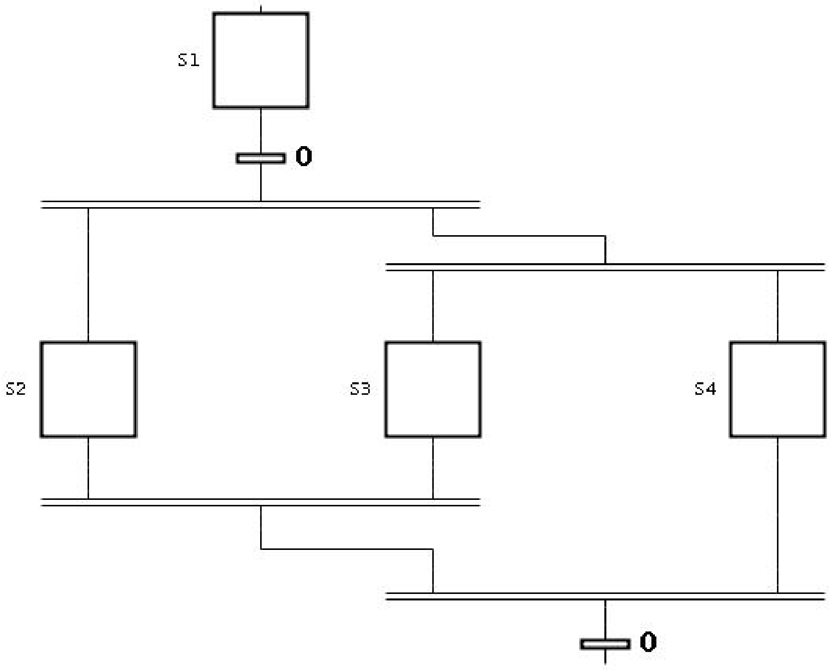


Figure 3.6 Parallel branching with three branches.

Macro Steps A macro step represents a hierarchical abstraction. The macro step contains an internal structure of steps, transitions, and other Grafchart elements represented on a separate (sub-)workspace. The sub-workspace is made visible and hidden by double-clicking on the macro step and using the pop-up 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. Both the enter step and exit step are ordinary steps and may, e.g., have actions. The situation is shown in Fig. 3.7. A macro step itself may have actions, e.g. a stored action of the macro step will be executed before the actions of the enter step are executed and a periodic action of the macro step will be executed all the time while the macro step is active. An exception transition can be connected to the left side of the macro step. For the description of the use see the section on exception transitions.

Macro steps remember their execution state from the time they were aborted and it is possible to resume them from that state by using the history port on the right side.

The sub-workspace of a macro step has a local name-space lexically contained within the name-space of the macro step itself. For example, the sub-workspace of the macro step `M1` may itself contain a macro step named `M1`, without causing any ambiguities. Variables are distinguished by dot-notation.

A macro step with several enter and exit steps is known as a multiple input multiple output (MIMO) macro step, similar to the *superstate* in Statecharts [16]. By adding more than one enter and exit step within the macro step additional ports will be added to the step, see Fig. 3.8.

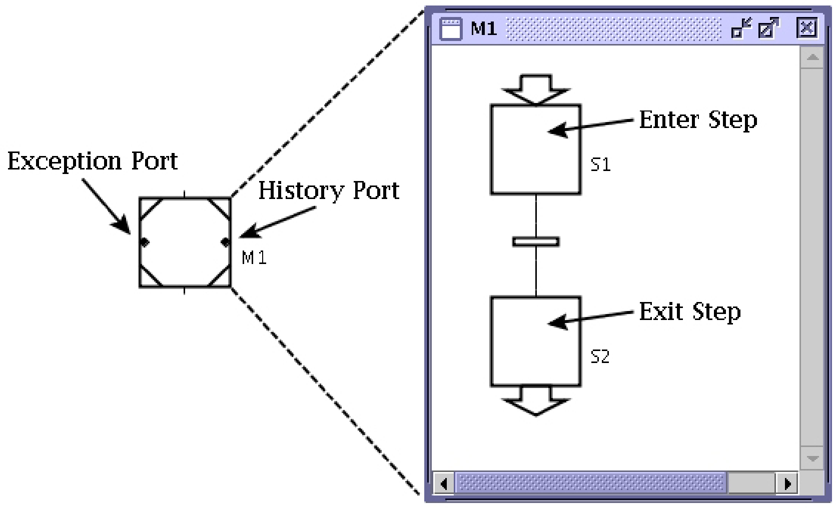


Figure 3.7 Macro step with internal structure.

The MIMO macro step have the same functionality as the *superstate* in Statecharts. An example with a *superstate* (adopted from [16]) is shown in Fig. 3.9. The signals and changes of state are:

- d exit from $J \Rightarrow (B, E)$

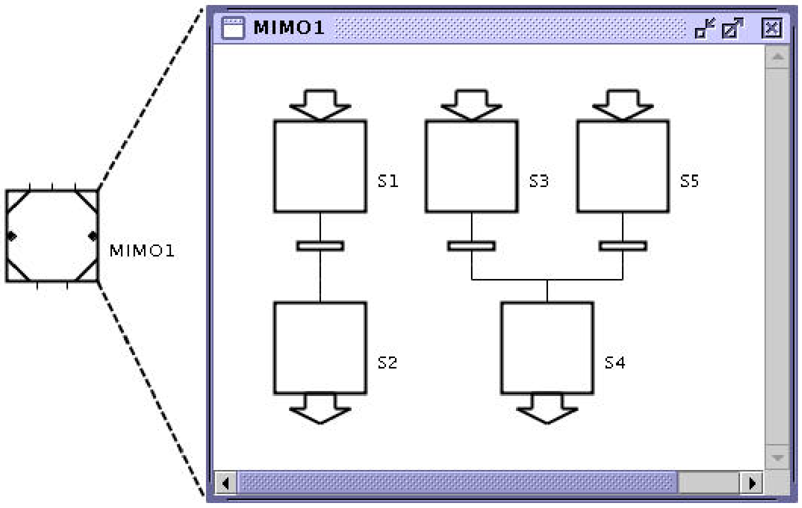


Figure 3.8 MIMO macro step with three enter steps and two exit steps.

- a exit from $K \Rightarrow (C, F)$
- v exit from $J \Rightarrow (B, F)$
- b exit from $L \Rightarrow$ (C,most recently visited state in D)
- w exit from $(B, G) \Rightarrow K$
- n exit from $(B, F) \Rightarrow H$
- t exit from $(C, D) \Rightarrow K$
- e exit from $(A, D) \Rightarrow L$

The example implemented with MIMO macro steps is shown in the Figures 3.10, 3.11, and 3.12. In Fig. 3.11 the connection post `vA1` is connected to `vA2`, `Aw1` is connected to `Aw2`, *etc*. The step `t1`, in Fig. 3.11, is needed because an exception transition is only allowed to be connected to an exception port. Using MIMO macro steps it is possible to implement hierarchical states with the same functionality as in Statecharts, but currently not as user-friendly. With minor modifications the latter could be improved, e.g. support for diagonal connections.

Procedures To reuse sequences in a function chart, a sequence can be placed on the sub-workspace of a procedure, see Fig. 3.13. An enter step indicates the start of the procedure and an exit step indicates the

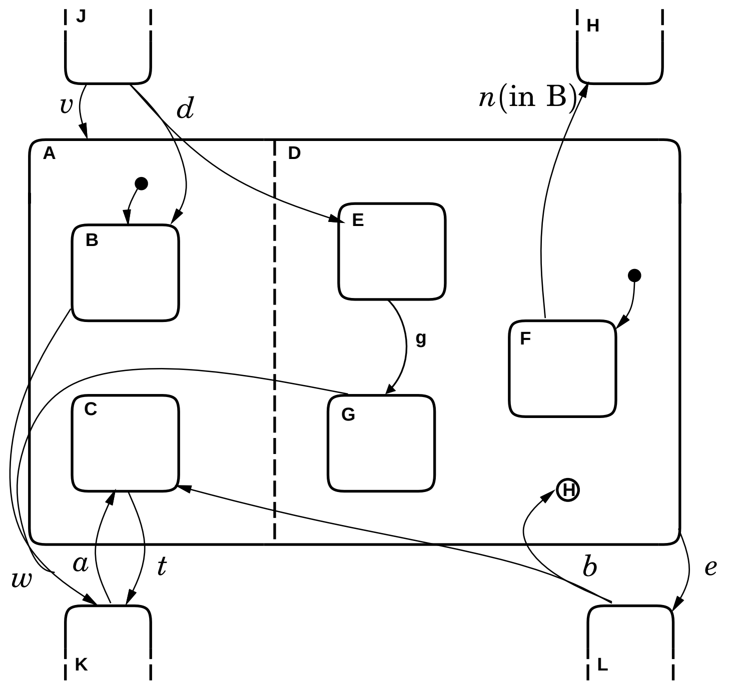


Figure 3.9 Superstate in Statecharts.

end of the procedure. A procedure may have parameters. The parameters can be called by value or by reference. If a parameter is called by reference the procedure can also return values, i.e. the procedure can write to variables external to the procedure. The procedures are reentrant and a copy of the procedure is created for every call. This makes it possible to make recursive calls to a procedure. A procedure can be called in two different ways: from a procedure step or a process step within a sequential function chart, or directly by the user through the GUI.

Procedure Steps A procedure step, Fig. 3.14, is used for calling a procedure, see Fig. 3.15. The procedure step contains a reference to the procedure to be called and either references or values for the procedure parameters.

The syntax for a parameter call by value is:
- `‘V’ id ‘=’ exp ‘;’`

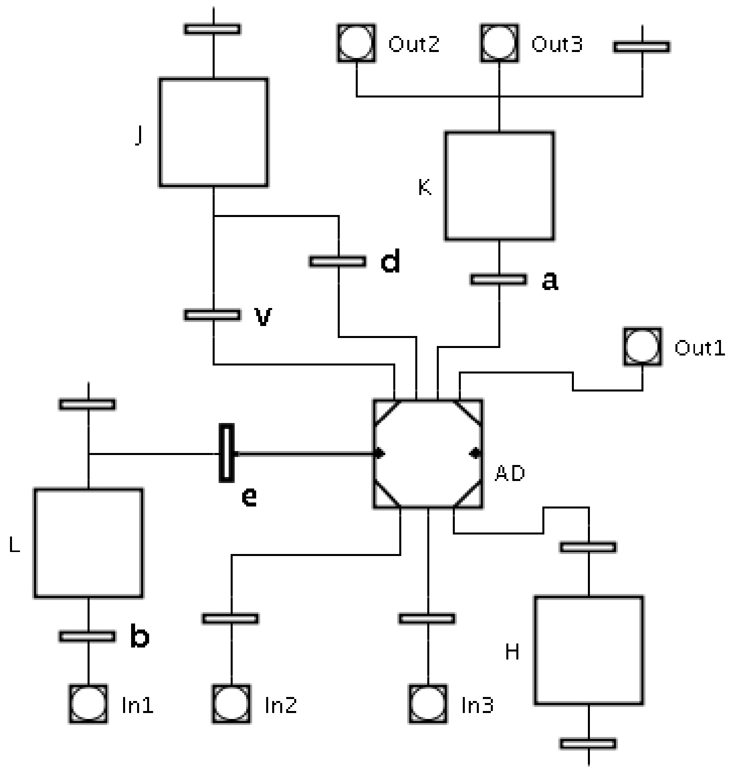


Figure 3.10 MIMO macro step with superstate functionality.

The syntax for a parameter call by reference is:
• `‘R’ id ‘=’ id ‘;’`

When a parameter is called by value the parameter gets the value of the exp. When a parameter is called by reference the procedure can change the value of both the procedure parameter and the variable outside the procedure.

An example of a procedure call from a procedure step, where one variable is called by value and one parameter is called by reference, can be seen in Fig. 3.14–3.16.

A procedure step itself may have actions, e.g. a periodic action of the procedure step will be executed all the time while the procedure step is active. An exception transition can be connected to the left side of the procedure step, see the section on exception transitions.

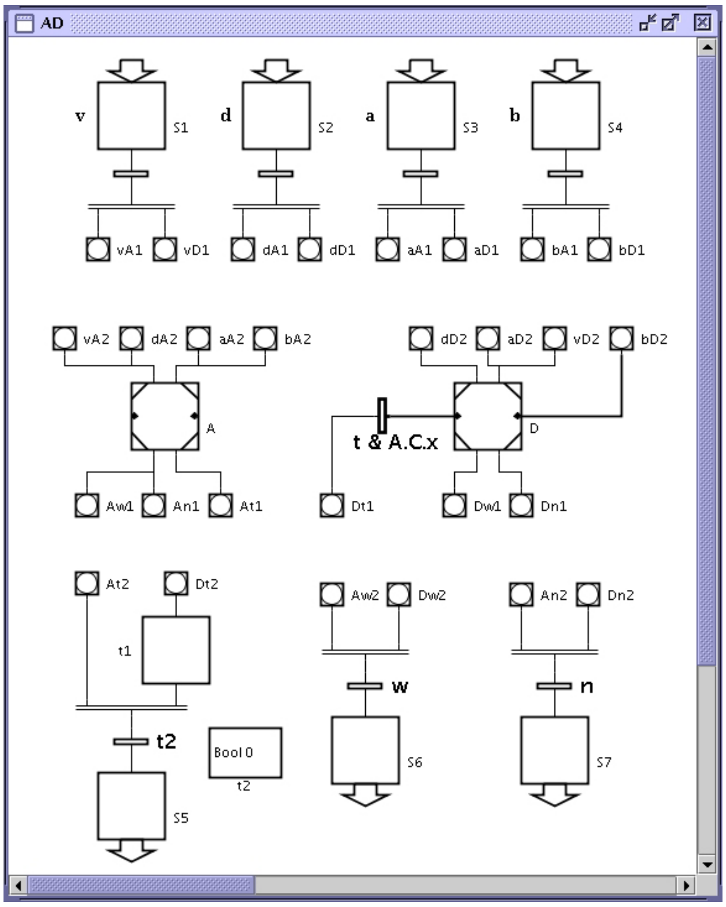


Figure 3.11 The macro step `AD` in Fig 3.10.

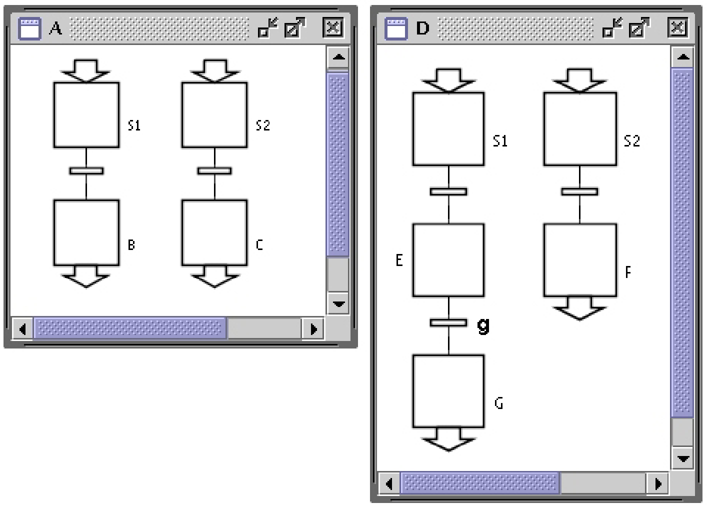


Figure 3.12 The macro steps A and D in Fig. 3.11.

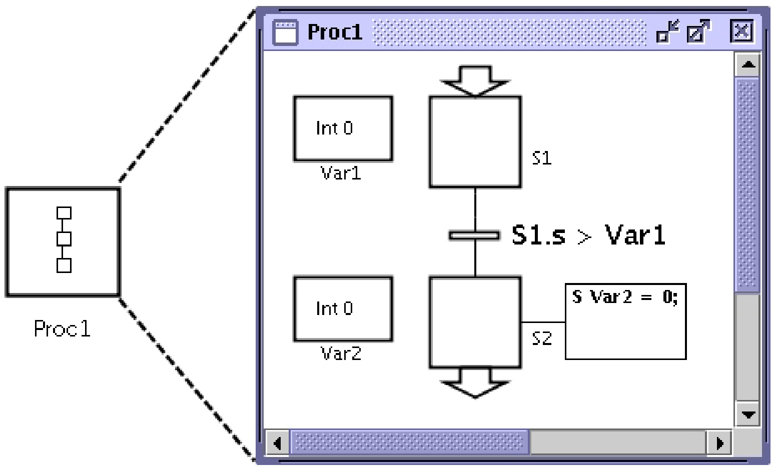
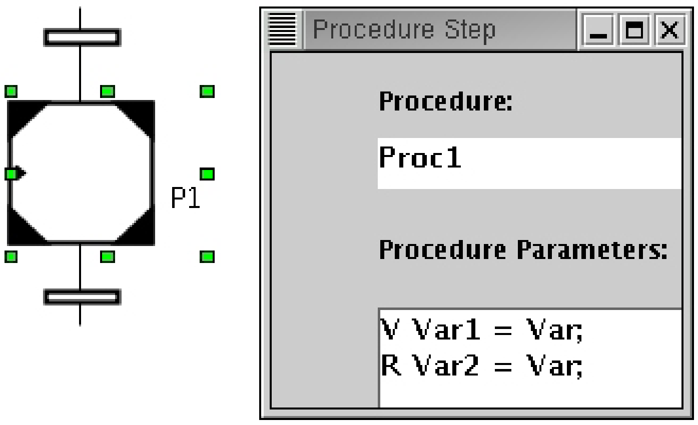


Figure 3.13 A procedure with its sub-workspace.



Proc1

```
V Var1 = Var;
R Var2 = Var;
```

Figure 3.14 A procedure step with a reference to the procedure `Proc1`. The parameter `Var1` in the procedure is called by value, and gets the value of the variable `Var`. The parameter `Var2` in the procedure is called by a reference to the variable `Var`.

Process Steps A process step is similar to a procedure step. It is used for calling a procedure, but it starts the procedure as a separate thread. This means that the procedure will continue to run even though the process step is no longer active. A process step may have more than one process running at the same time. The process step has a list over the procedure calls it has made, see Fig. 3.17. The process step contains a reference to the procedure to be called and either references or values for the procedure parameters just like the procedure step.

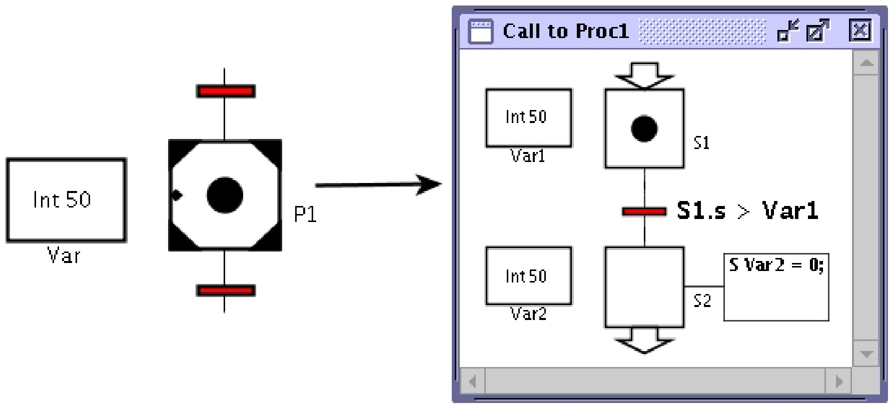


Figure 3.15 In the call to the procedure `Proc1`, both the variable `Var1` and `Var2` gets the value of `Var`.

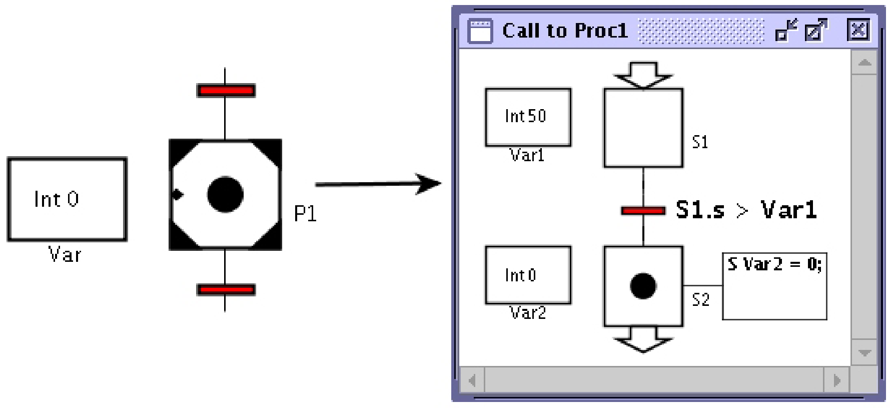
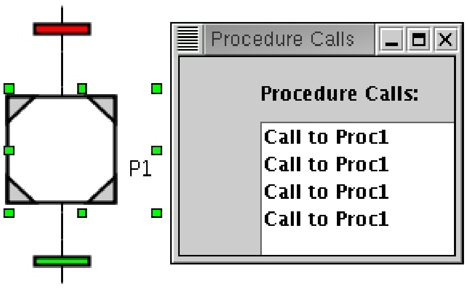


Figure 3.16 The procedure `Proc1` changes the value of the parameter `Var2` and hence the value of the variable `Var` outside the procedure, since the parameter is called by reference.

Exception Transitions An exception transition is a special type of transition that may be connected to a macro step or a procedure step. The exception transition is connected to the left hand side of the macro (procedure) step. An ordinary transition connected to a macro (procedure) step does not become enabled until the execution of the macro (procedure) step has reached the exit step. An exception transition, however, is enabled all the time while the macro (procedure) step is active. When the transition is fired the execution inside the macro (procedure) step is aborted and the step succeeding the exception transition becomes activated. Exception transitions have priority



Call to Proc1
Call to Proc1
Call to Proc1
Call to Proc1

Figure 3.17 A process step with a list of procedure calls.

over ordinary transitions in cases where both transitions are fireable at the same time. An exception transition connected to a macro step is shown in Fig. 3.18. The exception transition will fire when `M1` has been active longer than 5 seconds.

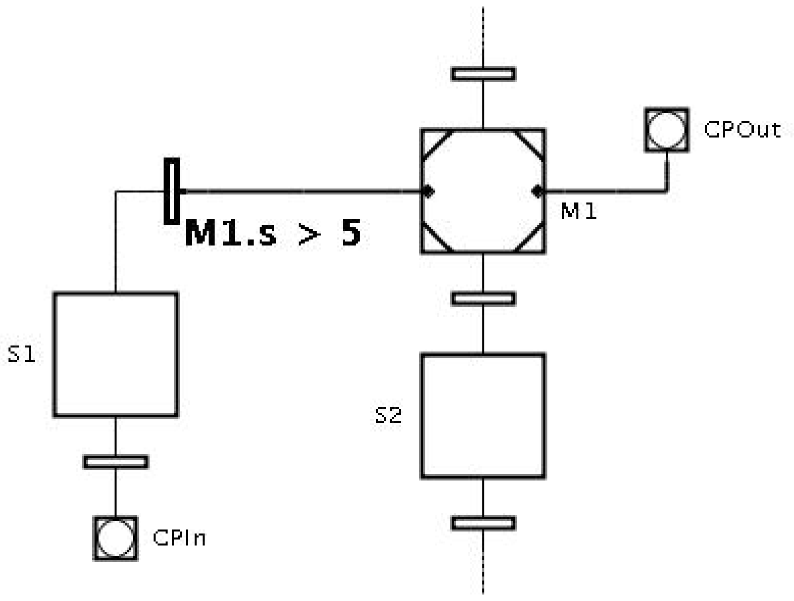


Figure 3.18 An exception transition connected to the macro step `M1`.

Connection Posts Connection posts are used to break the graphical connection between the ports of, e.g., steps and transitions, see Fig. 3.18. In the figure `CPIn` and `CPOut` are logically connected by cross-references. This makes it possible to avoid mixing up connections that run the same path, and to break up sequences and store them on different sub-workspaces in the editor. By placing the mouse pointer on a connection post the logically connected connection post is high-lighted.

Workspace Objects A workspace object can be used to organize programs in JGrafchart. A workspace object on a top-level workspace contains a sub-workspace that can be used in the same way as the top-level workspace, see Fig. 3.19. Using workspace objects it is also possible to create complex variables, e.g. structs, see Fig. 3.20. Workspace objects can also be used to encapsulate procedures. In this way it is possible to mimic simple object structures. A workspace object on the top-level workspace is the object itself. The sub-workspace contains the attributes and procedure (methods) of the object. The attributes can be simple attributes or objects.

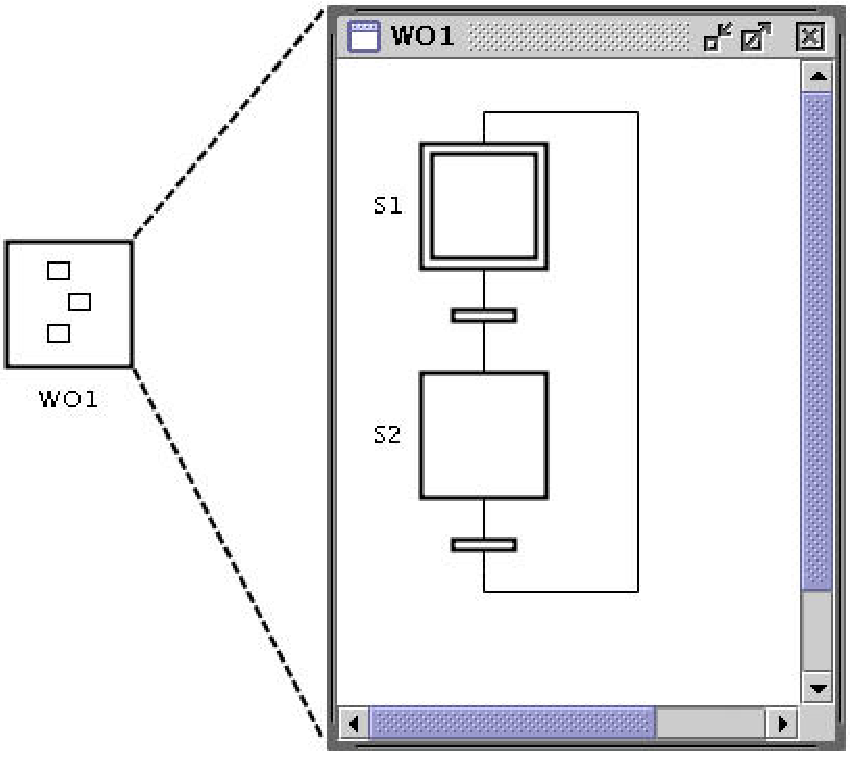


Figure 3.19 A workspace object used as a sub-workspace.

Step Fusion Sets Step fusion sets [24] make it possible to have different graphical representations of the same step. The steps in a

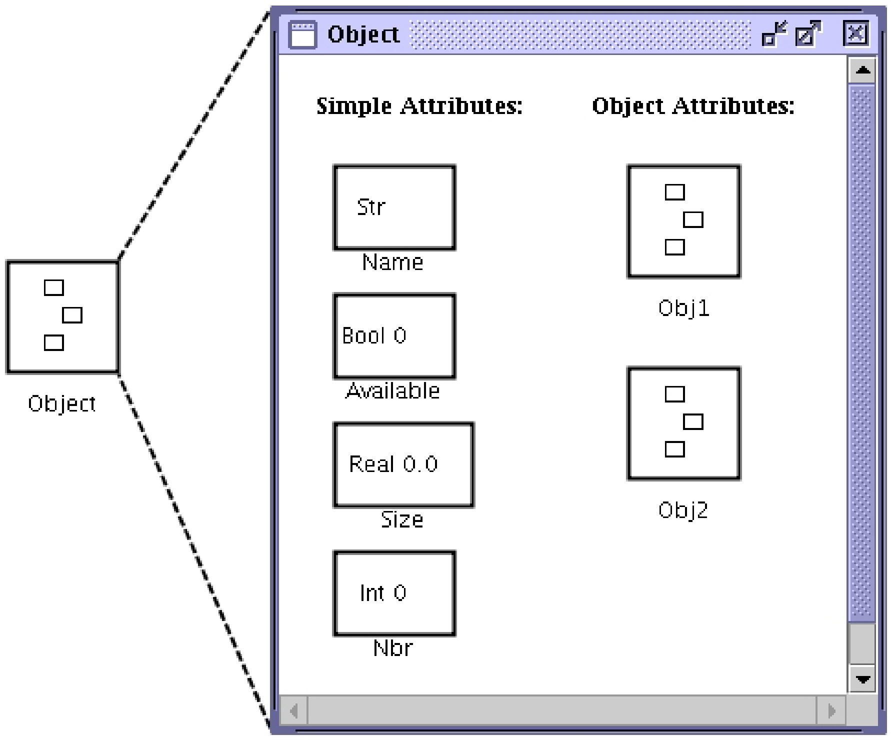


Figure 3.20 A workspace object used as a struct.

step fusion set can be seen as different views of the same step, which are separated and put at different locations as shown in Fig. 3.21. This way sequences can be divided into smaller, easier to read, parts. The steps in a step fusion set do not have to be of the same sort. When one of the steps in the set become active, all the steps in the set become active.

A step fusion set can be either abortive or non-abortive. If the step fusion set is abortive an exit transition of a step becomes enabled when the step reaches its exit step. If the transition condition becomes true the transition fires and all the steps in the step fusion set becomes inactive. Procedure and macro steps will abort their execution and the steps' abortive action will be executed. The abortive actions have to be taken in to account when designing macro steps and procedures.

If the step fusion step is non-abortive all the steps in the step fusion set have to reach their exit step before the exit transitions become enabled.

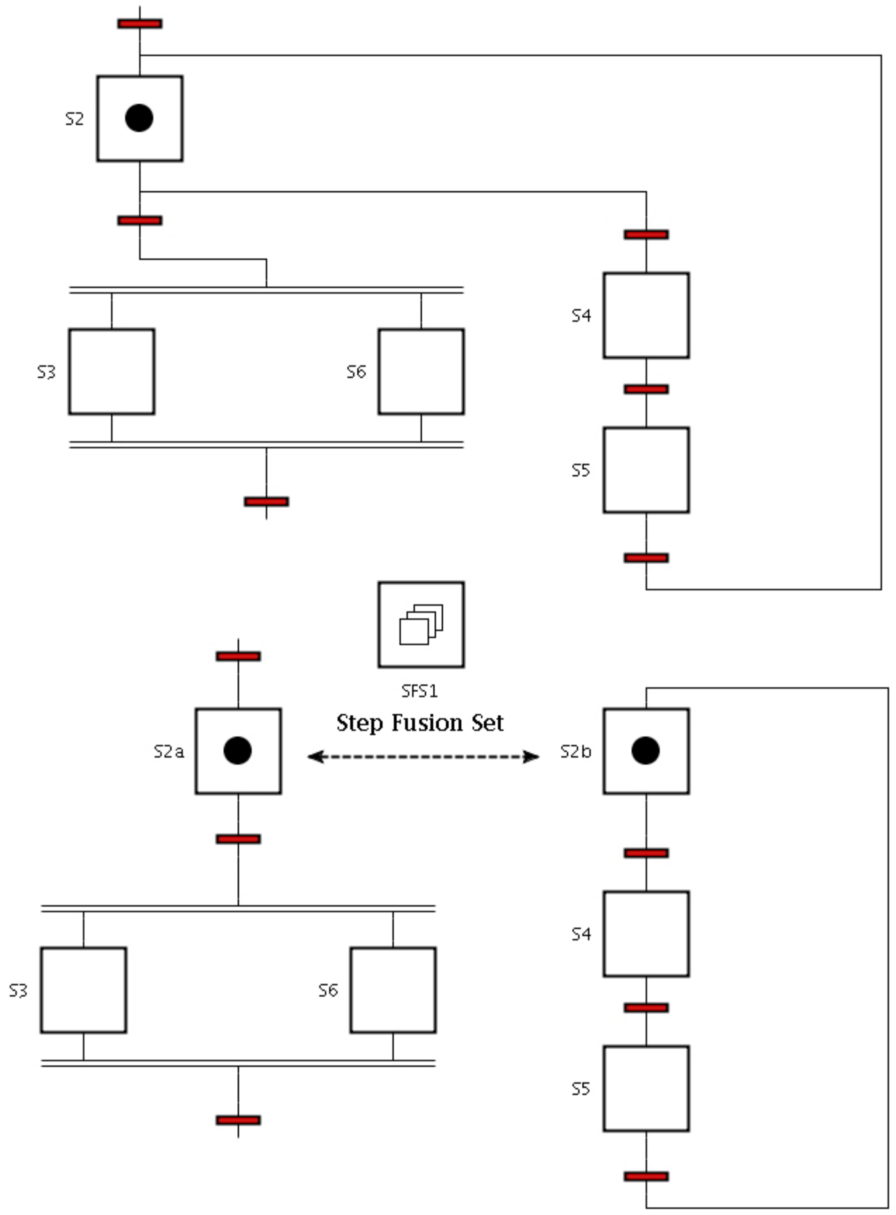


Figure 3.21 Step S2 separated into S2a and S2b using a step fusion set. If the transition after S2b fires S2a and S2b becomes inactive and S4 becomes active. If the transitions after S4 and S4 become true, S2a and S2b become active again.

Internal Variables Internal variables are variables that can be both read from and written to. Four types of variables are available:

- Real
- Boolean

- Integer
- String

A value and a name is associated with each variable, see Fig. 3.22. The value of a variable can be changed by click-and-edit.

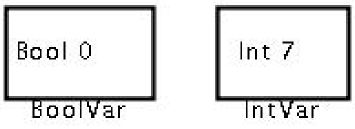
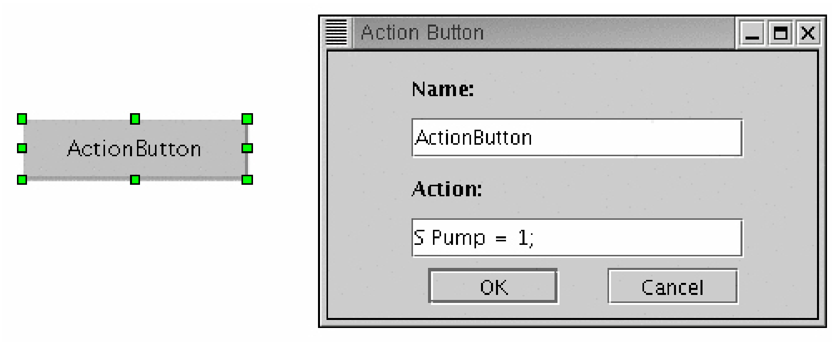


Figure 3.22 Boolean variable (left) and integer variable (right).

String variables can be used as name pointers. The expression '`stringVariable^`' returns the value of the variable named by the string variable.

Action Buttons An action button performs an action when clicked on during execution. The syntax of the action is the same as for stored actions of a step, see Fig. 3.23.



Name:
ActionButton
Action:
S Pump = 1;

Figure 3.23 Action Button with edit frame.

I/O Capabilities

JGrafchart can interact with the external environment in four different ways: using digital I/O, using analog I/O, using sockets, and using XML based communication. Using digital inputs and outputs JGrafchart can be used directly as a logical controller. Using analog input and output blocks JGrafchart can communicate with A-D and

D-A converters. Each top-level workspace can act as a client in a TCP socket connection connecting to a server, e.g. a simulator. Using socket input and output blocks it is possible to read and write variable values using a simple text-based protocol. Finally, using XML [40] input and output blocks, JGrafchart can communicate with xmlBlaster [43], an XML-based message-oriented middleware (MOM) layered on top of XML-RPC or CORBA. XmlBlaster is a publish/subscribe and point to point MOM server, which exchanges messages between xmlBlaster clients.

Digital Inputs and Outputs Digital inputs represent boolean variables that can be read by the steps and transitions in a function chart. Similarly, digital outputs represent boolean variables that can be written to by the step actions in the function chart. The digital outputs can also be read from. Each input and output has an associated value (0 or 1), a name, and a channel number, see Fig. 3.24. Digital inputs have the initial value 0. Two types of digital inputs and outputs exist. One with ordinary logic and one with inverted logic.

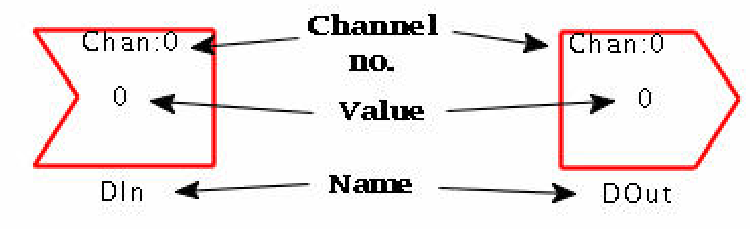


Figure 3.24 Digital input (left) and output (right).

Analog Inputs and Outputs Analog inputs represent real variables that can be read by the steps and transitions in a function chart. Similarly, analog outputs represent variables that can be written to and read from by the step actions in the function chart. Each input and output has an associated value, a name, and a channel number, much like the digital ones in Fig. 3.24.

Socket Inputs and Outputs Socket inputs and outputs can be of four different types: real, boolean, integer, and string. The text-based

protocol for the sockets have the following structure:
`variable-identifier '|' value`. The identifier for the socket object is given by an optional text-string. If no explicit identifier is given, the name of the socket object is used as the identifier. The host and port to connect to are specified in the top-level workspace properties.

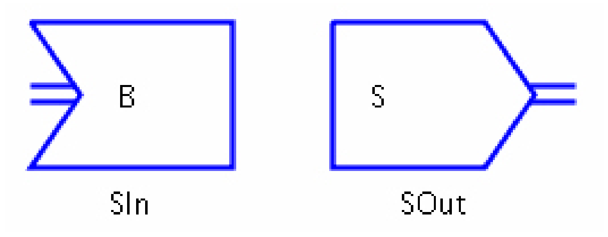


Figure 3.25 Socket input (boolean) and output (string).

XML Inputs and Outputs XML inputs and outputs send and receive XML messages through an xmlBlaster server. The XML inputs and outputs have a sub-workspace containing the elements of the message. The XML inputs and outputs are powerful when sending large data and object structures.

Execution

Grafchart function charts are executed by a periodic thread associated with each top-level workspace. The thread cyclically performs three operations:

1. Read Inputs. The values of the inputs are read.
2. Execute Diagram. All the transitions in the function chart are checked. Steps are activated and deactivated.
3. Write Outputs. The values of the outputs are written.

A Grafchart function chart can be executed in two different modes. In simulated mode the inputs and outputs are only connected to the graphics on the screen. In on-line mode (non-simulated mode), additionally, inputs are read from the different kinds of I/O and outputs

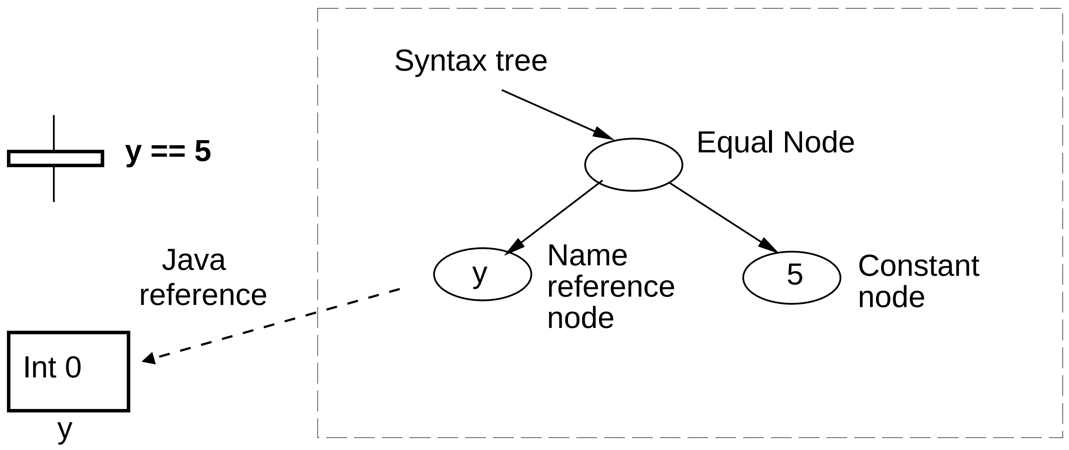


Figure 3.26 Syntax tree for the expression y == 5.

are written to the different kinds of I/O. The execution mode and the thread sleep interval determining the scan rate may be changed using the *Properties* menu choice or by using the toolbar icon.

Before a function chart can be executed it must be compiled. During compilation two things are performed. First, for every transition two lists are built up. One list contains references to all the steps preceding the transition, and one list contains references to all the steps succeeding the transition. Second, the transition expressions and step actions are compiled.

The syntax for transition expressions and step actions are expressed by formal grammars. The parser generator tool JavaCC [42] is used to generate Java parsers for these text expressions. During the parsing, an abstract syntax tree is built up. During compilation the syntax tree is traversed, and all nodes representing name references are replaced by Java references to the corresponding Grafchart objects. The expressions are evaluated on-line, again by traversing the syntax tree. For example, assume that a transition contains the transition expression `y == 5` and that `y` is the name of an integer variable. During parsing the syntax tree in Fig. 3.26 is generated and during compilation the symbol reference is created.

Two types of problems may arise during compilation: parsing errors and symbol table lookup errors. For example, the transition expression `(y OR z)` would generate a parsing error. (The syntactically correct expression should be `(y | z)`). Symbol table lookup errors occur if a name reference does not exist, e.g., if there does not exist any variables named y or z in the previous example. In the editor both parsing errors and symbol table lookup errors are indicated by a change in the text color of the transition expression or step action from black to red. There will also be an error message written in the field on the toolbar, see Fig. 3.1.

During the Execute Diagram part of the execution cycle the following operations are performed. For each transition in the chart, the transition expression is evaluated. If it is false, the transition icon is changed to red. If it is true, the transition icon is changed to green. If, additionally, all steps preceding the transition are active, then the steps preceding the transition is marked to become deactivated in the next cycle, and all the steps succeeding the transition are marked to become activated in the next cycle. When all transitions have been checked, the change of step state is effectuated. In addition to the things above, step actions are executed and the step timing information is updated.

3.3 JGrafchart and Batch Control

Grafchart has been used for batch control recipe handling and resource allocation, see e.g. [27, 28, 26]. Different possibilities for representing recipes and combining recipe execution with resource allocation have been explored using both versions of Grafchart. Grafchart can be used at all levels in the hierarchical procedure model, from the programmable logic controller (PLC) level sequence control to the representation of entire recipes. Grafchart makes it possible to use the same language both at the local control level and at the supervisory control level, see Fig. 3.27.

Physical Model

The physical model in S88 describing the hierarchical relationships between the physical objects involved in batch control can be modeled using workspace objects in JGrafchart. The structure is shown in Fig. 3.28.

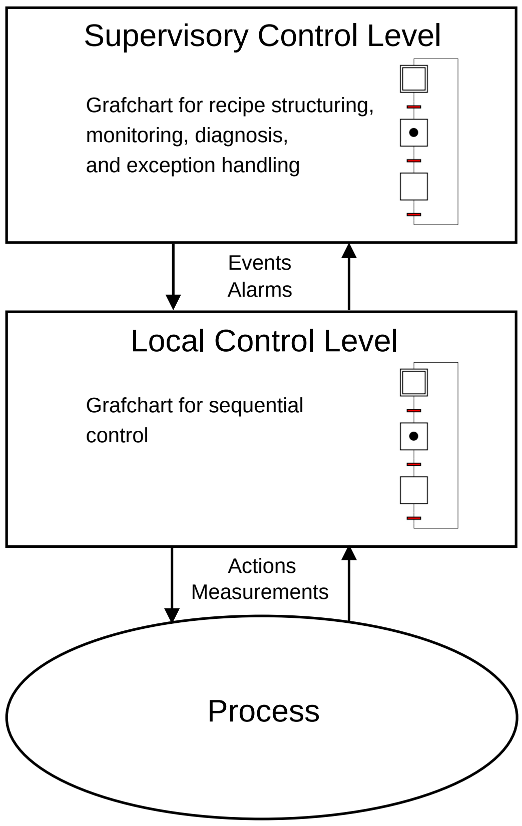

Supervisory Control Level
Grafchart for recipe structuring, monitoring, diagnosis, and exception handling
Events
Alarms
Local Control Level
Grafchart for sequential control
Actions
Measurements
Process

Figure 3.27 Supervision of a sequential process

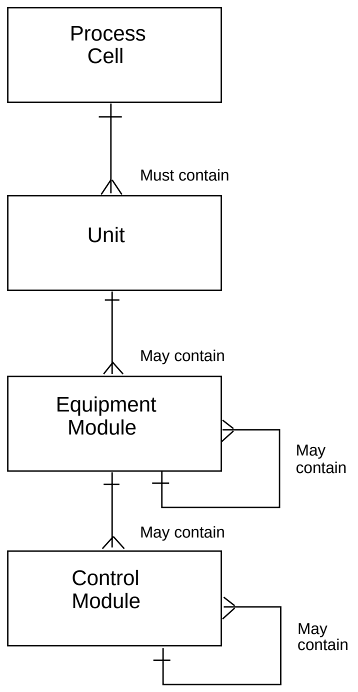
Procedural Model

The hierarchical structure of the S88 procedural model is straightforward to model in Grafchart using macro steps, see Fig. 3.29. Grafchart has had a considerable impact on the definition of Procedure Function Charts (PFC) in S88.02 [10].

Recipes

Recipes can be represented by procedures or workspace objects in JGrafchart. Procedures can be called from procedures or process steps and workspace objects can be copied to become control recipes after completing the recipe with parameters.

The linking between the control recipe and the equipment control is implemented using methods and message passing according to Fig. 3.30. The element in the control recipe where the linking should take place is represented by a procedure step. Depending on at which



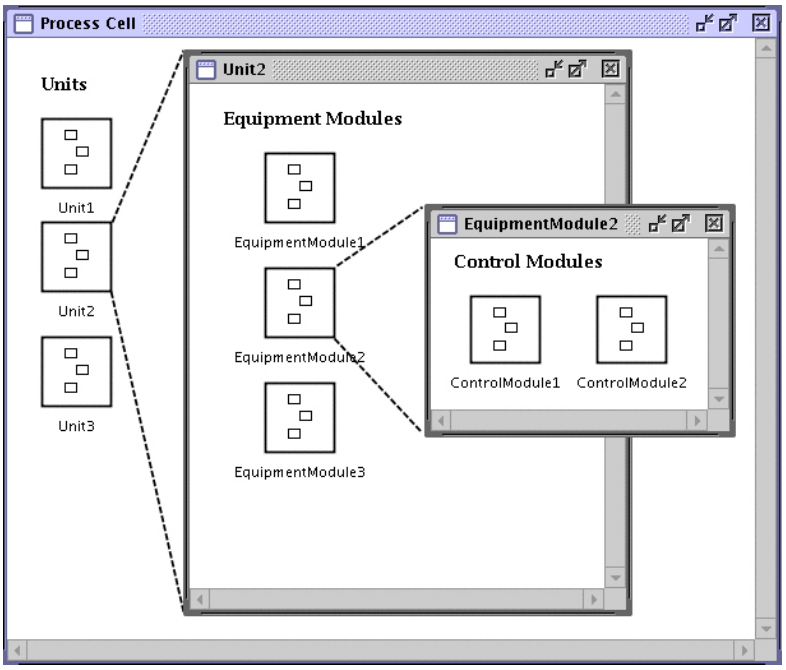


Figure 3.28 The physical model in S88 (left) and JGrafchart (right)

level the linking takes place, the procedure step could represent a recipe procedure, recipe unit procedure, recipe operation or recipe phase. The procedure step calls the corresponding equipment control element, which is stored as a method in the corresponding equipment object.

A number of different ways to represent recipes were proposed in [26] using both the basic and high-level versions of Grafchart. In this thesis only the approach based on basic Grafchart has been used.

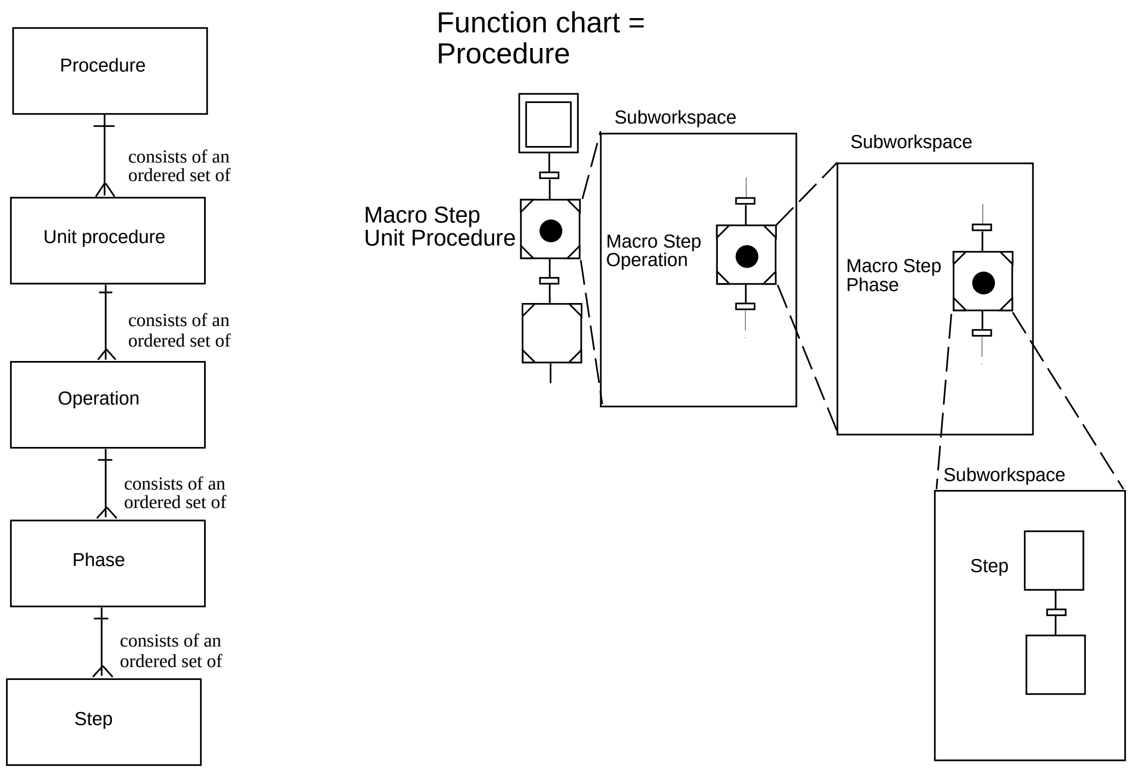

Procedure
consists of an ordered set of
Unit procedure
consists of an ordered set of
Operation
consists of an ordered set of
Phase
consists of an ordered set of
Step
Function chart = Procedure
Macro Step Unit Procedure
Subworkspace
Macro Step Operation
Subworkspace
Macro Step Phase
Subworkspace
Step

Figure 3.29 S88 Procedural Model (left) and its representation in Grafchart (right).

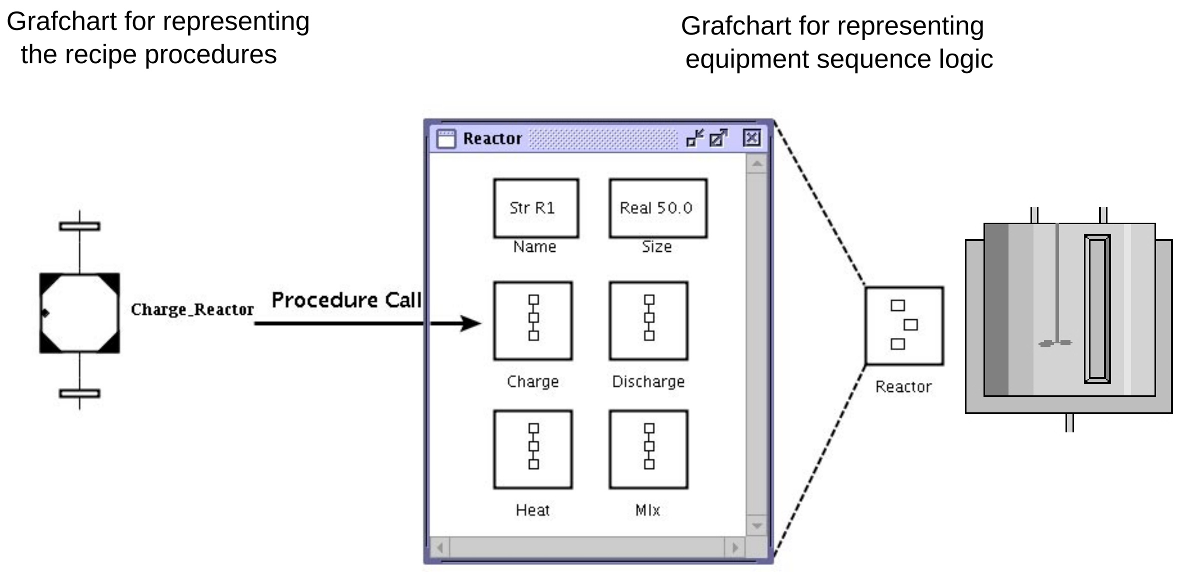

Grafchart for representing the recipe procedures
Grafchart for representing equipment sequence logic
Charge_Reactor
Procedure Call
Reactor
Str R1
Name
Real 50.0
Size
Charge
Discharge
Heat
Mix
Reactor

Figure 3.30 Control Recipe/Equipment Control linking.

4

Exception Handling

4.1 Introduction

An exception is an event that occurs outside the normal or desired behavior of the process. Exception handling is a critical element for achieving long-term success in batch production. It is reported to constitute 40-60 percent of the batch control design and implementation effort [8]. Correct handling of exceptions is a key element in process safety, consistent product quality, and production cost minimization. In short there is money to be made with well structured and automated exception handling.

In this chapter the work on Grafchart for batch process recipe handling and resource allocation is extended to also include exception handling. An internal model approach is proposed where each equipment object is extended with a state machine-based model that is used online to structure and implement the safety interlock logic, and to provide a safety check to ensure that recipe operations are performed in a correct order. The goal is to make the exception handling easier to design and maintain.

As mentioned is Section 2.2 there is no specific safety standard for batch processes, although a number of problems arise especially for these processes. Some of these problems are [39]:

- Separation between the basic process control system (BPCS) and

the safety instrumented system (SIS).
- Synchronization of the process steps between the BPCS and the SIS.
- Operator interaction.
- Implementation of variable (recipe dependent) alarm levels.
- Frequent operational state changes.
- Frequent recipe changes.

How some of these problems can be solved using Grafchart will be discussed in this chapter.

4.2 Unit Supervision

The proposed method for unit supervision is based on augmenting each equipment object (i.e. units, equipment modules, control modules, *etc*) with a finite state machine as shown for the reactor unit in Fig. 4.1. The reactor unit contains three parts: a set of attributes, Grafchart procedures representing equipment unit operations or phases, and the equipment state machine. The attributes could either be attributes of simple types, e.g. max-capacity, or they could be objects, e.g., representing the equipment/control modules within the equipment unit. In the latter case the proposed structure applies recursively, i.e. the equipment/control modules also contain the same three parts.

The equipment state machine is used to model the behavior of the physical object, i.e. there are states for normal operation and there are states for faults. See Fig. 4.2 for the equipment state machine of a valve, with the states `Opening`, `Open_OK`, `Closing`, `Closed_OK`, `Error_Open`, and `Error_Closed`.

The equipment state machine could either be a single automaton modeling, e.g., the behavior of a unit and all its equipment modules, or consist of several smaller parallel automata describing each of the equipment modules in the unit. If the parallel automata are composed they will form the single automaton, see Fig. 4.3. Several smaller parallel automata are probably easier to overview and more user-friendly. This is the approach used in the rest of this chapter. Hierarchical state machines, where each state can contain a whole state machine recursively, can be used to get a better overview of the model.

When using multiple parallel equipment state machines the state

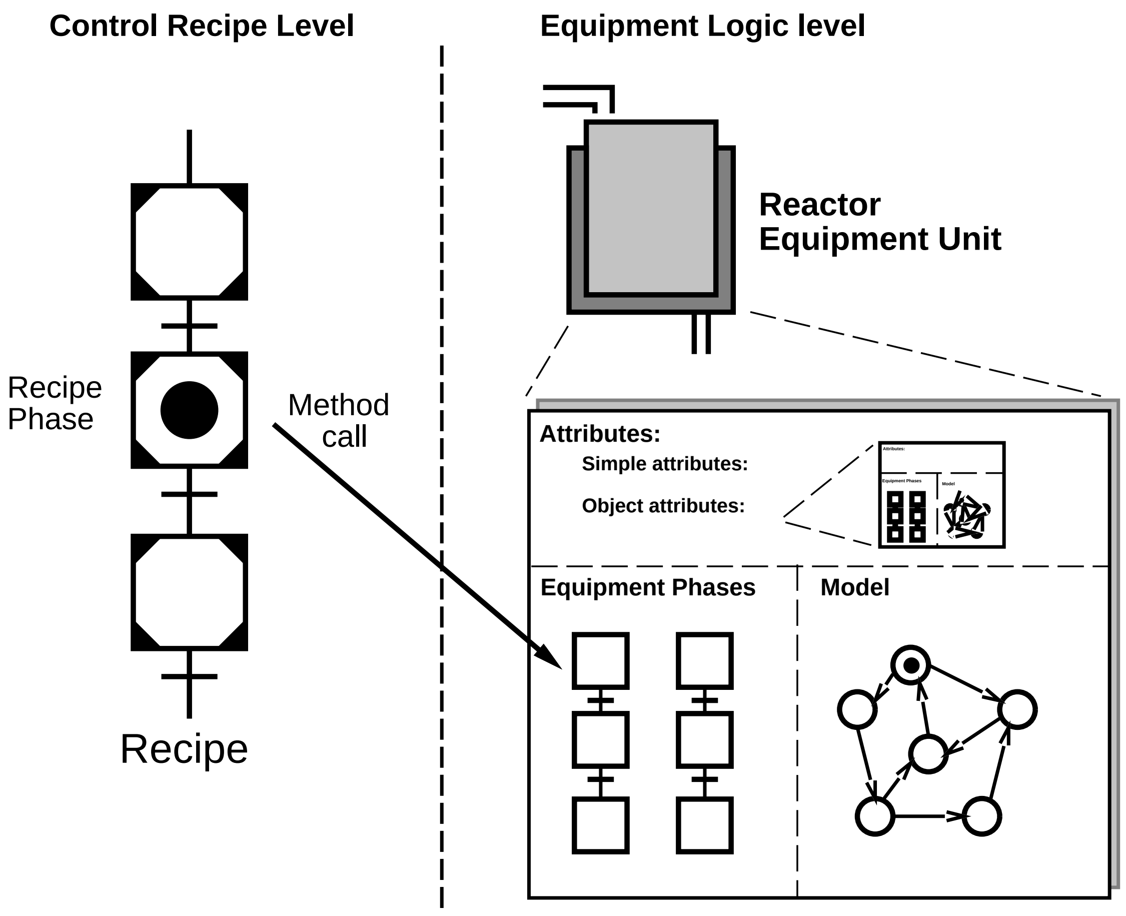


Figure 4.1 Equipment unit with finite state machine.

of an equipment/control module will propagate up to the equipment state machine of the unit, e.g. if a level sensor breaks the state of the unit should go to an error state to indicate there is something wrong within the unit.

The normal execution of an operation causes the equipment state machine to change state, see Fig. 4.4. For example, when the control system sends a signal to a valve to open, the equipment state machine of the valve will go from the state `Closed` to the state `Open`.

The equipment state machine serves two purposes. The first purpose is to be able to check that all the equipment objects are in a consistent state when a method is invoked. For example, it should not be allowed to open a valve if the valve already is open, and it should not be allowed to fill an equipment vessel that is already full.

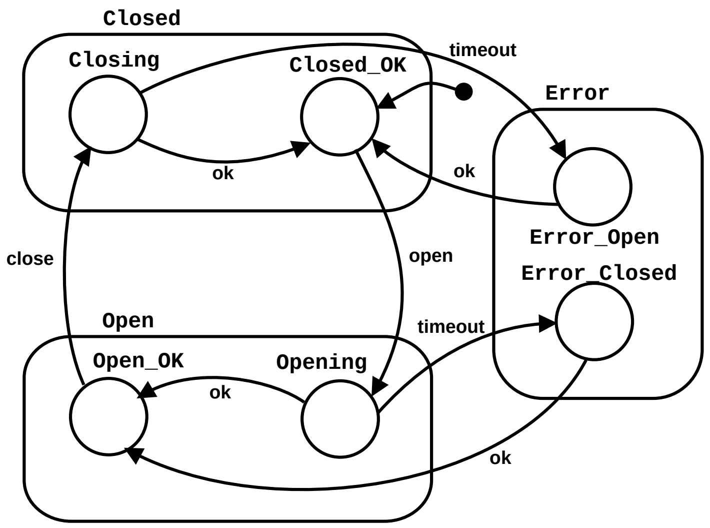


Figure 4.2 The equipment state machine of a valve.

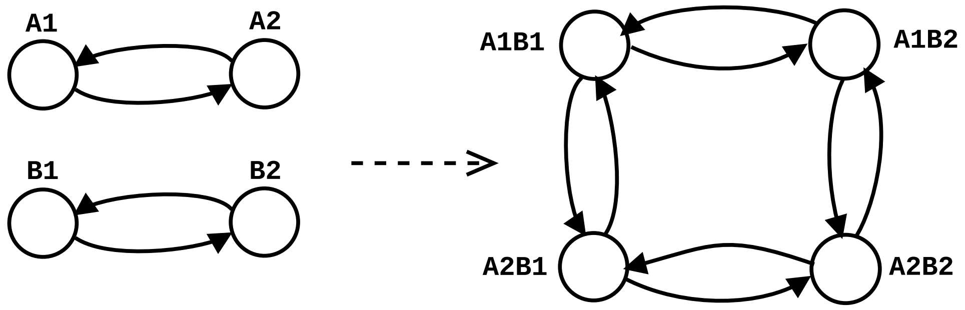


Figure 4.3 Two automata composed to a single automaton.

In a properly designed batch control system, which always executes in automatic mode, one could argue that consistency checking of this type is already performed through off-line validation and verification of recipes, equipment logic, and production schedules. However, in practice batch processes are often run in manual mode for substantial parts of time. Then, it is the operator that manually invokes different equipment phases and a consistency check of the proposed type could be very useful. The consistency check is realized by associating a set of allowed states (or one state if a single state machine is used) with each operation in the equipment control. It is only allowed to start the execution of an operation if the state of the equipment unit belongs to the allowed set of states. The consistency check is implemented as a

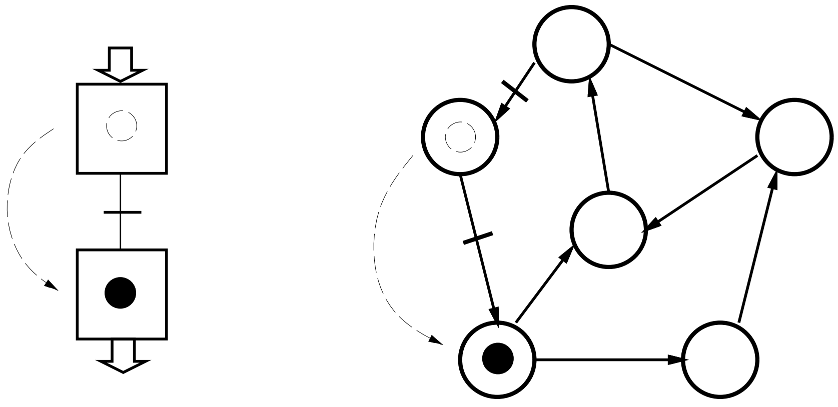


Figure 4.4 The execution of an operation, implemented with a Grafchart procedure, changes the state of an equipment state machine.

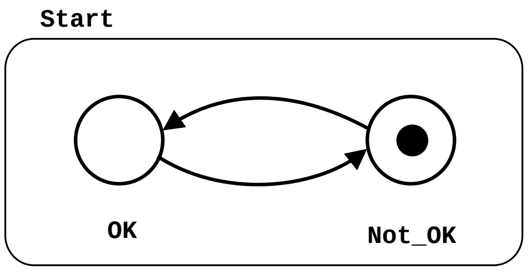


Figure 4.5 The start state machine of an operation for the consistency check for the start of the operation.

start state machine associated with each operation, see Fig. 4.5. In the figure the `Start` state machine is in the `Not_OK` state, which means that the operation, which the start state machine belongs to is not allowed to start if it was called at this instance. For example, if the operation is a `Heat` operation, the reason that it is not allowed to start might be that the temperature of the unit is too high.

The second purpose of the equipment state machine is to provide a structure for organizing the safety and supervision logic at the equipment control level. This is done by implementing the safety logic as transitions or guards in the equipment state machine, as in Fig. 4.6. The safety logic expressed in a transition is only enabled when its preceding state is active. If a fault occurs, the safety logic causes a state transition from a normal state to a fault state. For example, when the valve in Fig. 4.2 receives a signal from the control system to open, the state changes to `Opening`. The error transitions of this state will

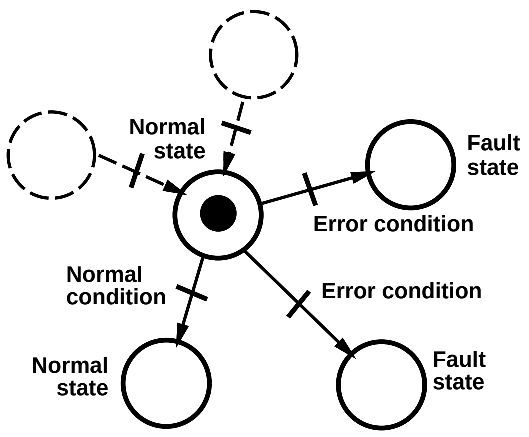


Figure 4.6 State machine with safety and supervision logic.

become active. One of the transition conditions is that the valve does not respond to the signal within a given time, and sends back a signal that it is physically open, the equipment state machine of the valve will go to the `Error_Closed` state, representing that the valve is stuck in the closed position. There is a large difference if the valve fails to respond to a command when it is in the `Open` state compared to if the same problem occurs when it is in the `Closed` state. Which is the most severe state depends on the process, e.g. if the valve is for cooling water a failure in the `Closed` state is probably worse than a failure in the `Open` state.

The state machines can be implemented in several ways. In the implementation in this thesis multiple input multiple output (MIMO) macro steps have been used, see Fig. 4.7. Using the MIMO functionality of the macro step it is possible and convenient to model hierarchical state machines of the proposed type in JGrafchart.

A hierarchical state machine is convenient when modeling the errors of an equipment object. For example, when producing a product in a reactor it might be important not to exceed a given temperature to maintain quality. The reactor tank have hard constraints on what its operating range is, e.g. maximum pressure and temperature. This results in different levels of severity of exceptions. A bad product is not nearly as important as the risk of causing human injury. The different error states would typically result in different alarms to the operator. If there is a risk that the quality will go out of the specifica-

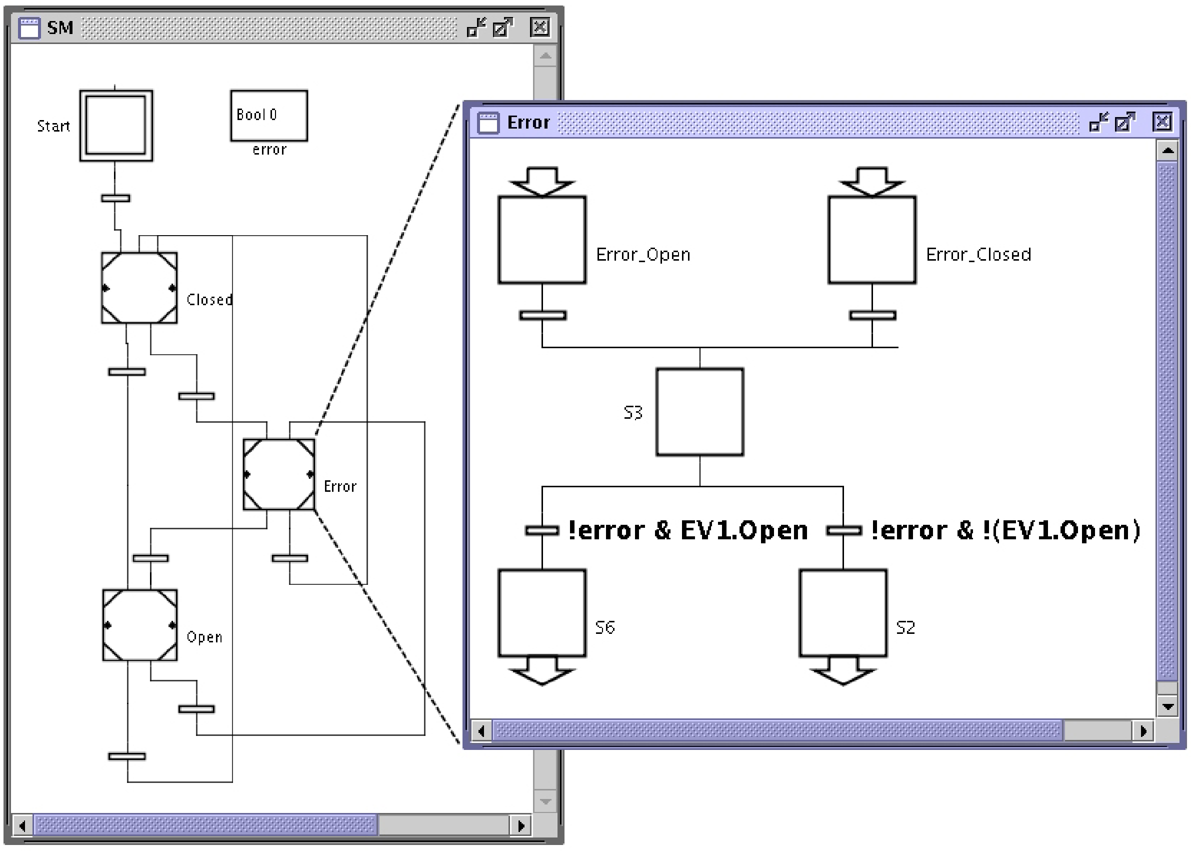


Figure 4.7 Equipment state machine of a valve implemented in JGrafchart using MIMO macro steps.

tions there will be a warning to the operator, but if the unit is going to a safety-critical state the unit needs to be taken to a safe state by the exception handling. The equipment state machine of a temperature sensor would typically look as in Fig. 4.8. The `High` state of the state machine contains the two states `Quality` and `Danger`, which corresponds to the states described above. The same state machine for the temperature sensor implemented in JGrafchart is shown in Fig. 4.9. The conditions in the transitions of the state machines are dependent on the recipe. How the conditions can be set is described in Section 4.3. The equipment state machine of a temperature sensor may also have error states, but they are not part of this example.

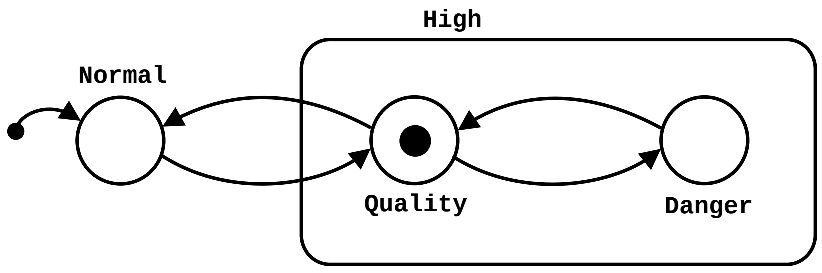


Figure 4.8 Equipment state machine of a temperature sensor

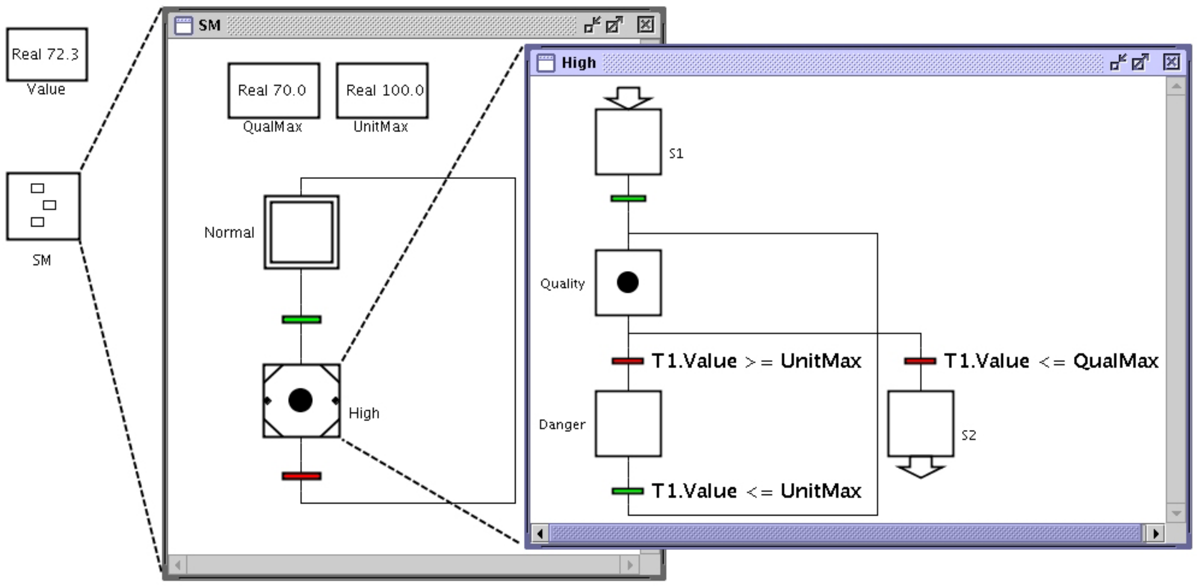


Figure 4.9 Equipment state machine of a temperature sensor (T1) implemented in JGrafchart using MIMO macro step.

Exception Handling Structure

In the proposed structure for exception handling most of the functionality is associated with the equipment operations. Each equipment operation, e.g. `Charge`, `Heat`, and `Clean`, is a workspace object on the workspace of a unit in JGrafchart, see Fig 4.10.

Each operation workspace object contains a procedure (i.e. the sequential control), the procedure state machine describing the procedural state, the start state machine (for the consistency check), and an exception handling workspace object, see Fig 4.11.

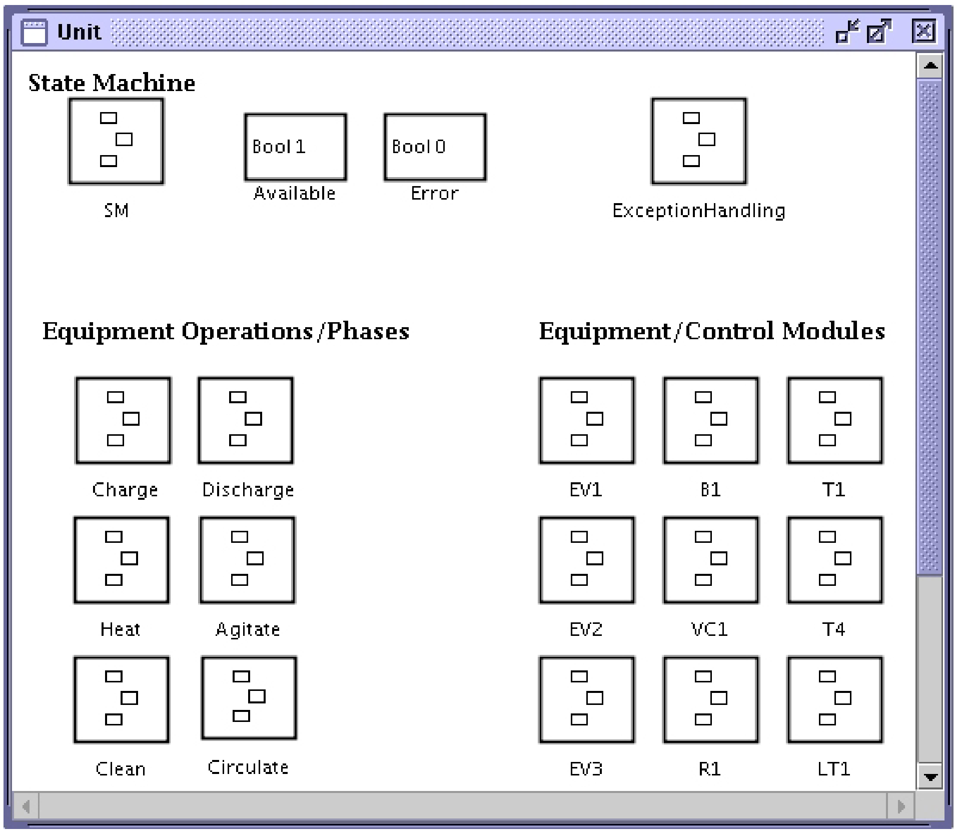


Figure 4.10 A workspace of a unit with equipment state machine, equipment operations/phases, equipment/control modules, and unit exception handling.

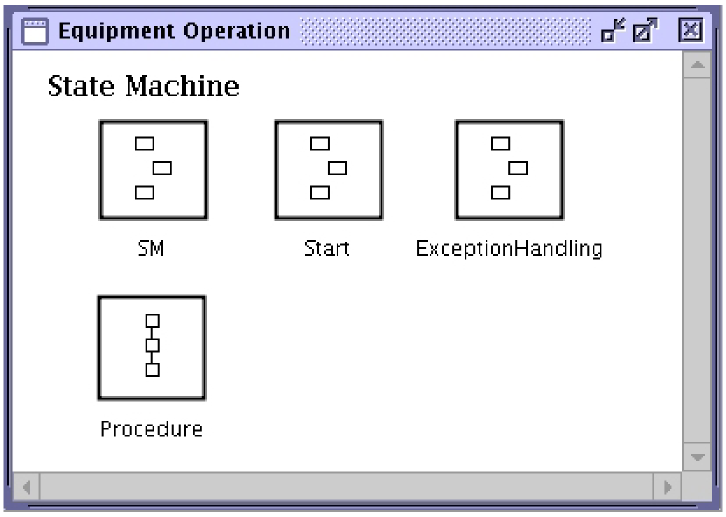


Figure 4.11 Equipment operation workspace.

The procedure of an equipment operation holds not only the equipment sequential control, but also contains several checks, which need to be performed when a procedure is called from a recipe. First it checks if the unit is available (in S88 only one operation at a time is allowed to be active in a unit). It reserves the unit if it is available and then checks if the procedure itself is in the `Idle` state and if so changes the state to `Running`. The check if the unit is in a consistent state at the start of the operation is also checked here by using the start state machine. This could be implemented in several ways, one way is to implement it in JGrafchart according to Fig 4.12. In the figure the unit's name is `U1`, the operation called is `Charge`, the signal to reserve the unit is `reserve`, `SM` is the name of the equipment state machine describing the state of the unit, `LT1` is a level sensor, `EV1` is a valve, and the check if the unit is in a consistent state when starting the operation is a start state machine named `Start`. The implementation of the start state machine in JGrafchart is shown in Fig 4.13. The start state machine consists of only two steps, `OK` and `Not_OK`. The two states are mutually exclusive and if the `OK` state is active the operation is allowed to start. In this small example the unit is not allowed to be full and the out-valve (`EV1`) is not allowed to be open for the operation `Charge` to start.

The procedure state machine of the operation can be implemented in the same way as the state machine of an equipment object using MIMO macro steps. The procedure state machine can be used by an operator to, e.g., pause the execution of the operation or just as a display object to depict in which state an operation is. The following twelve procedural states from S88 and Fig 2.4 are modeled in Fig 4.14: idle, running, complete, pausing, paused, holding, held, restarting, stopping, stopped, aborting, and aborted.

In this implementation a large part of the exception handling is specific to an operation. The exception handling workspace of the operation holds both the recipe level, and the equipment level exception handling logic, see Fig 4.15. Both of these would be running in the Safety Instrumented System in a control system implementation for a real plant. The recipe level exception handling logic will be discussed in Section 4.3.

In the equipment level exception handling logic, two types of operations can be identified:

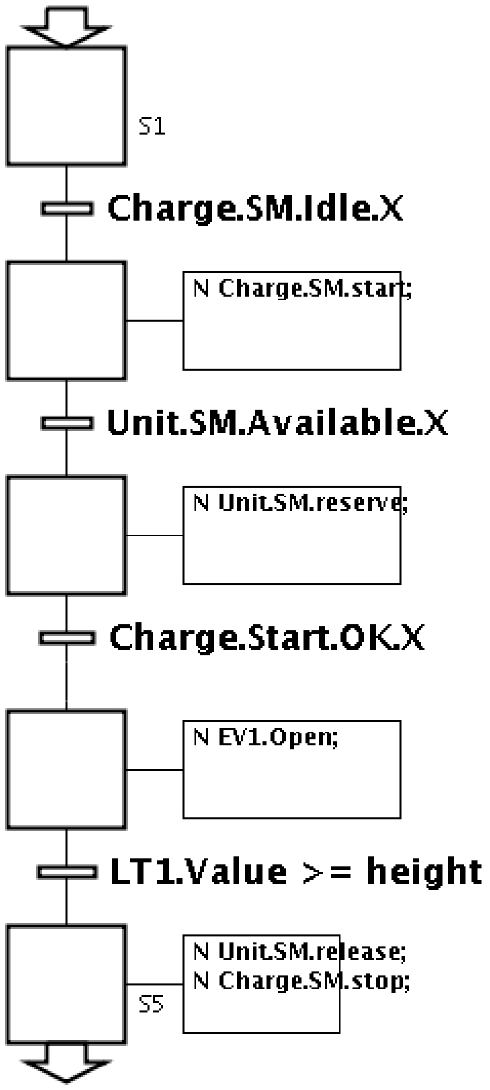


Figure 4.12 Checks and control in the `Charge` operation implemented in JGrafchart. The procedure checks if the state of the operations is `Idle`, changes the state of the state of the operation to running by sending the signal start. It reserves the unit if it is available, checks if the start of the operation is allowed by the state of the unit and then opens the in-valve `EV1`. When the level is above the specified height the valve is closed and the unit released.

- Detection logic, based on the equipment state machines of the equipment/control modules.
- Logic to handle the exceptions.

Both kinds of operations will be called from the recipe level excep-

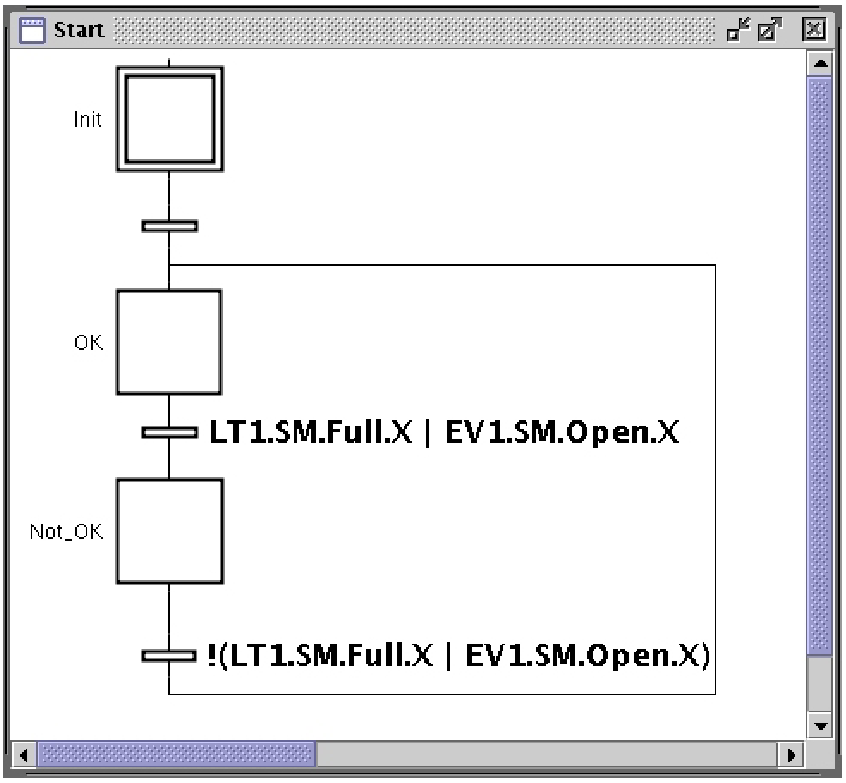

Start
Init
OK
LT1.SM.Full.X | EV1.SM.Open.X
Not_OK
!(LT1.SM.Full.X | EV1.SM.Open.X)

Figure 4.13 The start state machine of the `Charge` operation. The two states important to the start of this operation is that the unit is not full and that the in-valve (`EV1`) is not already open.

tion handling logic. The detection logic, `Detection` in this example, checks the state of the unit by looking at either only the unit's equipment state machine, or at the equipment objects' individual equipment state machines depending on how the states of the equipment objects propagate to the unit. One way to implement the propagation of exceptions is to let the unit's equipment state machine only consist of the states `OK` and `Error`, and let the detection logic be trigged by the unit's equipment state machine, see Fig. 4.16.

The operations to handle the exceptions in Fig. 4.16 are:

- `StartExc` - the sequence, which takes the unit back to a state where the operation is allowed to start.

- `Emergency` - emergency shut-down (could be the same for all operations in a unit).

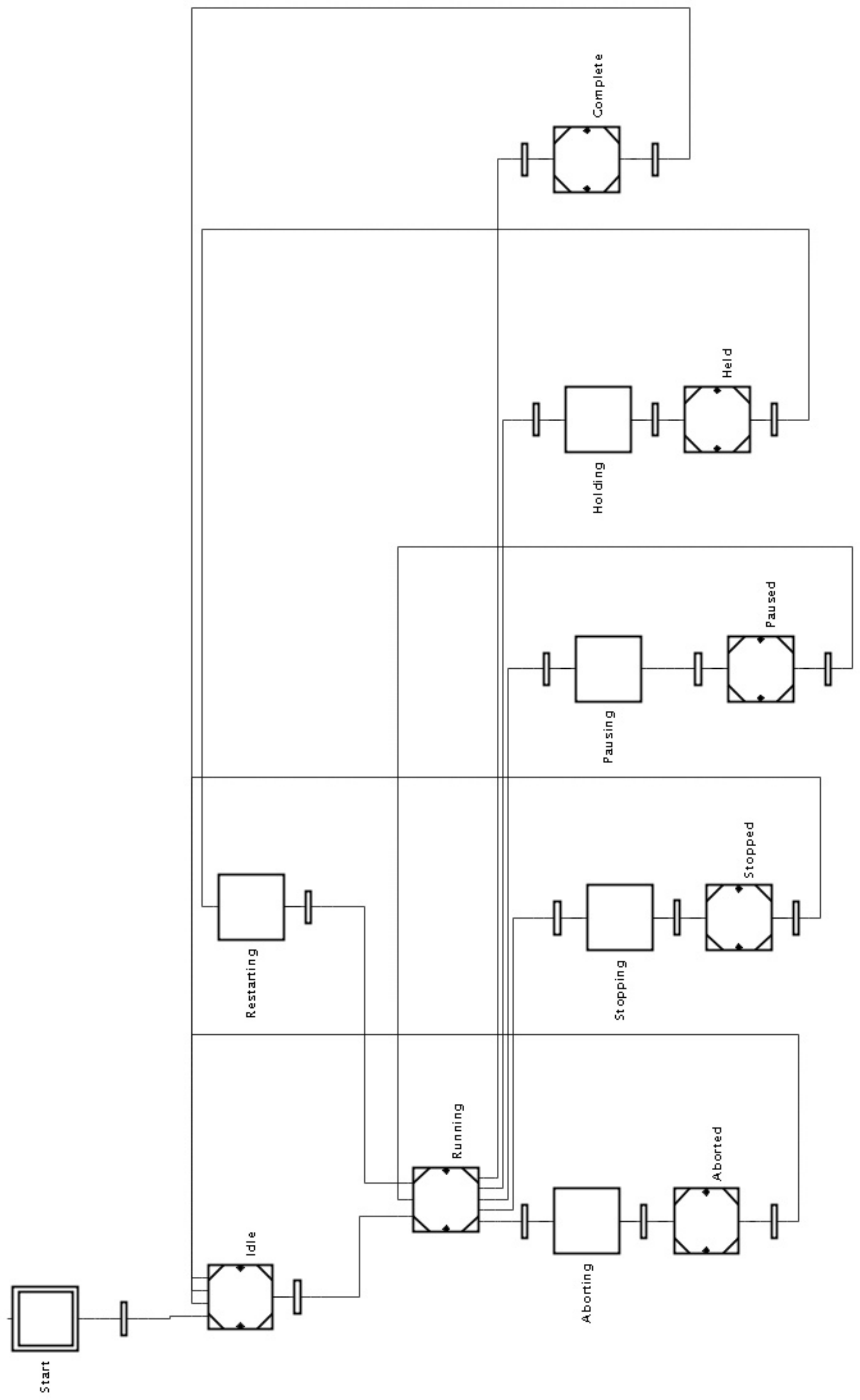


Figure 4.14 The states of an equipment operation modeled with MIMO macro steps.

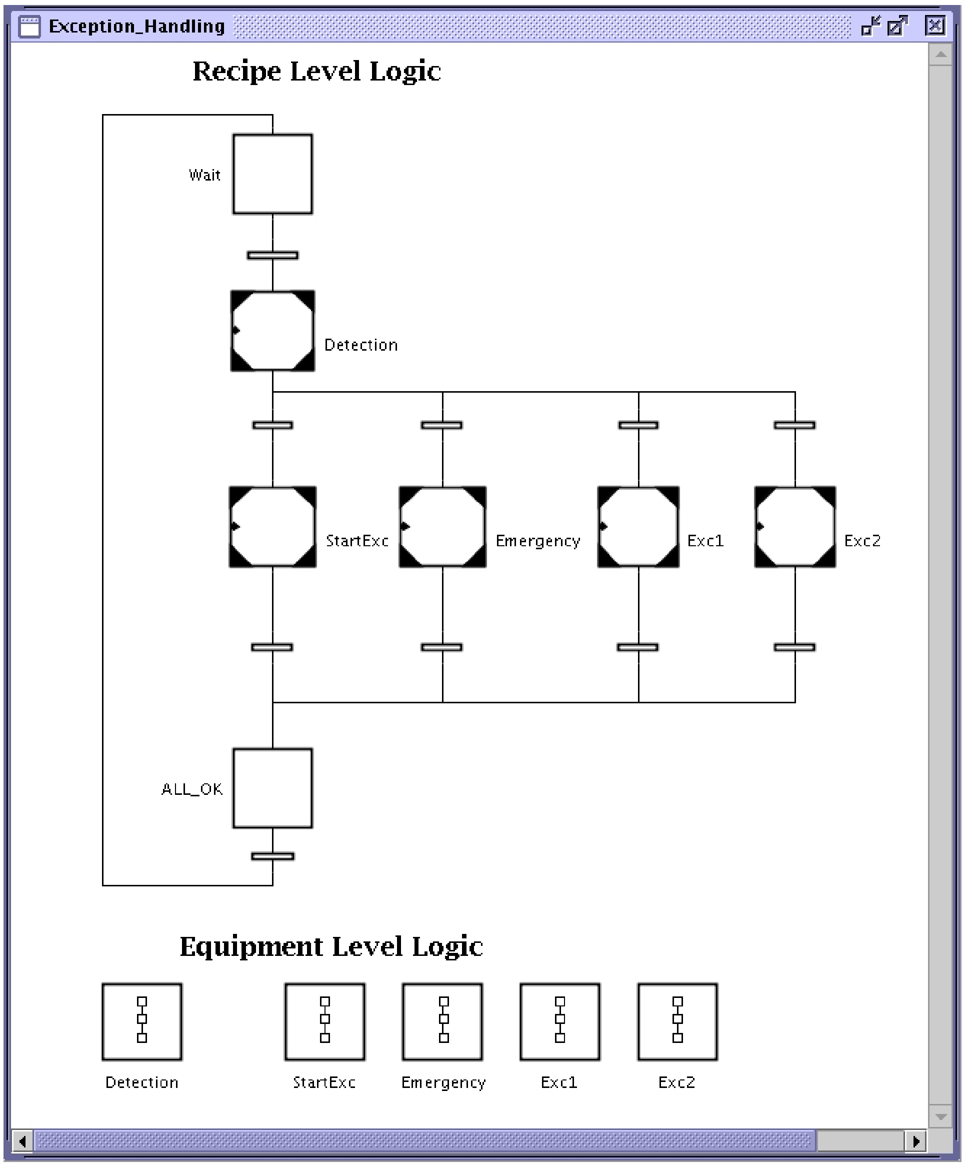


Figure 4.15 Exception handling associated with an operation.

- `Exc1-Exc2` - handling of exceptions 1 and 2, which can be any exceptions specified.

The unit exception handling is running all the time, see Fig. 4.17. There is, e.g., exception detection logic checking the state of the equipment modules even though there is not any operation running in the unit. The unit-specific exception handling and the operation-specific exception handling logic need to be synchronized to avoid false alarms.

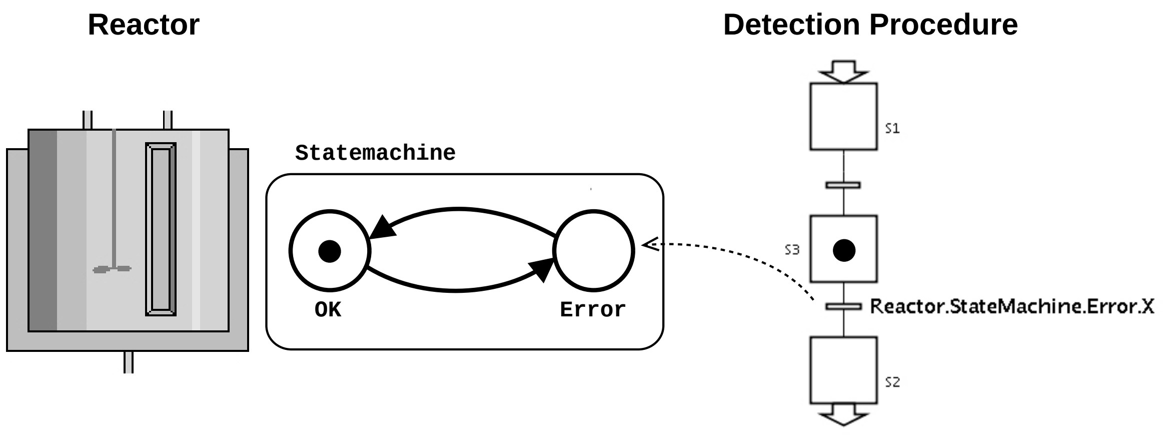


Figure 4.16 Detection logic of an operation controlling a reactor.

4.3 Recipe Level Exception Handling

In the proposed approach the main responsibility for fault detection and exception handling lies at the equipment control level. However, exception handling is also needed at the recipe level. For example, an exception that has occurred must be fed back to the control recipe, recorded in the batch report, and appropriate actions must be taken.

An important consideration is how to separate the recipe information from the exception handling logic and operations. If the latter is included in the recipe, it becomes difficult to develop, maintain, and use. The exception handling would probably be larger than the recipe itself. Grafchart provides several features that simplify the representation of exception handling logic at the recipe level.

It is possible to use exception transitions for recipe level exception handling. The exception transitions are connected to the procedure steps representing the the control recipe operations, see Fig. 4.18. At least two extra graphical objects, an exception transition and a connection post, are needed for each procedure step in the control recipe. The structure for the recipe level exception handling is lost and it is spread out in the control recipe.

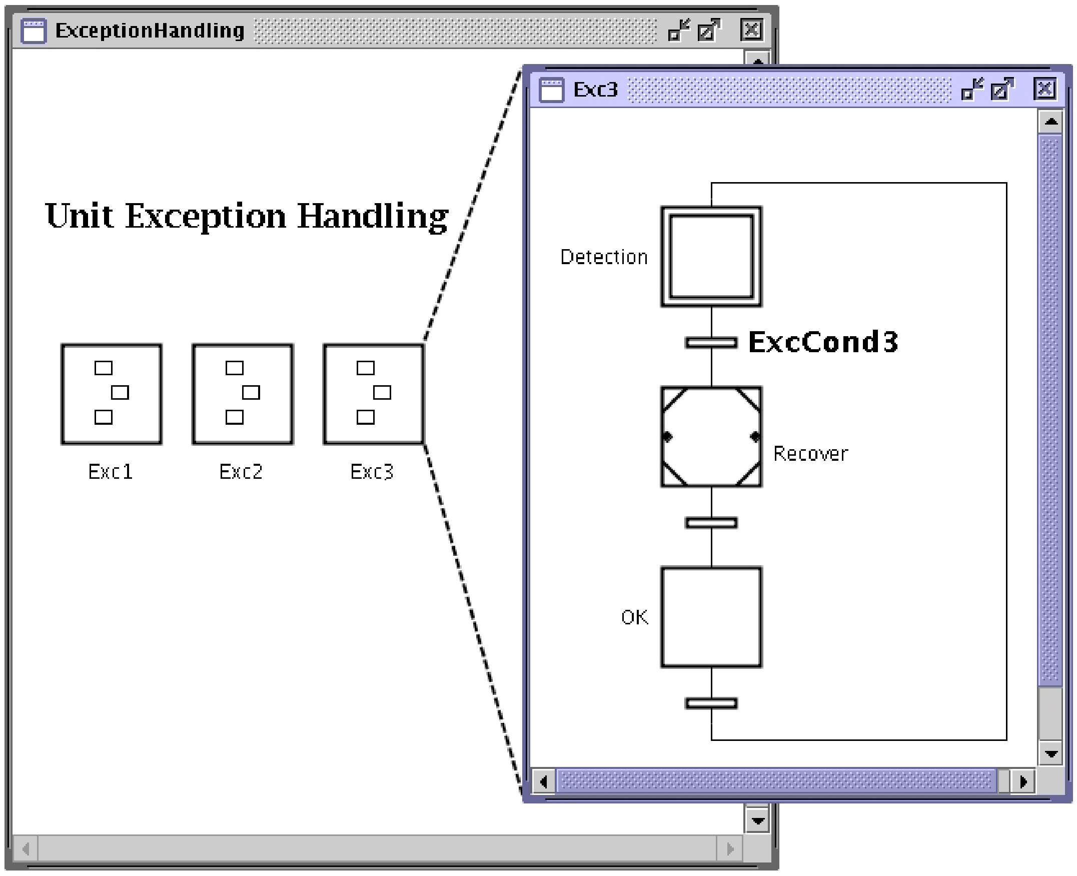


Figure 4.17 Unit detection and exception handling logic. Three different workspace object for the exceptions Exc1-Exc3 are shown. ExcCond3 is the condition that triggers the exception handling for Exc3. The detection is based on the equipment state machines for the equipment/control modules in the unit.

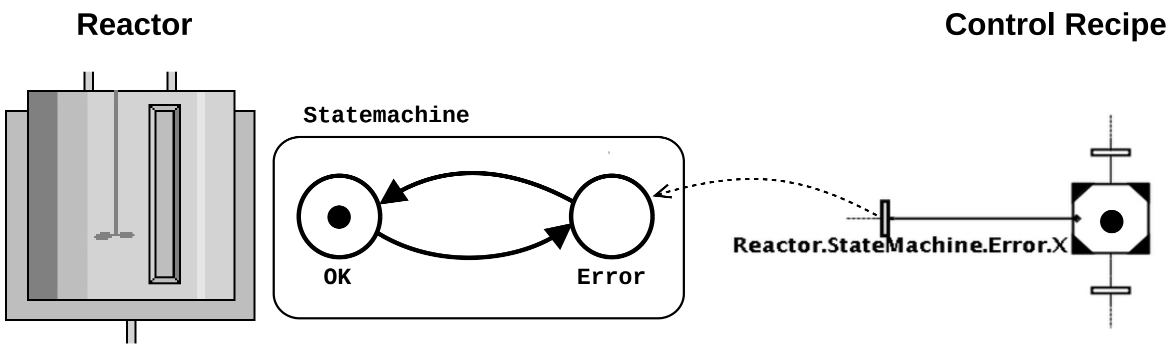


Figure 4.18 Exception transitions for recipe level exception handling.

Another possibility, to avoid the extra graphical objects in the operation, is to use *step fusion sets* [24]. The approach is to have the procedure step that calls an operation in the control recipe be in the same step fusion set as the procedure step that calls the detection logic of the operation, see Fig 4.15.

Consider a control recipe consisting of a sequence of procedure steps (`I`) making procedure calls to different equipment operations (`II`), see Fig. 4.19. The transition after the procedure step in the control recipe (`III`) becomes enabled when the execution of the corresponding operation is finished. The procedure step in the control recipe has a corresponding procedure step in the recipe level exception handling logic (`IV`), these two steps are in the same step fusion set. The procedure step in the recipe level exception handling logic calls the detection operation at the equipment level (`V`). If an exception occurs before the operation in the control recipe is finished, the equipment level exception handling logic detects it. The detection of an exception enables the transitions connected to the out-port of the detection procedure step (`VI Error Exits`). The transition associated with the specific exception that has occurred becomes true, and the operation for the handling of the specific exception starts. If the step fusion set is abortive the execution of the operation called from the control recipe is aborted and the abortive actions of the procedure step in the control recipe are executed.

The first two error exits would typically be the exit for emergency shutdown of the unit (`VII`) and the exit for when the starting state is not a member of the allowed starting states (`VIII`). Other error exits would be for the malfunction of a valve, a sensor, or any other equipment belonging to the unit. A default operation, which takes the unit to a fail-safe state if an unspecified exception occurs should also be implemented. The operations would generate the operator alarms and other information during the execution of the exception handling logic. The operations for the exception handling may be automatic, semi-automatic, or manual.

The nature of the actions that must be taken depends on the application. In a few very special cases it might be possible to "undo" an operation and rollback the execution of the recipe to a safe execution point, and from there continue the execution using, e.g., a new unit. This is similar to the check-pointing and rollback employed in

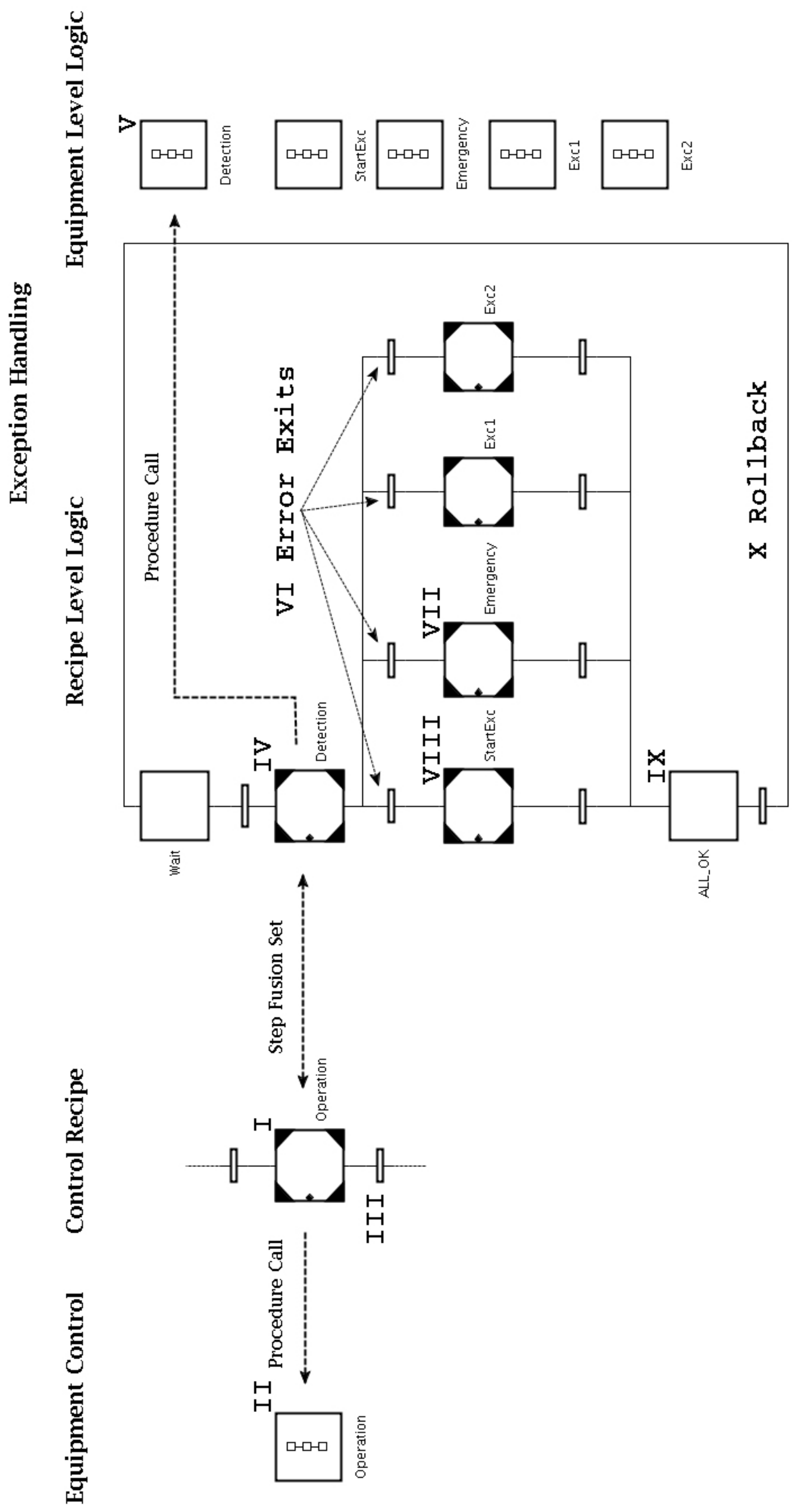


Figure 4.19 Step fusion sets for exception handling: recipe view and exception view.

fault-tolerant real-time systems [23]. One situation where it would be natural to be able to rollback the execution is when an operation is called and the equipment object is not in a allowed state for the operation to start. When the state of the equipment object is changed into one of the allowed states, the operation would be called again and the execution of the recipe would be able to continue.

If Fig. 4.19 is used as an example, the steps of the exception handling would be:

- The detection logic in `Detection (V)` is triggered by that the state of the unit is inconsistent with the start of the operation.
- The condition of the error exit for the start exception would become true and the `StartExc (VIII)` operation is called from the recipe level exception handling.
- The `StartExc (VIII)` operation takes the unit to an allowed state for the start of the original operation, called from the control recipe, and the step `ALL_OK (IX)` will become active.
- The exception recipe would make a `Rollback (X)` and restart the detection operation and the operation in the control recipe, since the two steps are in the same step fusion set.

However, due to the nature of chemical batch processes a rollback is in most cases not a viable alternative. For example, it is very seldom possible to undo a chemical reaction. Also in the more common case where the batch cannot be produced as intended there are several alternatives. In certain situations it might be possible to still make use of the batch to produce a product of a different grade or quality. In other situations it is possible to recirculate the batch ingredients for later reuse. Also in the case where the batch cannot be used as a product, special actions must be taken. Due to environmental regulations the partly produced batch must be taken care of in an appropriate way. This may include further processing to separate or destroy the batch ingredients.

One problem, which occurs when an operation is restarted after an exception is taken care of, and the normal execution should continue, is that in the implementation described above the operation will start from the beginning. It will try to reserve the unit again, *etc*. Since the unit is already reserved by the operation itself the check need to be overridden and some manual control by the operator is necessary. One way to take care of this problem is to change the linking between the recipe and the equipment logic to a lower level. The recipe operation is divided into several recipe phases or even smaller parts (e.g. the reservation of the unit), see Fig. 4.20. Each of the recipe phases makes a procedure call to the corresponding phase at the equipment control level. In this approach the number of exception views is increased. One view per phase is needed, the views will only contain the exception handling needed for the phase. It is not necessary to have the logic that deals with the reservation of a unit once it is reserved, *etc*. The detection procedure becomes more specific and the number of error exits are decreased in each exception view.

If the operations are divided into smaller parts some of the phases can be reused by other operations. For example, the procedures for reserving and releasing a unit are unit specific and could be reused by all the operations belonging to the same unit.

In the implementation described above the separation between the recipe level and the equipment control exception is made on the operation level using procedure steps and procedures in JGrafchart. The separation of the exception handling is on the same level as the separation in the procedural control. The separation could be at any level specified in S88, see Fig. 2.3.

Another way to implement the exception handling is to have the exception detection logic in the equipment operations, see Fig. 4.21. If an exception occurs it will be detected by the exception conditions in the operation. The exception conditions are based on the equipment state machines of the unit and the equipment/control modules. The normal execution of the operations is aborted and the operation will finish. The transition in the control recipe will not be fireable. Instead the error exit corresponding to the exception will be fired and the exception handling will try to recover from the exception, see Fig. 4.22.

Recipe Dependent Conditions

Units are usually used for the manufacturing of several different products and different grades of the products. The recipe parameters, e.g. size, temperature of reaction, duration of mixing, and catalyst, are therefore changed according to the specifications of the product. How-

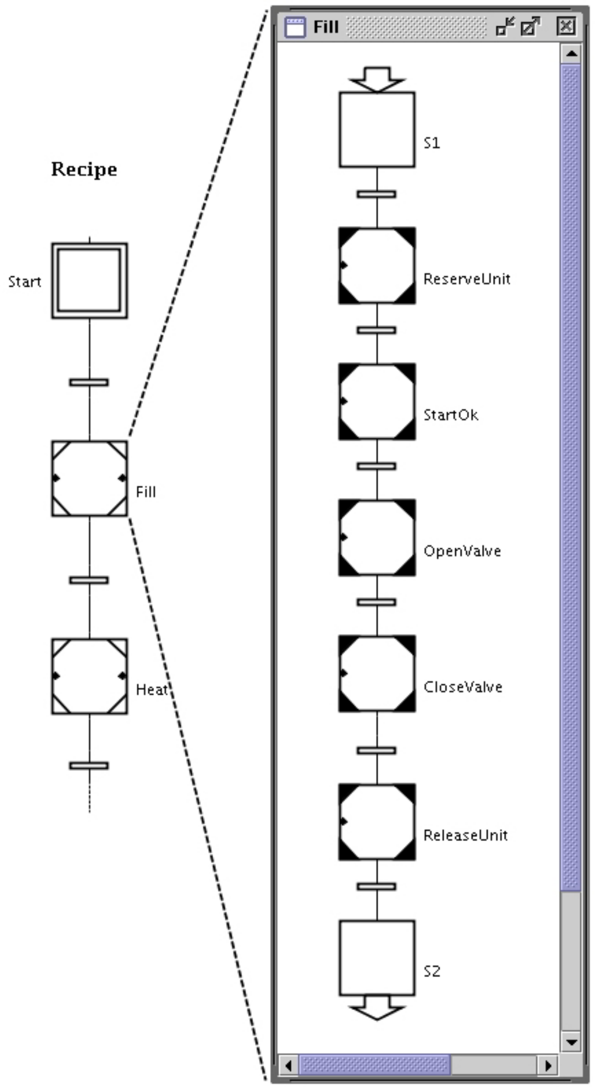


Figure 4.20 A recipe operation divided into phases and smaller parts. Each of the procedure steps calls a procedure at the equipment control level.

ever, it is equally important to change the parameters of the exception handling, i.e. the conditions in the equipment state machines of the different equipment/control modules. For example, what was considered a normal temperature when producing one product might lead to a run-away reaction when producing another product. The conditions of the transitions can easily be changed by using stored actions in the procedure step, which calls an operation, to change the variables used in the conditions of the transitions in the state machine. In the same way exit actions can set the variables to, e.g., the default values of

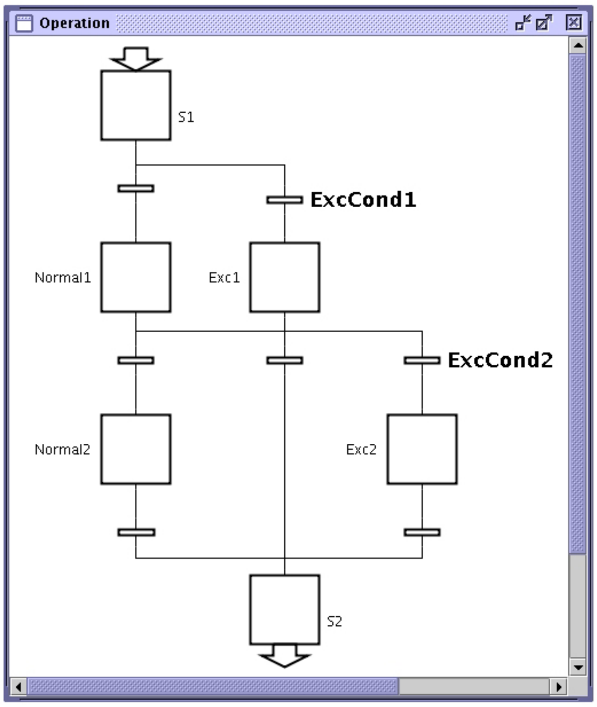


Figure 4.21 Exception detection logic in an equipment operation. If an exception occurs the execution of the operation will finish but by following the normal path. The condition of transition after the procedure step in the control recipe will not be true. Instead the error exit in Fig. 4.22 corresponding to the error, which has occurred is fired.

the unit, when a procedure is finished. When the procedure step in Fig. 4.23 is activated the variable `QualMAx` in the state machine will get the value `90.0`. This means that if the temperature is not changed the state will go to `Normal`, since the value of the temperature sensor is less than the value of `QualMAx`.

4.4 Synchronization

One important, and often hard to implement part, in batch control systems is the synchronization between units. In the recipe, two operations seem to be simultaneous, but on the equipment control level synchronization is needed. For example to make sure that valves and pumps are opened/closed and started/stopped in the correct order during a transfer from one unit to another, the units need to perform some

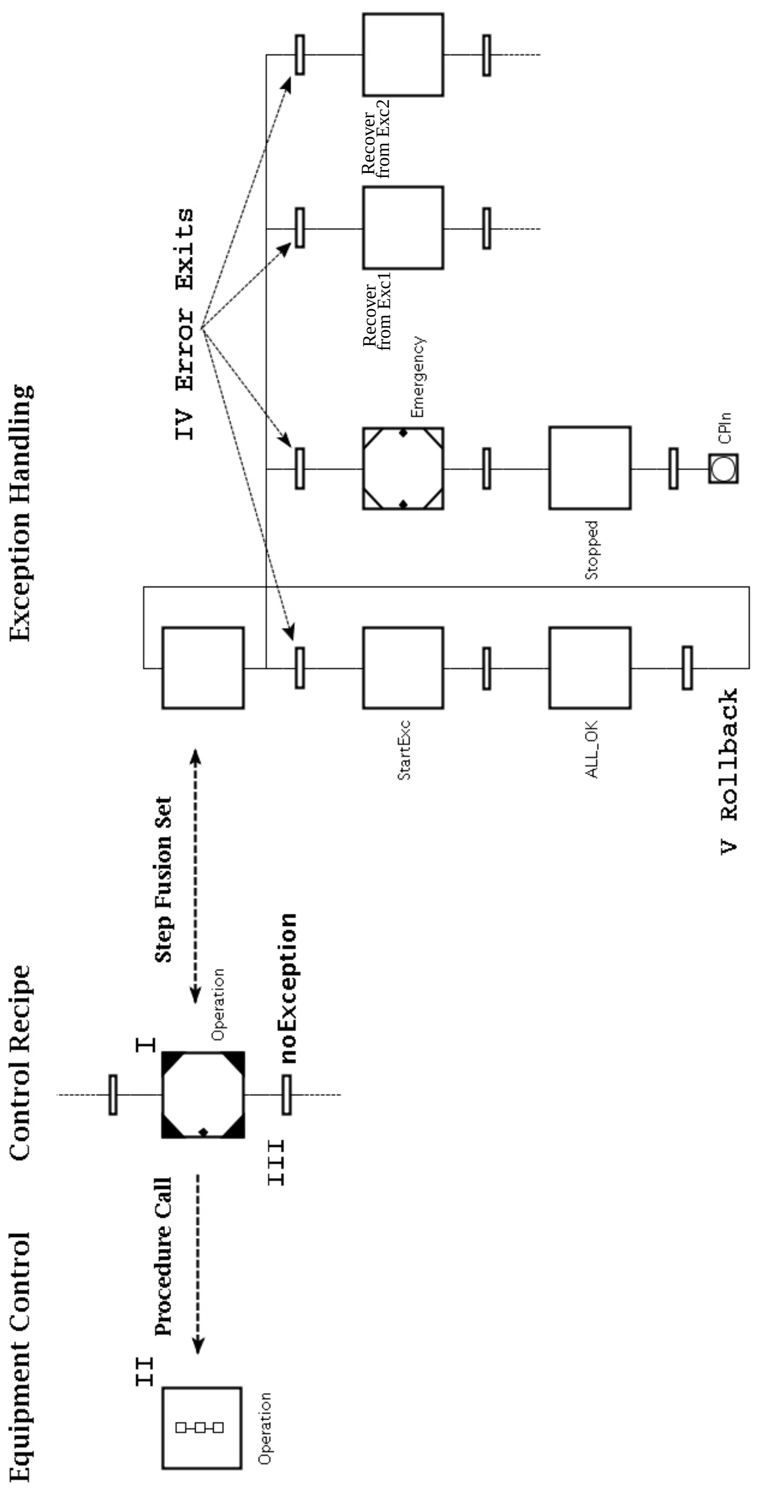


Figure 4.22 A procedure step (I) calls the operation (II) with the detection logic, see Fig. 4.21. The transition (III) after the procedure step will only fire if no exception is detected. If an exception is detected the corresponding error exit (IV) will fire and the exception handling will try to recover from the exception.

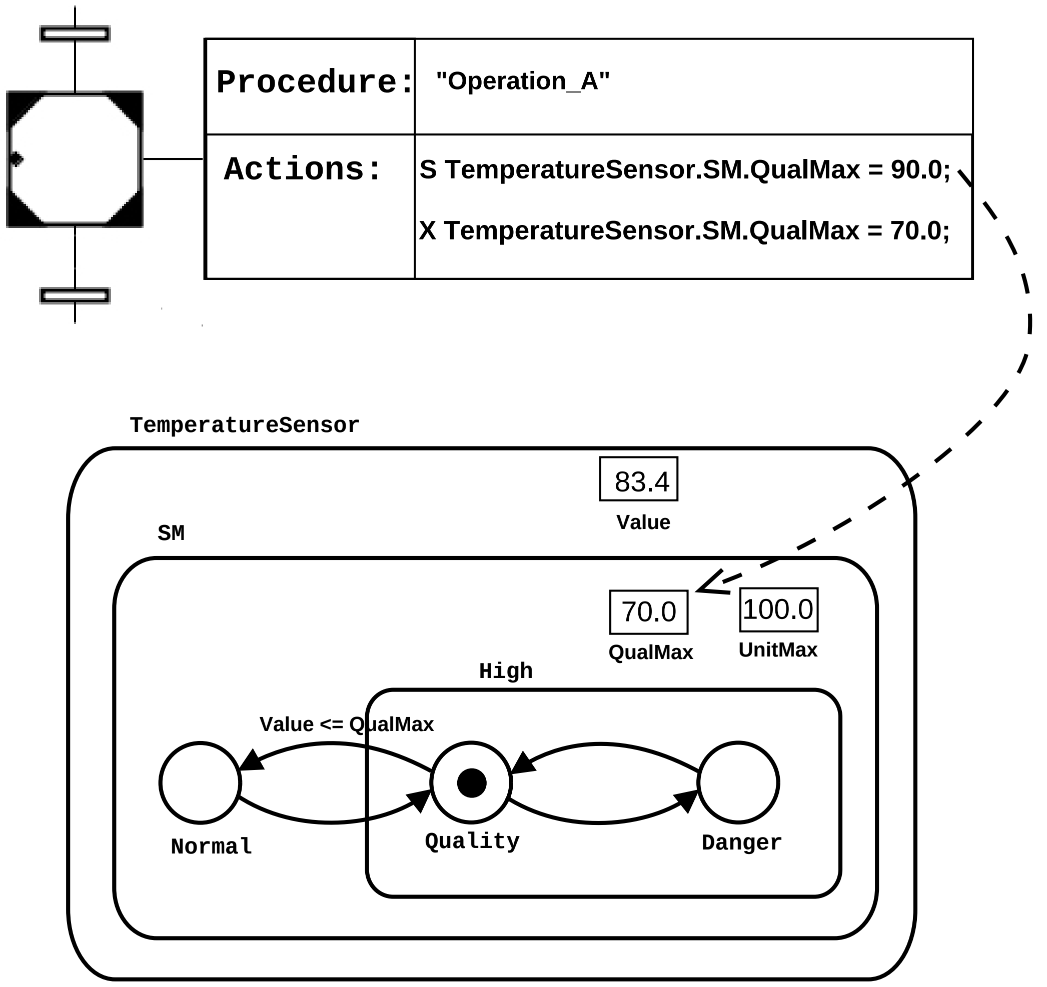


Figure 4.23 Conditions in state machine set by a procedure step.

sort of handshake. If the pump is a displacement pump and the pump is started before the valve is opened, there is a risk that the pipe will burst. The handshake works as an interlock to make sure the actions are taken in the right order. In Fig 4.24 an implementation of a handshake between a transfer operation and a receive operation in two different units is shown.

The task becomes even harder when also the recipe level exception handling should be synchronized with the normal parallel recipe operations, i.e. concurrent operations in the recipe.

In Fig 4.25 the transfer operation of `Unit1` and the receive opera-

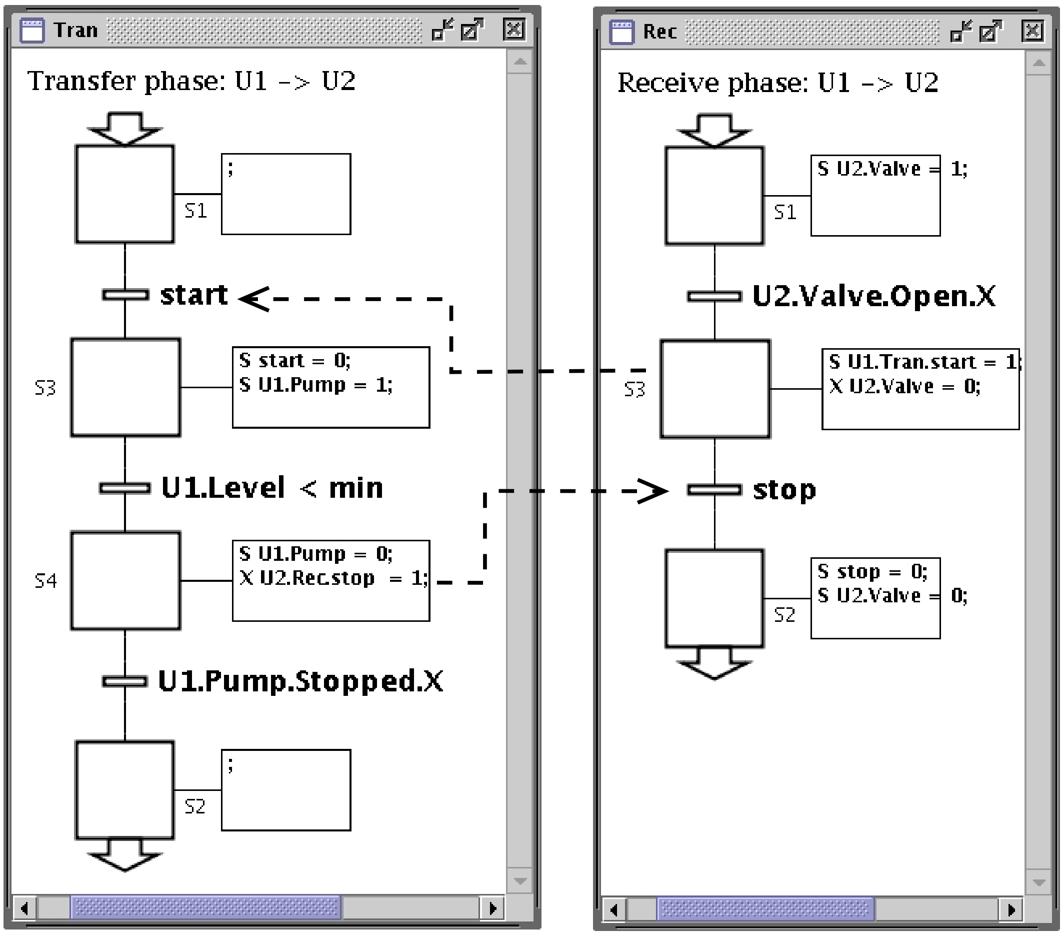


Figure 4.24 Synchronization using handshake between operations in two units.

tion of `Unit2` are shown at the recipe level as well as the procedure step representing the detection operations in the recipe level exception handling in both units. The four procedure steps in the figure will be activated at the same time instance.

If an exception occurs in `Unit1` during the execution, the exception handling for `Unit1` will be activated. If `Unit2` does not receive this information it will continue to try to transfer material. `Unit2` needs to know which kind of fault has occurred in `Unit1`. The two units need to perform a controlled abortion of the two operations in a handshake manner just like when starting and stopping the normal execution of the two operations.

The result is that the exception logic for the errors in the receive op-

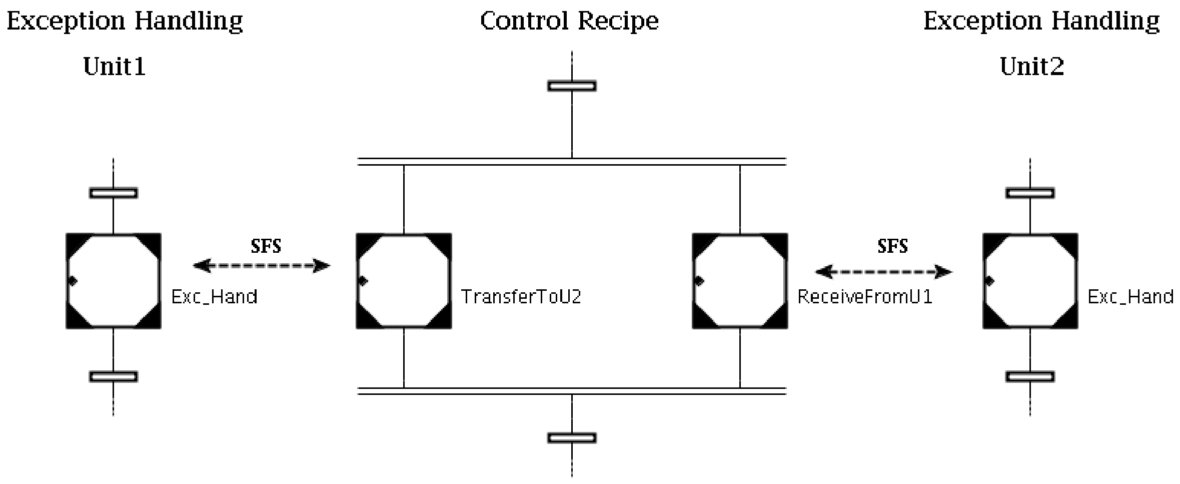


Figure 4.25 Transfer and receive operations in a recipe with exception handling.

eration must mirror the exception handling logic for the corresponding transfer operation (and the other way around) to perform a controlled abortion of the two operations. Exception handling logic for exceptions occurring in `Unit1` must be part of the exception handling logic of `Unit2` and vice versa.

5

A Case Study

5.1 Introduction

The Department of Chemical Engineering at Universitat Politècnica de Catalunya (UPC) in Barcelona, Spain, has developed a small batch pilot plant called Procel. The plant consists of three reactors of glass with agitators, heaters, level sensors and temperature sensors. The reactors are connected in a highly flexible way using pumps, pipes, and magnetic valves, see Fig. 5.1. The plant also has a system of heat exchangers. Currently these are only used when the plant is running in continuous mode. However, they could be used during the transfer of a batch from one reactor to another or during circulation within a unit. The physical plant is located in Barcelona, but there also exist a realistic simulation model of the plant implemented in MATLAB/Simulink. The simulation model has been used to develop a batch control system for the real plant. The system includes recipe management, a graphical user interface, and exception handling. The batch control system is implemented in JGrafchart. The communication between the Simulink model and JGrafchart is accomplished using Java TCP sockets. The socket server is running in Matlab and JGrafchart acts as a client.

Procel has been used as a test pilot plant in the research of monitoring, control, on-line fault diagnosis and reactive scheduling of batch processes, see for example [33, 7]. In this work the pilot plant has been used as a realistic example to test the exception handling scheme de-

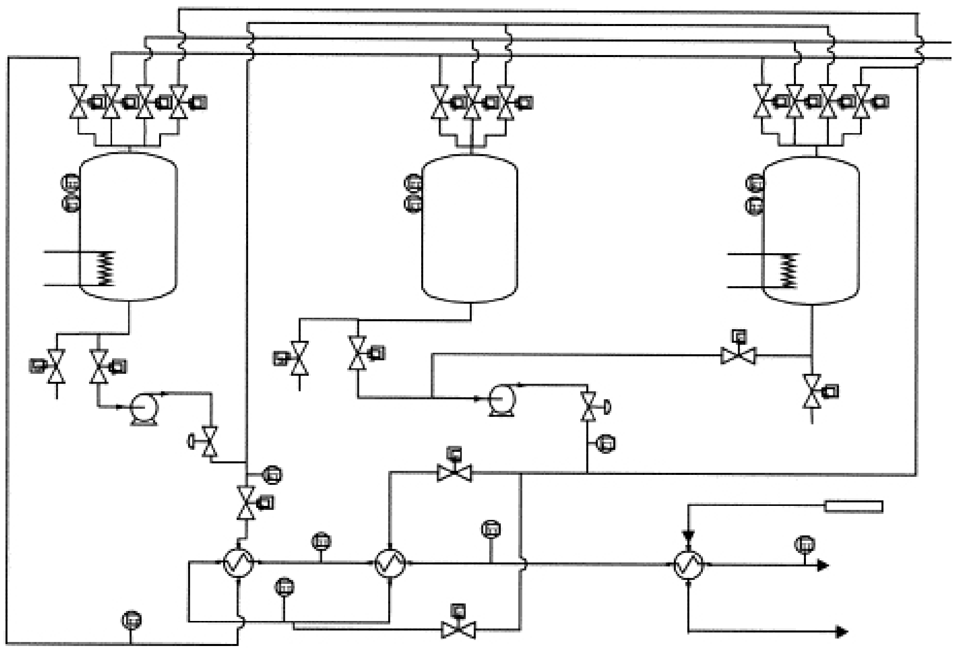


Figure 5.1 A schematic overview of the Procel plant (from [33]).

scribed in Chapter 4, and for testing how to close the information loop within a plant.

5.2 Control and Exception Handling

The original control system for the Procel plant is ABB's Sattline [25], which is still running underneath JGrafchart in the new control system. In this way safety is achieved and the use of JGrafchart becomes more flexible. Sattline is used as a safety net and has several safety and equipment interlocks, which would have been needed to re-implement otherwise. JGrafchart sends commands over TCP/IP to an OPC server connected to Sattline, which then changes the states of the physical equipment. Socket inputs and socket outputs are used in JGrafchart. The communication is shown in Fig 5.2. In the rest of this chapter Sattline will be considered as a part of the Procel plant instead of a control system.

A batch control system for the Procel pilot plant following the S88 standard has been implemented in JGrafchart including a graphical

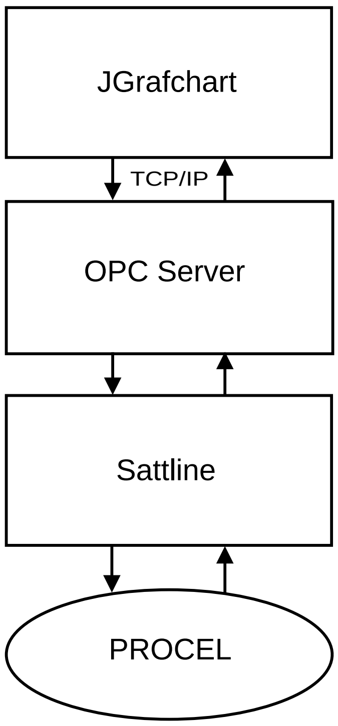


Figure 5.2 Communication from JGrafchart to Procel

user interface of the plant for operators, see Fig. 5.3. Currently the interface only displays the current values of the sensors and the state of the different equipment objects, but this could be extended with dialogs to make it more interactive. For example, the graphical interface could link to the workspaces of the equipment units.

The control system includes recipe management and exception handling. The plant is divided into four units according to the physical model in S88. See Fig. 5.3 for the names of the equipment. The first three, `U1-U3`, represent the three reactors and the fourth, `U4`, consists of auxiliary equipment not part of any of the other units and should maybe not be considered as a unit at all, but as shared resources. The valves and sensors, i.e. flow meters and temperature sensors, used in the heat exchanger network are part of `U4`. The pump `B2` is also in `U4` since it is used in different ways and is not directly part of any of the other units. Since `B2` is used to pump from both `U2` and `U3` it contains its own operation for pumping.

Each of the units representing the reactors consists of equipment

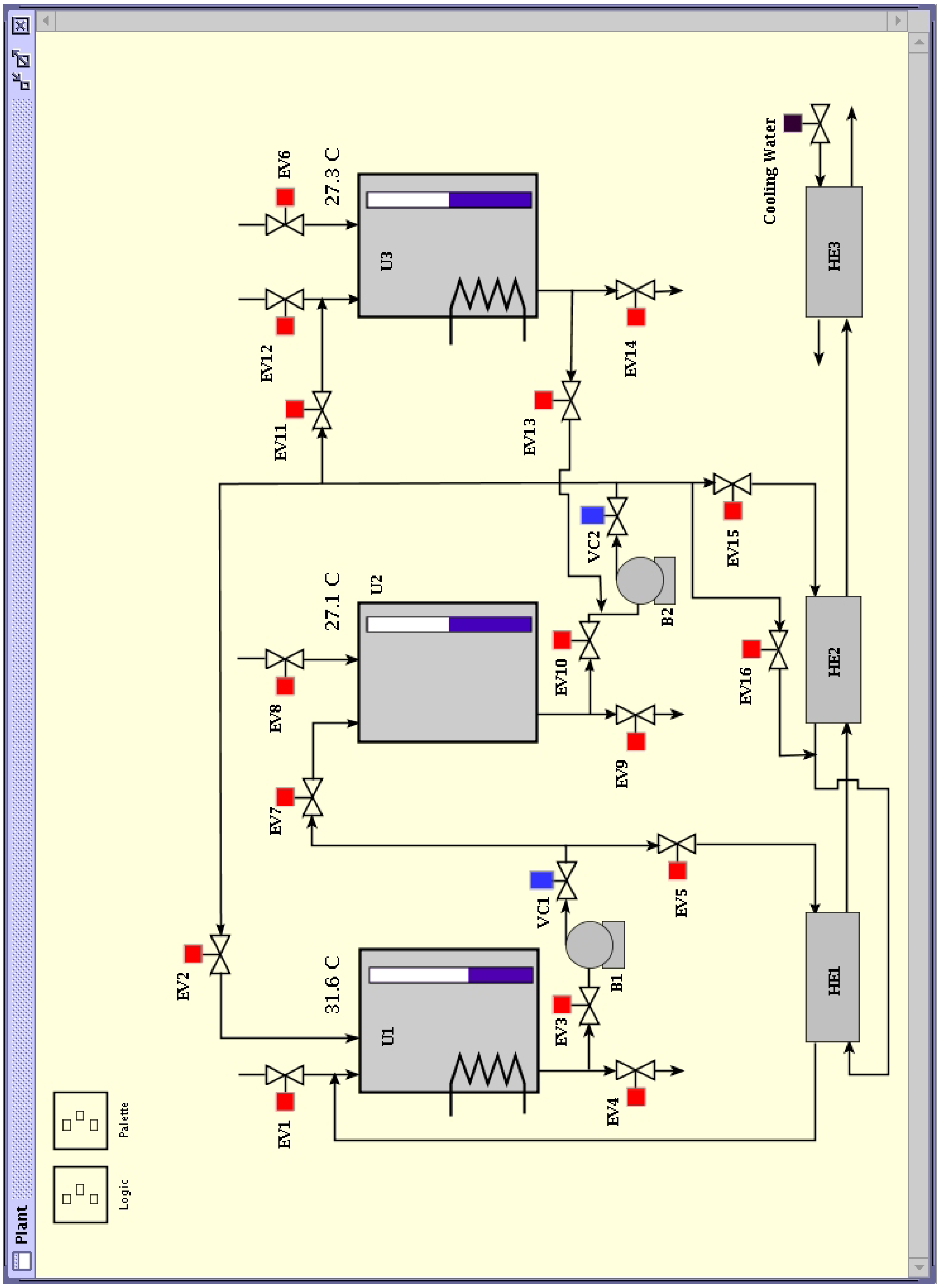


Figure 5.3 Operator interface for the Procel plant in JGrafchart

and control modules such as agitators, valves, level, and temperature sensors. The modules are stored on sub-workspaces of the units' workspaces. The units also contain the equipment control. In the implementation the recipe/equipment control separation is on the operation level in S88. This means that the recipe makes a procedure call from a procedure step representing a recipe operation to a procedure representing the corresponding equipment operation belonging to a unit. The operations are stored on sub-workspaces on the units' sub-workspace. In Fig. 5.4 the sub-workspace of `U1` is shown with its operations, e.g charge, heat, clean, transfer, and discharge and its equipment/control modules representing the valves, heater, agitator, and sensors. Also the unit's state machine, which is modeling the state of the unit is located on the unit's sub-workspace. The state machine has the states `Available`, `Reserved`, and `Error`. The unit's state machine will reach the `Error` state if any of the equipment/control modules reach their `Error` state.

The workspace of the `Charge` operation of `U1` is shown in Fig. 5.5. On the workspace are the state machine `SM`, the start state machine named `Start`, the exception handling `EH`, and the procedure `Proc`. The state machine models the charge operation according to Fig. 4.14. `Start` holds the logic for allowing the operation to start, according to Fig. 4.13. The exception handling workspace holds both the recipe level exception handling logic and the equipment level exception handling logic, according to Fig. 4.15.

The workspace of `EV1`, a valve, is shown in Fig. 5.6. It contains the state machine of the valve, and two boolean variables, `Open` and `Error`. The `Open` variable is the current control signal to the valve. The `Error` attribute becomes true if the error detection logic detects any error with the valve, e.g. it fails to respond to an open command within a certain time. On the `Logic` workspace object is a function chart, which reads from socket inputs and writes to socket outputs, located.

The control system for Procel in JGrafchart consists of 4 units, 30 equipment/control modules, 25 operations/phases, and 30 exception handling workspaces.

Recipes

Several different recipes can be used in the Procel plant. One of the recipes contains the following sequential operations:

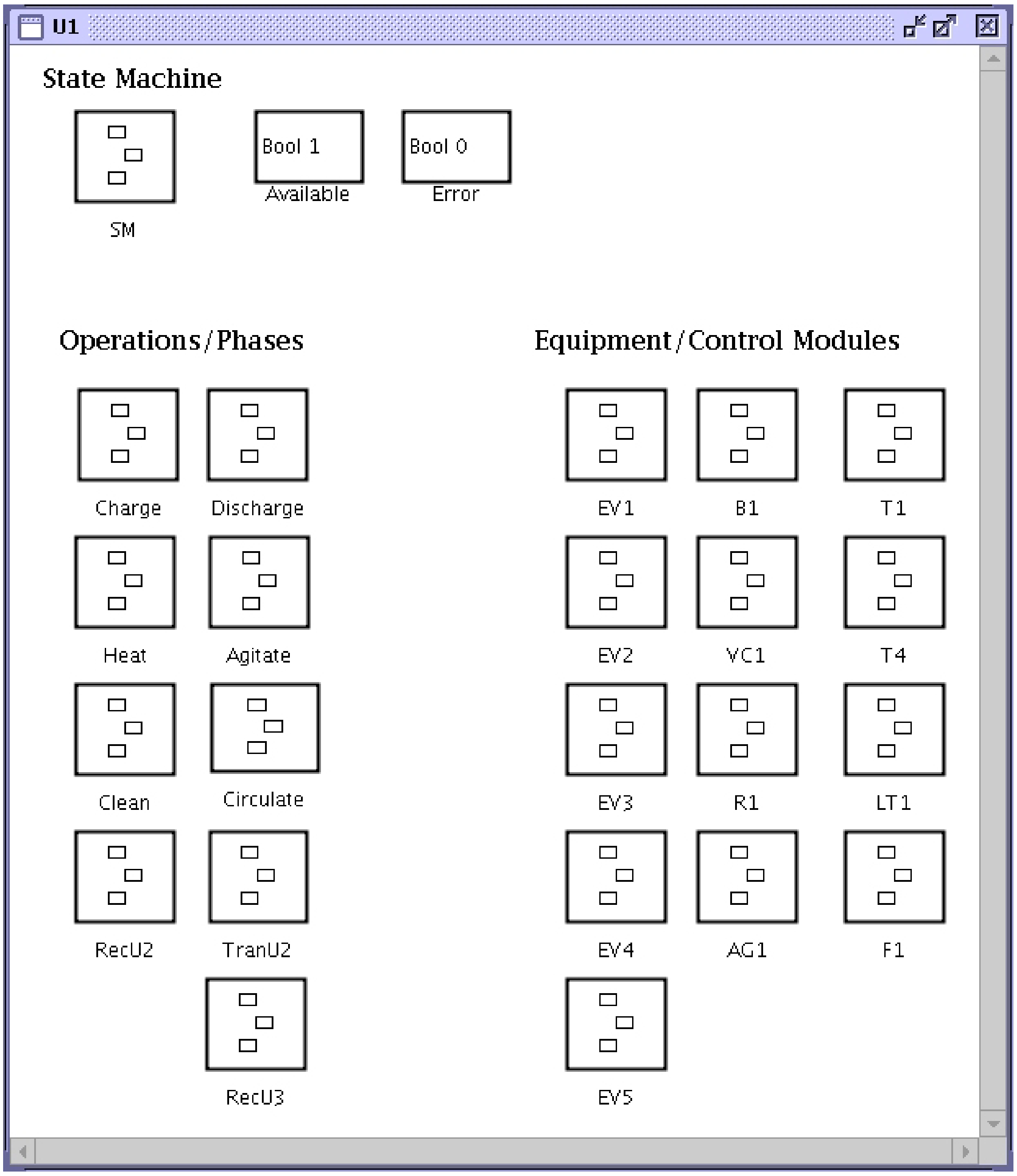


Figure 5.4 Control system configuration for U1.

- Charge `U2`.
- Hold the content in `U2` for X time units.
- Transfer the content in `U2` to `U1` using pump `B2` in `U4`.

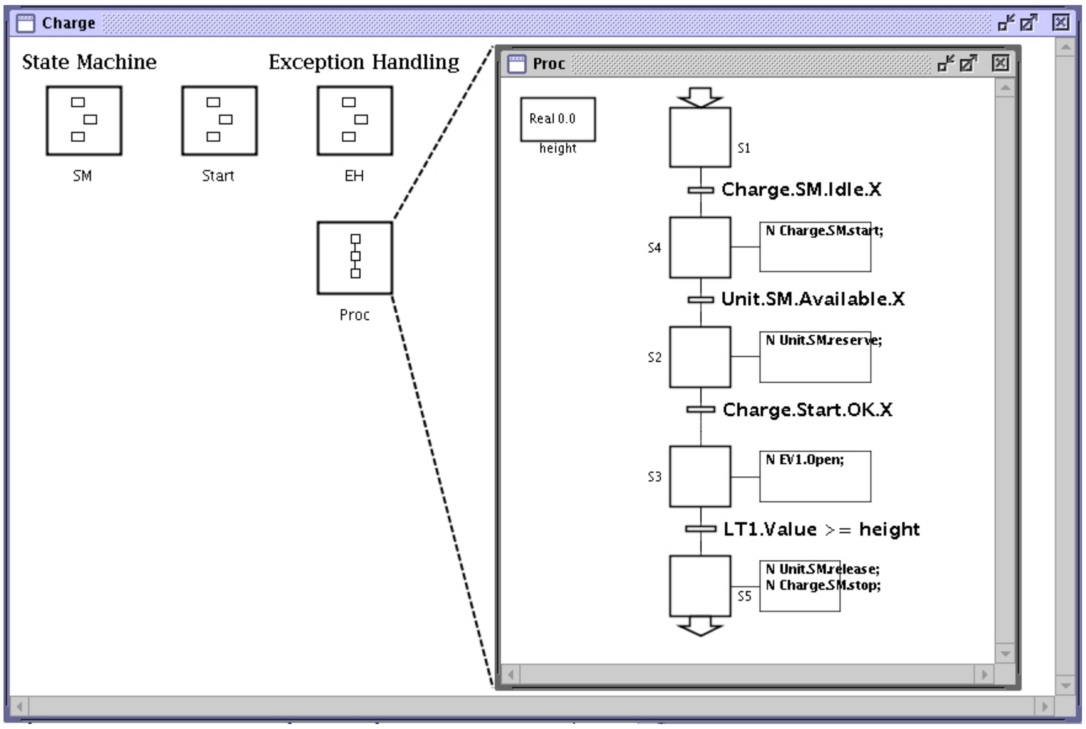


Figure 5.5 Workspace of the `Charge` equipment operation of `U1`.

- Heat the content of `U1`.
- Empty out the content of `U1`.
- Clean `U1`.

Each recipe is implemented as a procedure in JGrafchart. The implementation of the recipe in JGrafchart is shown in Fig. 5.7.

5.3 Exception Handling Example

The recipe in Fig. 5.7 is used as an example of the exception handling. The exception is chosen to be that the in-valve `EV8` for filling `U2` is not responding to an `Open` command from the `Charge` operation. The steps of the recipe execution and the exception handling are:

- The recipe is started from the scheduler.
- The `Charge` operation checks if `U2` is available.

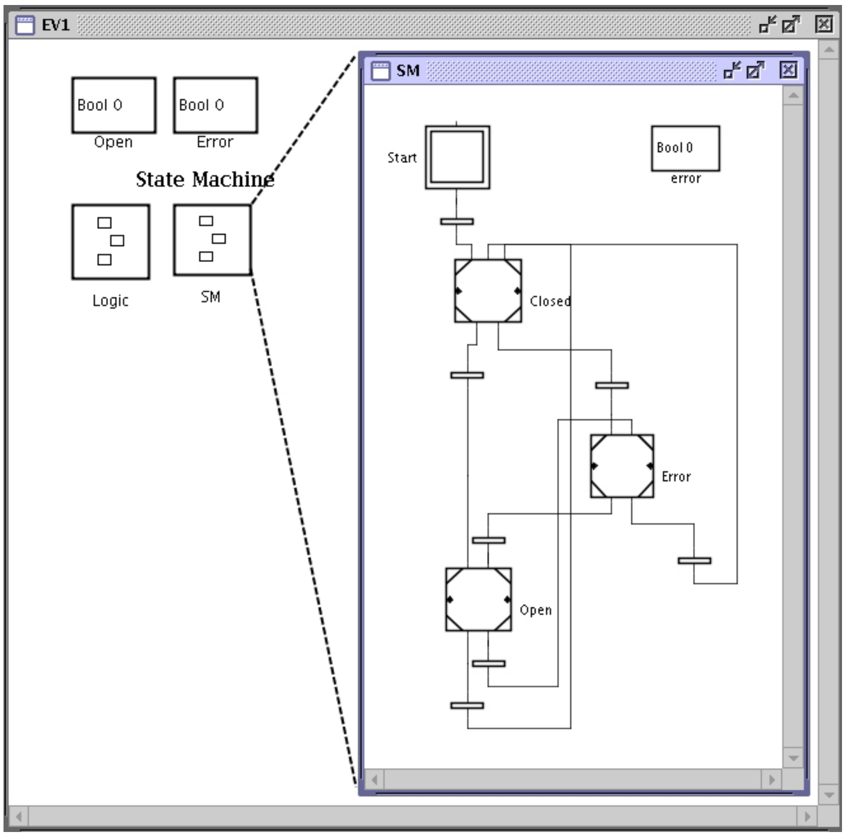


Figure 5.6 Workspace of the control module EV1, a valve.

- U2 is reserved by the Charge operation.
- The state of U2 is checked to be consistent with the start of Charge.
- The procedural state of the Charge operation goes to the Running state.
- The Charge operation sends an Open command to the EV8 equipment module.
- The control signal of EV8 is written to the IO.
- The state machine of EV8 detects that the physical valve is not responding to the control signal and it goes to the Error state.
- The Error state propagates to the state machine of unit .

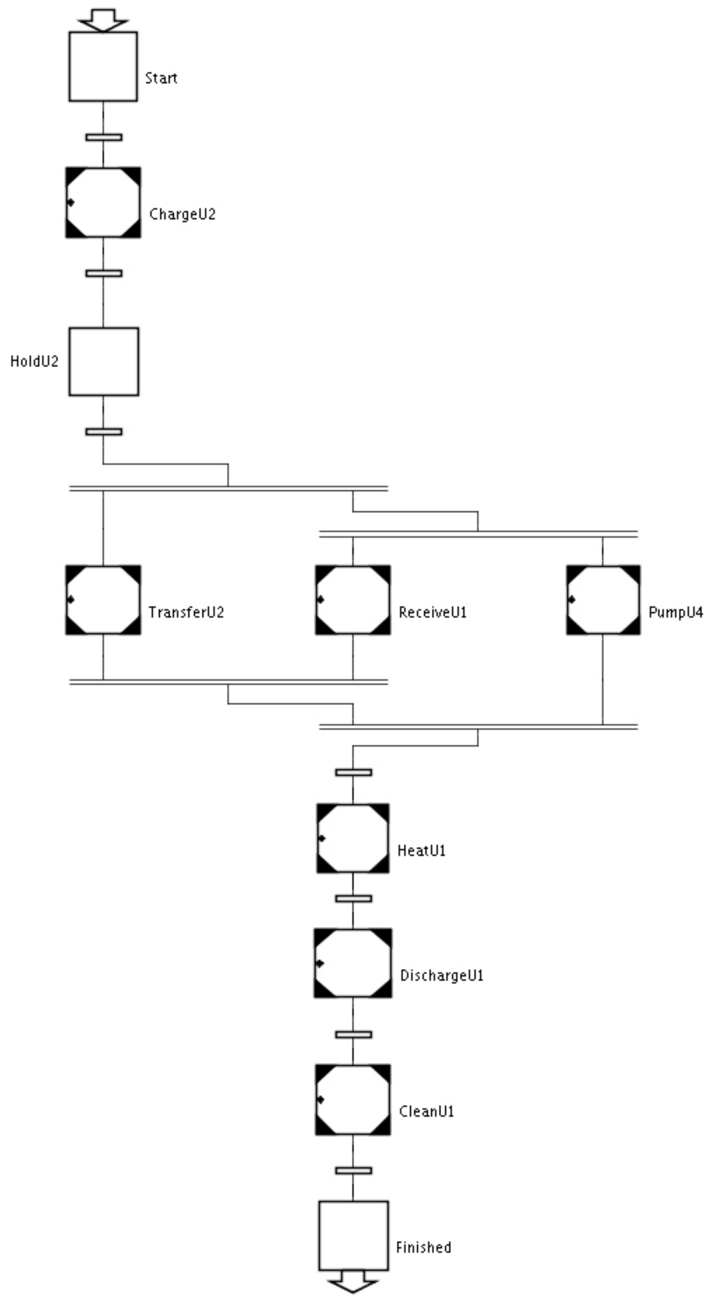


Figure 5.7 A recipe for the Procel plant.

- The recipe level exception handling logic detects that the state machine of `U2` is in the `Error` state.
- The `Error Exit`, in the recipe level exception handling, for the valve not responding is fired.

- The Charge operation is aborted and the procedural state of the Charge operation goes to the Aborted state.
- The exception handling operation for the valve not responding tells the operator that the valve is not responding.
- A technician is probably able to fix the valve.
- The state machines of all the equipment objects are reset from the Error state.
- Since no actions were taken that cannot be restarted, the recipe level exception handling makes a rollback and the exception detection and the Charge operation is restarted.
- The procedural state of the Charge operation goes to the Running state.
- When the correct amount of reactants are filled in U2 the Charge operation sends the Close command to EV8, which now closes.
- U2 is released.
- The procedural state of the Charge operation goes to the Idle state.
- The rest of the recipe is continued.

5.4 Integration

As described in Section 3.3 Grafchart can be used at the local control level and at the supervisory control level. At UPC we have extended Grafchart to include the management of batch schedules, see Fig. 5.8. JGrafchart receives a list of scheduled batches from MOPP, a scheduler package developed at the UPC [14, 33]. MOPP sends the schedule using xmlBlaster, see Section 3.2. JGrafchart unpacks the schedule and starts each batch as a control recipe that waits for its start-time to occur. The communication protocol used between MOPP and JGrafchart is the Batch Markup Language (BatchML) [41] specification. BatchML provides a set of XML schemas based upon the S88 family of standards. BatchML may be used to design interfaces in control, MES and ERP

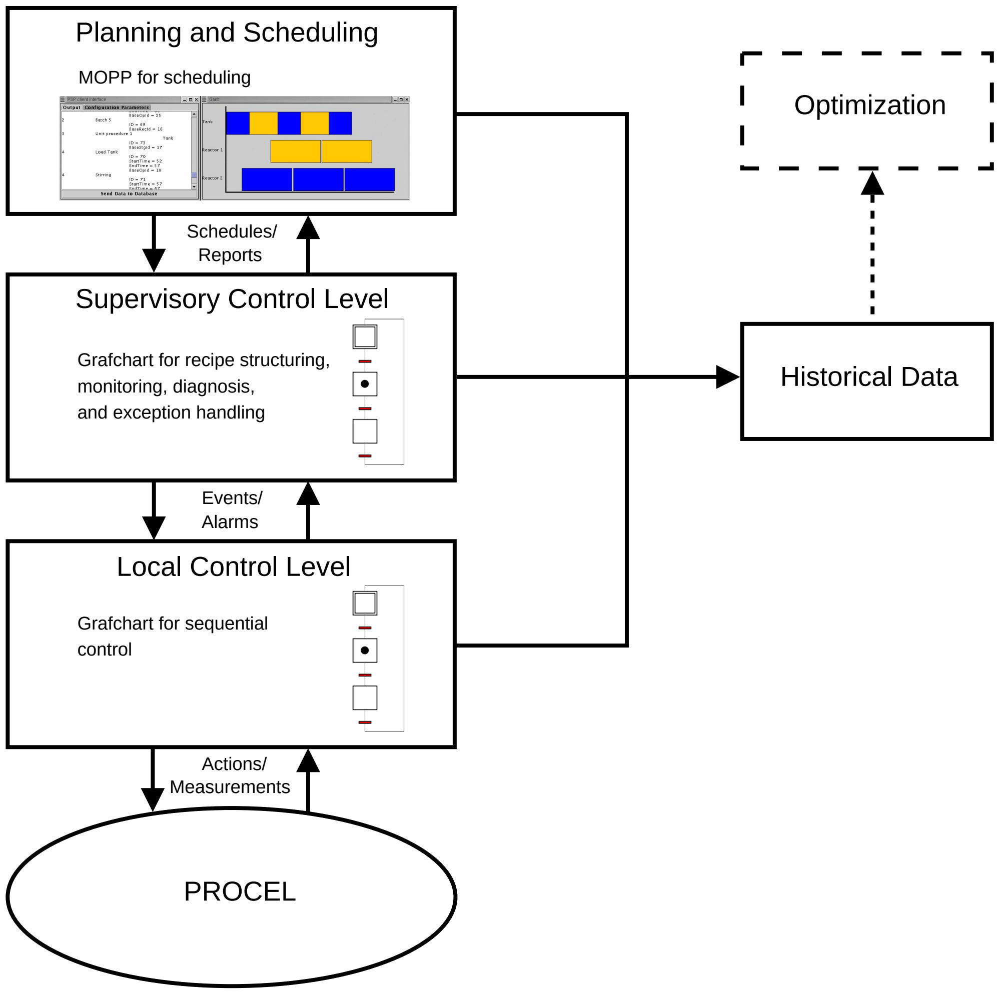


Figure 5.8 Integration of supervision, planning, and scheduling

systems as well as the basis for documenting requirements, designs and actual product and process data.

Since each batch is a control recipe carrying the whole information about the development of the batch a report can be sent back to the scheduler once the batch is finished. If an exception occurs and, e.g., the batch has to be canceled this information could be used to reschedule the batches. The historical data of every batch should be stored for documentation and the possibility to use the data for optimization of the process. The historical data base is important to be able to detect problems frequently occurring and to be able to concentrate resources to eliminate these.

6

A Modular Batch Laboratory Process

6.1 Introduction

A good control engineering course should include hands-on experiments. In the basic course in process control at the Department of Automatic Control, Lund Institute of Technology, a batch reactor process had been used in one of the laboratory exercises. The old process had not been working to satisfaction and a new process was needed. A number of laboratory control processes are available commercially, e.g., inverted pendulums, helicopter processes, level-controlled water tanks, *etc*. These are commonly used in laboratory exercises in different basic and advanced control courses. However, very few processes exist that illustrate the typical problems associated with batch control.

These led to the design of a new modular batch laboratory process. The process should be a tank of approximately half a liter in volume. The tank should be able to be heated, cooled, and agitated. The level and temperature of the tank should be measured. There should be an in-pump and an out-pump to be able to fill and empty the tank. The process should be inexpensive, portable, and with an easy to use I/O connection to the control system.

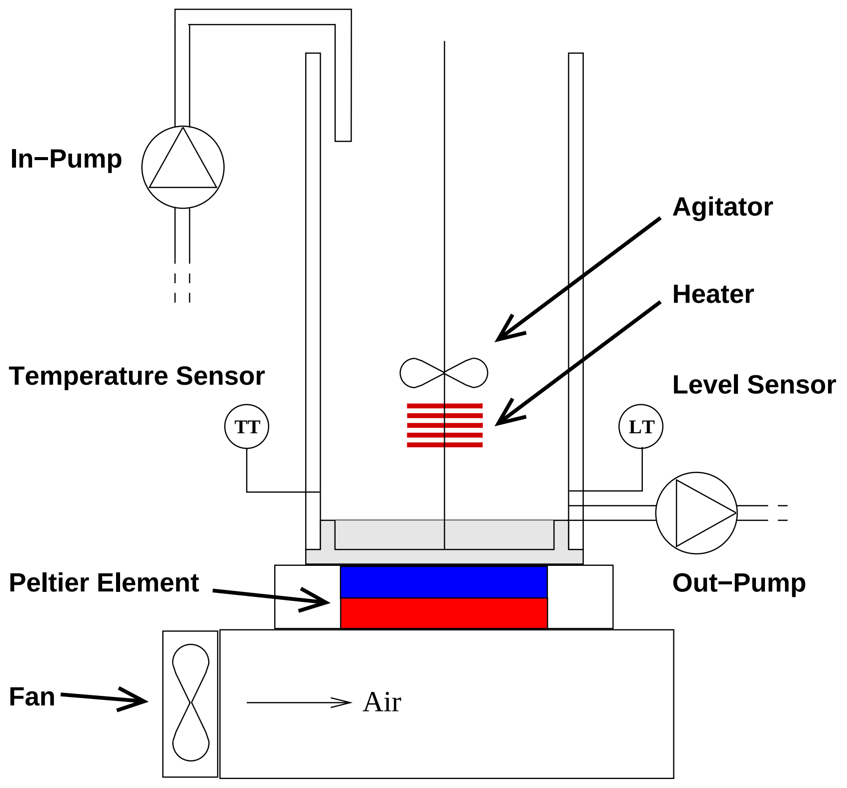


Figure 6.1 Outline of the laboratory process.

6.2 Process Description

The batch tank consists of a plexi-glass tube with a brass bottom. The volume of the tank is approximately 0.5 liter. Connected to the tank is an agitator, an in-pump, an out-pump, a level sensor, a temperature sensor, a heater, and a cooler, see Fig. 6.1. The cooler consists of a 40 W Peltier element cooled by a fan.

The outputs from the level and temperature sensors are in the range 0-10 V, which corresponds to Empty-Full tank and $0 - 100^oC$ respectively. The resolution can be up to 10-bit in microcontroller.

The pumps, heater, and cooler can either be controlled with digital, i.e. `On/Off`, signals or analog signals. The analog signal is transformed using an integrated microcontroller for pulse width modulation. The batch reactor has a built-in 8-bit microcontroller, ATmega8, from Atmel for this purpose. The microcontroller works with the GNU Compiler Collection [37]. The agitator only has a digital control signal, but to overcome the problem with friction the microcontroller may be programmed to give a higher control signal right when the agitator is turned on. The heater is a 24 V, 150 W heating element. There are three light-emitting diodes (LED) on the front of the process: one red for showing if the heater is on, one blue for showing if the cooler is on, and one red/green LED, which is red if there is some error and green if the power is on and the process is OK.

The microcontroller is also used for safety interlocks, e.g. it is not possible to turn the heater on if the tank is empty, the in-pump will stop once the tank is full, and there is a protection against over heating.

The microcontroller enables communication over RS-232. This makes it possible to control the process with a laptop, e.g. at presentations and lectures, without using an I/O card.

The water container of the tank consists of an off the shelf plastic tank with the size: 15 cm high, 30 cm wide, and 40 cm deep. The container also becomes a protection for the front part of the process during transportation. The real batch process can be seen Fig 6.2.

The process is made from standard components and therefore quite inexpensive. The cost of building one process, without the development cost, is approximately US$1200, of which half is material and half is work cost. The process is also small, so small that it fits easily on the desktop on the side of the standard PC used for the control system. The connection between the process and the PC is an ordinary RS-232 serial line. This means, that it is straightforward to control the process from an ordinary laptop PC at lectures or presentations. The final design of the batch laboratory process is only 40 cm high, 30 cm wide and 25 cm deep.

6.3 A Batch Process Laboratory Exercise

The process is used in a laboratory exercise in the course “Automatic Process Control”, a fourth-year course for chemical engineering students at the Lund Institute of Technology. The laboratory exercise introduces the students to discrete-event control of a batch reactor and it also requires the students to implement a discrete PID-controller. Both the sequential control and the PID-controller are implemented in JGrafchart. The sequential control consists of the following steps:

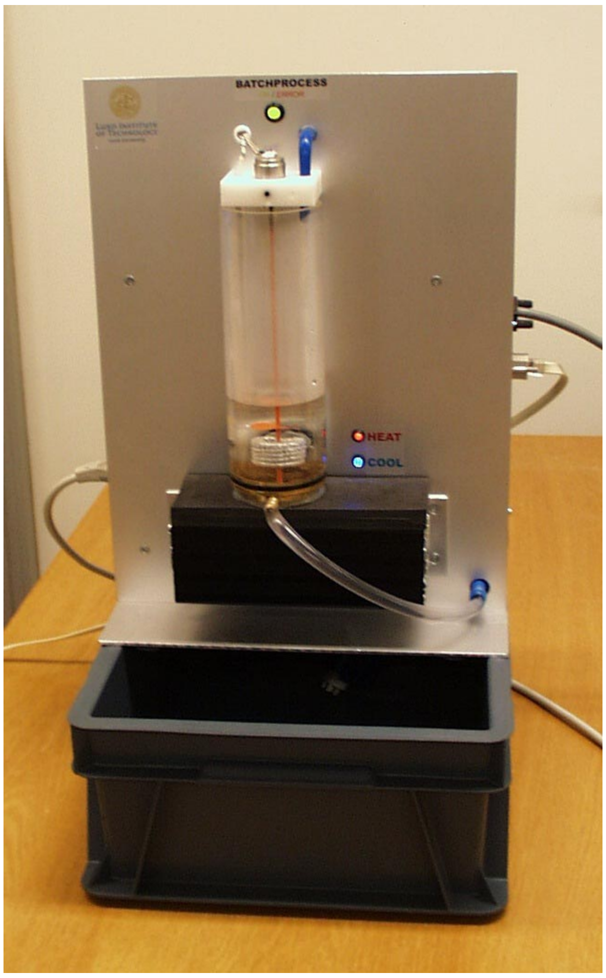


Figure 6.2 The batch laboratory process.

I) Fill the reactor.

II) Heat the reactor, during simulation of an endothermic reaction, to a specified temperature using PID-control.

III) Empty the reactor.

IV) Clean the reactor.

The sequence should then be restarted and a new batch made. A solution in JGrafchart can look like in Fig. 6.3.

In the exercise, the cooler is used to simulate an endothermic reaction in the reactor. This way the simulation of the rate of the reaction

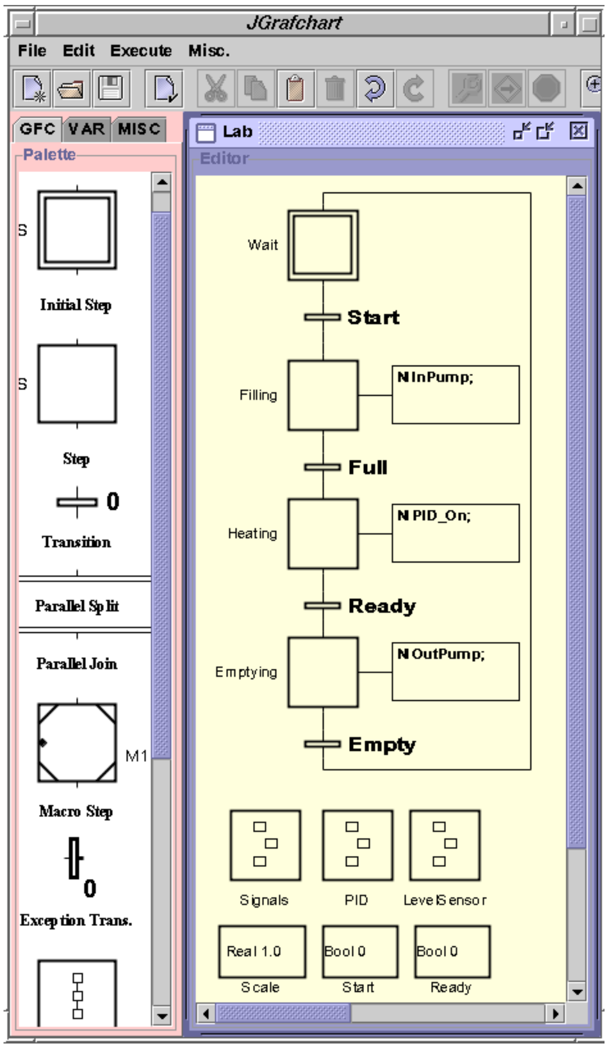


Figure 6.3 A solution to the laboratory exercise in JGrafchart.

can be made temperature dependent, non-linear, or constant.

Process Simulator

During the design of the hardware the time constant of the temperature dynamics was estimated to be a first order system with time constant $T \approx 1000$s (if the temperature step is not too large). The exercise would take a very long time if all the experiments should be done on the real process and therefore a simulation model has been developed. The simulation model can be used for development of controller schemes and parameter adjustments. The speed of the simulation can be changed.

To also be able to plot signals in real time for the real process it was decided to construct a server program that offers the following services:

a) A simulated batch process with the same interface as the real batch process.

b) Possibilities to plot temperature measurements, control and reference signals.

c) An animation of the batch process.

d) A socket interface so that clients can acquire all of the above services.

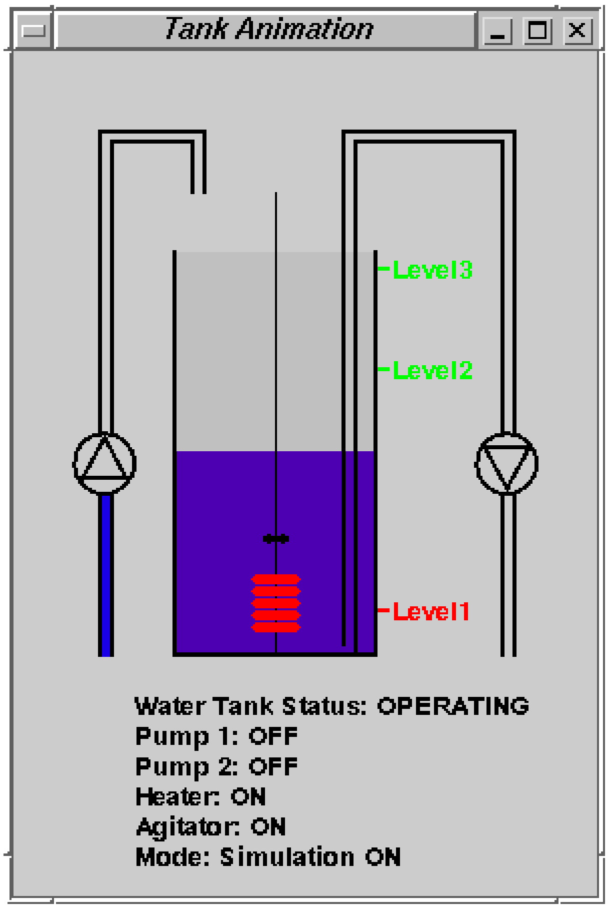
It was decided to implement the server in Java. Two communication threads were implemented. One that sends measurement signals from the virtual tank to clients connected to the server, and one that receives commands from clients.

The simulated batch process was implemented as a subclass of an existing process simulation class package. The class package provides methods to simulate systems that are described by differential or difference equations in real time. The coefficients for the differential equations were identified from the real process. The class also provides methods to animate the process with the public Graphics2D-class package. A screen-shot from the simulation is shown in Fig. 6.4.

A client, which connects to the server's socket can switch the simulation on and off. If the simulation is on the virtual process will react to the client's command signals and visualize the effect in the animation window. Moreover the signals will be plotted. If the simulation is turned off only the animation and plotting capabilities of the server are used to plot the signals from the real process.

6.4 A Laboratory Batch Plant

One of the main problems working with the Procel plant was the fact that the plant is located in Spain. The size of Procel (4 m wide, 2 m



Water Tank Status: OPERATING
Pump 1: OFF
Pump 2: OFF
Heater: ON
Agitator: ON
Mode: Simulation ON

Figure 6.4 Animation of the tank used both for the simulated and the real process.

high, and 1 m deep) prohibits the plant from being moved. Therefore, the development of the control system for Procel was based on the control of the simulated process.

The laboratory batch process is built in modules, which makes it possible to connect several processes together with extra pipes and valves to form a batch pilot plant. As a test, three processes have been connected together using plastic pipes and manual valves, see Fig. 6.5 and Fig. 6.6. This plant has the same functionality as the Procel plant and it is 55 cm high, 100 cm wide, and 40 cm deep.

The plant is controlled by one computer running JGrafchart in the laboratory at the department. Two of the processes are connected to a computer using RS232 and one is connected using an I/O card. The plant is highly flexible and connected in such a way that each of the processes can transfer its product to any of the other processes or empty it out to the tank beneath it.

The control system is implemented in JGrafchart according to the S88 batch control standard [22]. Each tank is modeled as an equipment unit in the control system.

To include the manual valves in the control system, dialog windows requesting the operator to manually open and close the valves have been added. Instead of opening the valve automatically it brings up the dialog and waits for the operator to confirm that the valves are in the correct position. A dialog window can be seen in Fig. 6.7. How JGrafchart can be used for operator support is described in [35]. If automatic magnetic valves are added to the plant they can either be considered to be a part of the existing units or they can form a separate valve-battery unit. If the valves are grouped as a new unit, the control system for the single tank process can be re-used without change. New operations need to be added for the transfer from one process to another, but to a large degree this could be part of the valve-battery unit. This way the plant could easily be extended to more units over time, only the control of the valve battery needs to be changed.

A major advantage of the modularity and size of the process is that it can easily be disconnected and stored or transported. Once disconnected each process can be fitted into a bag and easily carried by one person.

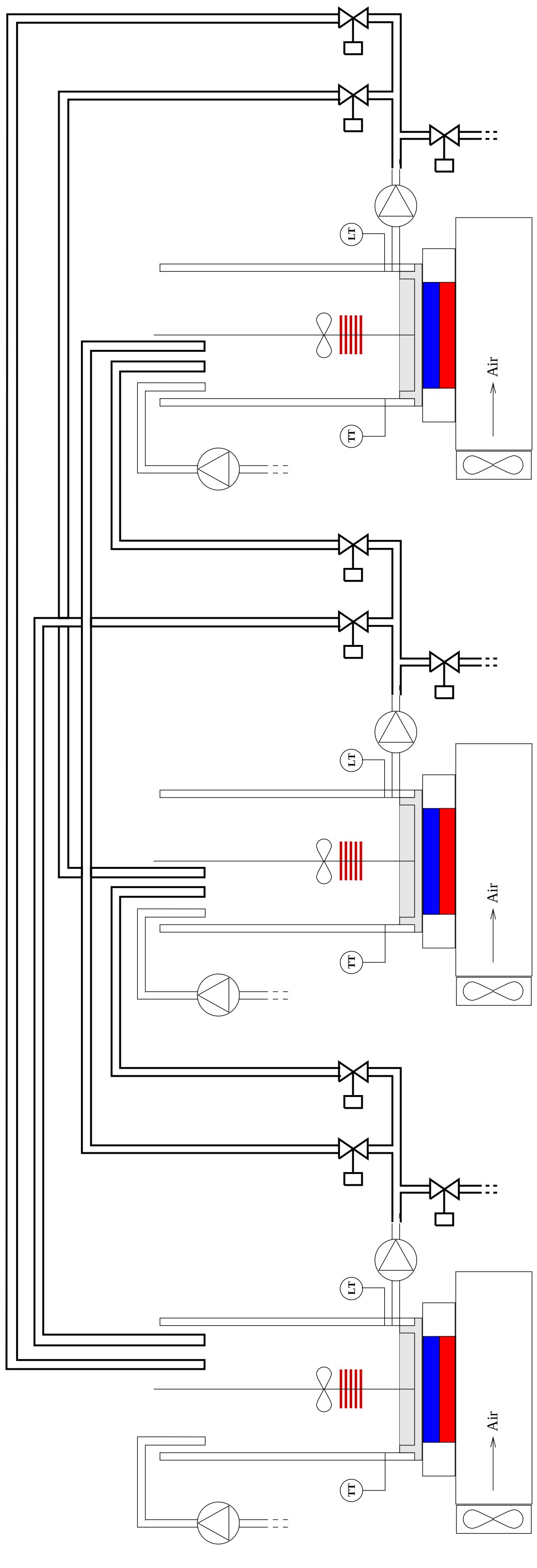


Figure 6.5 Outline of the batch pilot plant.

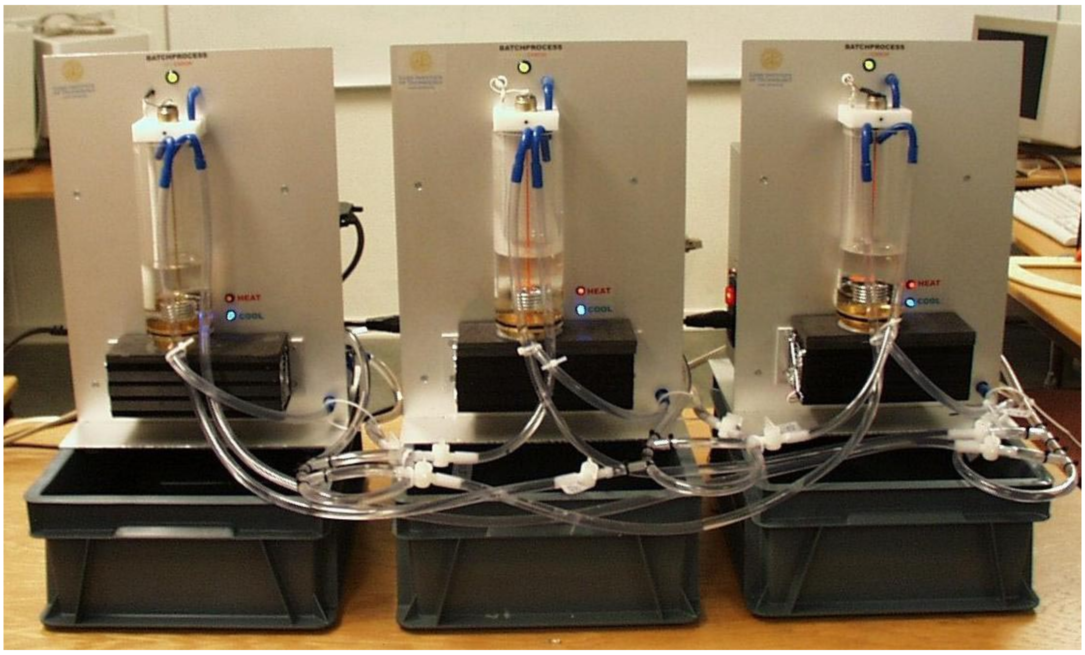
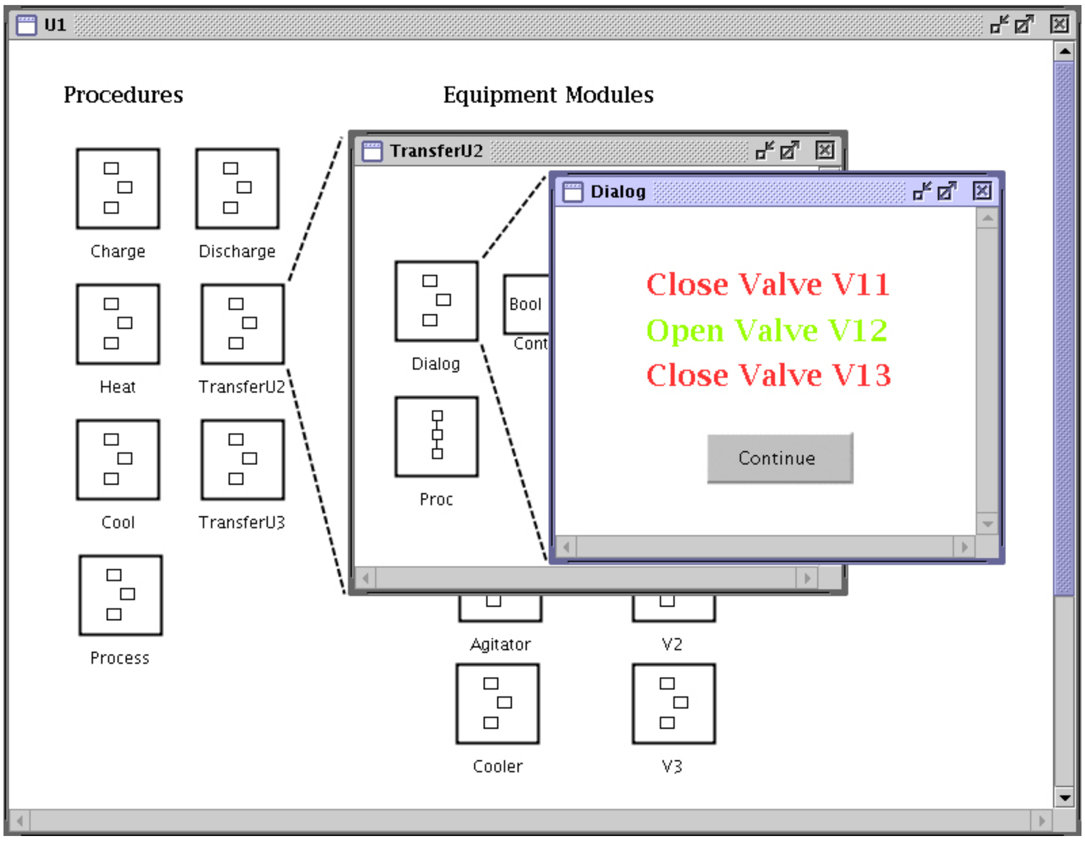


Figure 6.6 Batch processes connected together to a small batch pilot plant.



Close Valve V11
Open Valve V12
Close Valve V13

Figure 6.7 Dialog window for manual control of valves in the batch plant

7

Conclusions and Future Work

7.1 Conclusions

Exception handling is an important area of batch control that so far has received little interest from the standardization organizations. In this thesis a new approach to equipment supervision in recipe-based batch control systems is proposed. The approach is integrated with recipe execution, resource allocation, and scheduling. The approach uses hierarchical state machines represented in JGrafchart to model the equipment status and the status of the procedures executing in the equipment. A new approach for representing exception-handling at the recipe-level is proposed. The approach gives a clear separation between exception handling logic and the logic for the normal operation. Different possibilities for combining the two above approaches have been suggested. The proposed approach fits nicely into the S88 batch control standard.

The proposed approaches have been implemented in JGrafchart and tested on a realistic pilot plant, Procel, at UPC in Barcelona. The approach has proven to be able to structure exception handling in an easy to use way.

A new inexpensive and modular batch laboratory process has been developed. The process is used in the basic process control course for

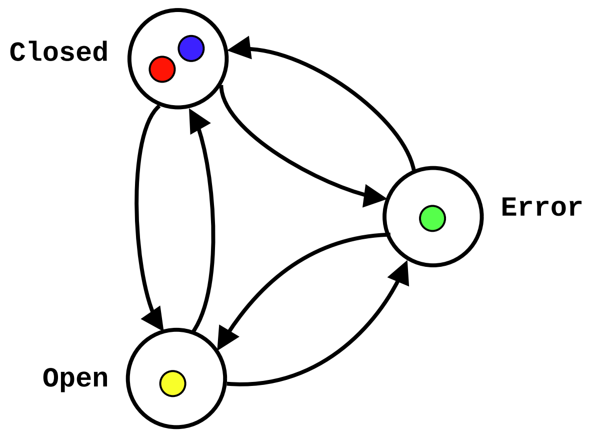


Figure 7.1 A high-level state machine for valves.

chemical-engineering students at Lund Institute of Technology to teach sequential control and discrete PID control. By connecting several processes a flexible multi-purpose batch cell can be constructed. The plant is small enough to fit on a desk in an officeğ.

7.2 Future Work

The state machine based structure for modeling equipment objects could be implemented in High-Level Grafchart. The state machine would model the behavior of a class, e.g. a certain type of valves, and each token in the state machine would be an instance of the class, e.g. a specific valve, see Fig. 7.1.

The use of the procedure state machine for operator support has not been fully investigated. In the same way as for the equipment state machines the procedure state machines could be implemented in High-Level Grafchart.

The strategy proposed on how to close the information loop between the scheduling and control of batches can be further investigated and implemented in JGrafchart. A lot of work can be made on how to implement the ideas for different scenarios and plants. The batch laboratory process developed at the department makes it possible to configure these scenarios and plants.

The methods described in the thesis rely on the fact that the faults are correctly detected. One topic for future work would be to see how the detection of faults can be integrated in the proposed structure for exception handling.

8

Bibliography

[1] American Institute of Chemical Engineers - AIChE. *Guidelines for Process Safety in Batch Reaction Systems*. AIChE, 1999.

[2] ANSI/ISA. "ISA S84.01 Application of Safety Instrumented Systems for the Process Industries." Instrument Society of America, 1996.

[3] K.-E. Årzén. "Sequential function charts for knowledge-based, real-time applications." In *Proc. Third IFAC Workshop on AI in Real-Time Control*, Rohnert Park, California, 1991.

[4] K.-E. Årzén. "Grafcet for intelligent real-time systems." In *Preprints IFAC 12th World Congress*, Sydney, Australia, 1993.

[5] K.-E. Årzén. "Grafcet for intelligent supervisory control applications." *Automatica*, **30:10**, pp. 1513–1526, 1994.

[6] H. Brettschmeider, H. Genrich, and H. Hanisch. "Verification and performance analysis of recipe based controllers by means of dynamic plant models." In *Second International Conference on Computer Integrated Manufacturing in the Process Industries*, Eindhoven, The Netherlands, June 1996.

[7] J. Cantón, D. Ruiz, C. Benqlilou, J. Nougués, and L. Puigjaner. "Integrated information system for monitoring, scheduling and control applied to batch chemical processes." In *Proceedings of the 7th IEEE International Conference on Emerging Technologies and Factory Automation*, 1999.

[8] D. Christie. “A methodology for batch control implementation - a real world lesson.” In *Proc. of World Batch Forum*, 1998.

[9] R. David and H. Alla. *Petri Nets and Grafcet: Tools for modelling discrete events systems*. Prentice-Hall International (UK) Ltd, 1992.

[10] D. Emerson. “What Does a Procedure Look Like? - The ISA S88.02 Recipe Representation Format.” WBF home page, http://www.wbf.org/, SP88 Part Two Overview - Paper 6, 1999.

[11] T. G. Fisher. *Batch Control Systems: Design, Application, and Implementation*. Instrument Society of America, Research Park, NC., 1990.

[12] M. Fritz, A. Liefeldt, and S. Engell. “Recipe-driven batch processes: event handling in hybrid system simulation.” In *Proc. of the 1999 IEEE International Symposium on Computer Aided Control System Design*, 1999.

[13] H. J. Genrich, H.-M. Hanisch, and K. Wöllhaf. “Verification of recipe-based control procedures by means of predicate/transition nets.” In *15th International Conference on Application and Theory of Petri nets, Zaragoza, Spain*, 1994.

[14] M. Graells, J. Cantón, B. Peschaud, and L. Puigjaner. “General approach and tool for the scheduling of complex production systems.” *Computers and Chemical Engineering*, **22**, pp. 395–402, 1998.

[15] H.-M. Hanisch and S. Fleck. “A resource allocation scheme for flexible batch plants based on high-level petri nets.” In *Proc. of CESA96 IMACS Multiconference*, 1996.

[16] D. Harel. “Statecharts, a visual formalism for complex systems.” *Science of Computer Programmming*, **8:3**, pp. 231–274, 1987.

[17] IEC. “IEC 61131 programmable controllers - part 3: Programming languages.” Technical Report, International Electrotechnical Commission, March 1993.

[18] IEC. “IEC 61512-1: Batch control, Part 1, Models and termonology.” Technical Report, International Electrotechnical Commission, August 1997.

[19] IEC. “IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems.” Technical Report, International Electrotechnical Commission, December 1998.

[20] IEC. “IEC 61512-2: Batch control, Part 2, Data structures and guidelines for languages.” Technical Report, International Electrotechnical Commission, November 2001.

[21] IEC. “IEC 61511: Functional safety instrumented systems for the process industry sector.” Technical Report, International Electrotechnical Commission, 2002. Work in progress.

[22] ISA. “ISA S88.01 batch control.” Instrument Society of America, 1995.

[23] P. Jalote. *Fault tolerance in distributed systems*. Prentice Hall, USA, 1994.

[24] K. Jensen and G. Rozenberg. *High-level Petri Nets*. Springer Verlag, 1991.

[25] G. Johanneson. *Object-oriented Process Automation with SattLine*. Chartwell-Bratt Ltd, 1994.

[26] C. Johnsson. *A Graphical Language for Batch Control*. PhD thesis ISRN LUTFD2/TFRT–1051–SE, Department of Automatic Control, Lund Institute of Technology, Sweden, March 1999.

[27] C. Johnsson and K.-E. Årzén. “Grafchart and batch recipe structures.” In *WBF'98 — World Batch Forum*, Baltimore, MD, USA, April 1998.

[28] C. Johnsson and K.-E. Årzén. “Grafchart and recipe-based batch control.” *Computers and Chemical Engineering*, **22:12**, pp. 1811–1828, 1998.

[29] NAMUR. *NAMUR-Emphehlung: Anforderungen an Systeme zur Rezeptfaheweise (Requirements for Batch Control Systems). NAMUR AK 2.3 Funktionen der Betriebs- und Produktionsleitebene*, 1992.

[30] Northwoods Software. “JGo: Java Graphics Library.” Nortwoods home page, http://www.nwoods.com/, 2002.

[31] J. Parshall and L. Lamb. *Applying S88 - Batch Control from a User's Perspective*. ISA - Instrument Society of America, Research Triangle Park, NC, USA, 2000.

[32] H. P. Rosenhof and A. Ghosh. *Batch Process Automation, Theory and Practice*. Van Nostrand Reinhold, 1987.

[33] D. Ruiz, J. Cantón, J. Nougués, A. Espuña, and L. Puigjaner. "On-line fault diagnosis system support for reactive scheduling in multipurpose batch chemical plants." *Computers and Chemical Engineering*, **25**, May, pp. 829–837, May 2001.

[34] K.-E. Årzén. "JGrafchart."
Grafchart home page,
http://www.control.lth.se/~grafchart/, 2002.

[35] K.-E. Årzén, R. Olsson, and J. Åkesson. "Grafchart for Procedural Operator Support Tasks." In *Proceedings of the 15th IFAC World Congress, Barcelona, Spain*, 2002.

[36] Sun Microsystems, Inc. "Java API for XML Processing (JAXP)." JAXP home page, http://java.sun.com/xml/downloads/jaxp.html, 2002.

[37] The GCC Team - Free Software Foundation. "GCC Home Page." GNU home page, http://www.gnu.org/software/gcc/gcc.html, 1999.

[38] M. Tittus and K. Åkesson. "Deadlock avoidance in batch processes." In *Proc. of IFAC World Congress*, Beijing, China, 1999.

[39] I. van Beurden and R. Amkreutz. "Emergency Batch Landing." *InTech*, August, pp. 30–32, August 2002.

[40] W3C - World Wide Web Consortium. "Extensible Markup Language (XML)." W3C home page, http://www.w3.org/XML/, 1997.

[41] WBF's XML Working Group. "Batch Markup Language - BatchML." WBF home page, http://www.wbf.org/, Markup Languages, 2002.

[42] Webgain. "JavaCC - The Java Parser Generator." Webgain home page, http://www.webgain.com/products/java_cc/, 2000.

[43] xmlBlaster.org. “xmlBlaster - Message Oriented Middleware.” xmlBlaster home page, http://www.xmlblaster.org/, 1999.