

LUND UNIVERSITY

Recipe-Based Batch Control Using High-Level Grafchart

Johnsson, Charlotta

1997

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Johnsson, C. (1997). *Recipe-Based Batch Control Using High-Level Grafchart*. [Licentiate Thesis, Department of Automatic Control]. Department of Automatic Control, Lund Institute of Technology, Lund University.

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

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

Document name
LICENTIATE THESIS

Date of issue
June 1997

Document Number
ISRN LUTFD2/TFRT--3217--SE

Author(s)
Charlotta Johnsson

Supervisor
Karl Erik Årzén

Sponsoring organisation
Swedish National Board for Industrial and Technical Development (NUTEK)

Title and subtitle
Recipe-Based Batch Control Using High-Level Grafchart

Abstract

High-Level Grafchart is a graphical programming language for control of sequential processes. Sequential control is important in all kinds of industries: discrete, continuous and batch. Sequential elements show up both on the local control level and on the supervisory control level.

High-Level Grafchart combines the graphical syntax of Grafcet/SFC with high-level programming language constructs and ideas from High-Level Petri Nets. High-Level Grafchart can be used to control sequential processes both on the local level and on the supervisory control level.

The main application area of High-Level Grafchart is control of batch processes, i.e., batch control. A batch process is a special class of sequential processes frequently occurring in chemical, pharmaceutical and food industries. Batch processes and batch control is currently the subject of large interest. A recent standard, called ISA S88.01, provides an important step towards a formal definition of batch systems. The specification of how to produce a batch is called a recipe.

In the thesis it is shown how High-Level Grafchart can be used for recipe structuring. By using the features of High-Level Grafchart in different ways, recipes can be represented in a number of alternative ways. They still however, comply with the standard ISA S88.01. The different structures are presented and discussed. A simulation of a multi-purpose, network structured batch plant has served as a test platform. High-Level Grafchart, the recipe-execution system and the batch plant are implemented in G2, an object-oriented programming environment.

Key words
Grafcet, Sequential Function Charts, Petri Nets, Sequential Control, Batch Processes, Batch Recipes

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
206

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through:
University Library 2, Box 3, S-221 00 Lund, Sweden
Fax +46 46 222 44 22 E-mail ub2@uub2.lu.se

Recipe-Based Batch Control Using High-Level Grafchart

Recipe-Based Batch Control Using High-Level Grafchart

Charlotta Johnsson

Department of Automatic Control
Lund Institute of Technology
Lund, June 1997

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

ISSN 0280–5316
ISRN LUTFD2/TFRT–3217–SE

©1997 by Charlotta Johnsson. All rights reserved.
Printed in Sweden by Reprocentralen, Lunds Universitet.
Lund 1997

Contents

Acknowledgments 11

1. Introduction 12

1.1 Contribution of the Thesis 16

1.2 Published Papers 16

1.3 Outline of the Thesis 18

2. Petri Nets 20

2.1 Basic Concepts 21

2.2 Formal Definition 24

2.3 Petri Net Properties 26

2.4 Petri Net Analysis Methods 27

2.5 Generalized Petri Nets 29

2.6 Other Petri Net classes 30

2.7 Summary 33

3. Grafcet 34

3.1 Syntax 35

3.2 Interpretation Algorithm 39

3.3 Formal Definition 42

3.4 Grafcet vs Petri Nets 47
3.5 Grafcet vs Finite State Machines 48
3.6 Summary 48

4. High-Level Nets 49
4.1 Coloured Petri Nets 49
4.2 Coloured Grafcet 56
4.3 Object Petri Nets and LOOPN 60
4.4 OBJSA Nets 62
4.5 Other High Level Languages influenced by Petri Nets 64
4.6 Summary 65

5. Grafchart 66
5.1 Graphical Language Elements 68
5.2 Actions and Receptivities 73
5.3 Error Handling 74
5.4 Dynamic Behavior 77
5.5 Interpretation Algorithm 79
5.6 Formal Definition 81
5.7 Grafchart vs Grafcet 88
5.8 The G2 Implementation 90
5.9 Applications 98
5.10 Summary 99

6. High-Level Grafchart 100
6.1 Parameterization 101
6.2 Methods and Message Passing 105
6.3 Object Tokens 107

6.4 Multi-dimensional Charts 119
6.5 Implementation 122
6.6 Application 125
6.7 Summary 127

7. Batch Control Systems 128
7.1 Batch Processing 129
7.2 Batch Control 131
7.3 The Batch Standard ISA-S88.01 132
7.4 Summary 140

8. High-Level Grafchart and Batch Control 141
8.1 Related Activities 142
8.2 High-Level Grafchart for Batch Control 144
8.3 A Batch Scenario Implemented in G2 148
8.4 Summary 157

9. Recipe Structuring using High-Level Grafchart . . . 158
9.1 Control Recipes as Grafchart Function Charts 159
9.2 Control Recipes as Object Tokens 165
9.3 Multi-dimensional Recipes 171
9.4 Process Cell Structured Recipes 176
9.5 Resource Allocation 177
9.6 Distributed Execution 181
9.7 Other Batch Related Applications of High-Level Grafchart 183
9.8 Summary 184

10. Conclusions 185

10.1 Future Research Directions 186

11. Bibliography 189

A. An introduction to G2 197

B. A Dynamic Simulator 199

B.1 Total Mass Balance 200

B.2 Component Mass Balance 200

B.3 Energy Balance 201

B.4 Level and Volume 202

B.5 Reaction Rates 202

B.6 Implementation 203

B.7 Notation and Constants 205

Acknowledgments

My first contact with Petri Nets and Grafcet was in 1993, when I did my master thesis in Grenoble, France. When I, later in 1993, started my work at the Department of Automatic Control in Lund, as a PhD-student, I was proposed a project where Grafcet was to be developed into a more sophisticated high-level language. The main application area of the project should be batch control. Since I had much liked what I had learned to know about Grafcet and Petri Nets in France, I agreed on the project. At the same time I was also automatically assigned Karl-Erik Årzén as supervisor.

During the four years that have passed and in spite of moments of blood, sweat and tears, I have not regretted my acceptance of this project. It has been, and will hopefully continue to be, a very interesting and inspiring project. Neither have I had any reason to be disappointed in the allotment of supervisor, Karl-Erik has been, and will hopefully continue to be, very inspiring, helpful, and encouraging. To work with him has been a pleasure.

Tied to the project is an industrial steering committee: Tage Jonsson – ABB Industrial Systems, Lars Pernebo – Alfa Laval Automation, Stefan Johansson – Astra Production Chemicals, Björn Perned – Van den Berghs Food, Carl-Erik Flodmark – Kabi Pharmacia. The feedback from them has been very valuable and I hope the connection and the cooperation with all of them will continue.

The work has been supported by NUTEK, the Swedish National Board for Industrial and Technical Development, Project Regina, 97-00061.



Charlotta Johnsson

1

Introduction

Sequential control is an important part of industrial control systems that has long been disregarded by the control community. The main part of the research has focused on control of continuous processes. However, it is most often the sequential parts in a control system that cause problems. Sequential control is often associated with discrete manufacturing but also in the continuous process industry it constitutes an important part, e.g., for control of start-up, shut-down, production and/or mode changes. The increasing demand of small scale production, quick product-changes, and order-based production increase even further the importance of sequential control.

Grafcet, originally a French standard, has been accepted as an international standard (IEC 848, IEC 1131-3) for representation of sequential control logic at the local level. In the standard, Grafcet is refered to as Sequential Function Charts (SFC). Grafcet has a graphical syntax that has become well accepted in industry. The semantics, or the execution model of Grafcet is well specified. However, it is not always used when implementing Grafcet in industrial control systems. Grafcet, is based on Petri Nets. It can even be seen as a special class of Petri nets. Formal analysis methods that can be used to find out or verify properties of a system exist for Petri Nets. In parallel with the development of Grafcet, Petri Nets have been developed into High-Level Petri Nets which is a richer and more elaborated version of Petri Nets.

The aim of this thesis is to show how Grafcet can be developed into High-Level Grafcet. High-Level Grafcet can, unlike Grafcet, be used to implement all levels in a control system, supervisory as well as

local. The development of High-Level Grafcet has, as much as possible, followed the Grafcet standard. High-Level Grafcet has a syntax similar to Grafcet, i.e., it is based on the concepts of steps, representing the states, and transitions, representing the change of state. As a first step, Grafchart was developed. Grafchart is a Grafcet-based model with an extended syntax for abstraction facilities. A toolbox implementation of Grafchart exists. In addition to the syntax of Grafchart, High-Level Grafcet contains high level programming language constructs and features inspired by High-Level Petri Nets. A toolbox for High-Level Grafcet has been defined and implemented. The toolbox is called High-Level Grafchart. The aim of High-Level Grafchart is to combine the execution model of Grafcet/SFC with the analysis power of Petri Nets. Grafchart and preliminary versions of High-Level Grafchart has been developed in Lund since 1991.

From a systems theoretic point of view sequential control logic belongs to the area of Discrete Event Dynamic Systems (DEDS). This is an area that is relatively new in the control community and where, currently, a large amount of research is performed. A large number of alternative approaches have been introduced for modeling, analysis, and, in certain restricted cases, also for synthesis of DEDS. The largest problem with DEDS is the combinatorial complexity. This makes it very difficult to scale up DEDS approaches so that they can handle industrial size applications.

The research on discrete event dynamic systems can be divided into two main areas:

1. Formal methods for verification and synthesis.
2. Improved tools and languages for controller implementation.

In the first research area, formal specification languages and formal modeling techniques are used to describe the desired behavior of the system, and formal analysis techniques are used to verify that the actual behavior of the system satisfies certain critical properties. A number of approaches have been developed. They are based on, e.g., state machines, Petri nets, logics, and process algebras. The approaches must provide appropriate means for modeling the process and the controller. The model must be able to represent the dynamic and reactive

nature of the process, and allow for proper expression of timing properties. Hence, a strong focus of formal methods is modeling and modeling languages.

Most of the proposed approaches only concern verification. In verification the verifier is presented with a formal model of the process and the control system and a specification of how the process should behave. The verification problem consists of demonstrating that the model satisfies the specification. In the synthesis problem a specification is given that the process should satisfy. The synthesis problem consists of the construction of a controller that ensures the the combined model of the process and the controller fulfills the specification.

In the control community the Supervisory Control Theory (SCT) [Ramadge and Wonham, 1989] has gained a lot of interest. This has also been combined with Grafcet [Charbonnier, 1996], [Charbonnier *et al.*, 1995]. SCT has also been developed in to Procedural Control Theory (PCT) [Sanchez *et al.*, 1995]. Petri nets have also been used as the basis for controller analysis and synthesis, in the form of Controlled Petri Nets [Holloway and Krogh, 1994], [Holloway and Krogh, 1990].

The second research area instead focuses on providing more powerful tools and languages for implementation of discrete event controllers. With better abstraction and structuring possibilities the control problem that should be implemented becomes easier to handle. The focus in this research area is on programming languages rather than on modeling and specification languages. The situation, of having better abstraction and structuring possibilities, can be compared with moving from assembly languages to object-oriented high-level languages in ordinary programming languages. Grafcet can be compared to a simple assembly language.

The two research areas complement each other. Most formal approaches only support verification. Hence, the designer must himself develop the controller. Therefore, it is important to have good structuring mechanisms also in these frameworks. It is also important to be able to use formal approaches for systems that have been designed with high-level sequential languages.

This thesis belongs to the second area. The major aim is to provide better abstraction and structuring possibilities for Grafcet. A secondary

aim of the work is to be able to make use of the available formal methods that exists for Petri Nets. The approach taken is to show that Grafchart and High-Level Grafchart can be translated into Grafcet and/or into Petri Nets and High-Level Petri Nets and then use the available analysis frameworks rather than to develop special analysis frameworks for High-Level Grafchart.

The main application area of High-Level Grafchart and of this thesis is control of batch processes, i.e., batch control. A batch-process is a special class of sequential processes frequently occurring in chemical-, pharmaceutical-, and food industries. These industries are often of the multipurpose-, multiproduct- type, which means that different batches of different products can be produced at the same time in the same plant. The batch plants can have a network structure, i.e., the batch can take several different paths when passing through the plant. The combination multiproduct, network-structured batch plants are the most difficult plants to control. The specification of how to produce a batch is called a recipe. Recipe-based control is a special type of sequential control.

Batch processes and batch control is currently the subject of large interest. A recent international standard, called ISA S88.01, provides an important step towards a formal definition of the terminology, models and functionality of batch systems. The standard mentions the possibility to use Grafcet to structure a recipe and to perform the actions associated with it. However, it is only proposed to be used at the very lowest level of control. No suggestions are made for how to do the overall structuring.

In the thesis it is shown how High-Level Grafchart can be used for recipe structuring. By using the features of High-Level Grafchart in different ways, recipes can be represented in a number of alternative ways. They still, however, comply with S88.01. The different structures are presented and discussed in the thesis. A simulation of a multipurpose, network structured batch plant has served as a test-platform. High-Level Grafchart, the recipe-execution system, and the batch plant are implemented in G2, an object oriented programming environment. G2 is also an industrial environment which makes it possible to directly use the results in industry.

1.1 Contribution of the Thesis

The contributions of this thesis are the following:

- The semantics of Grafchart is defined and the translation between Grafchart and Grafcet is presented.
- The current version of High-Level Grafchart is presented in detail.
- It is shown how High-Level Grafchart can be used in recipe based batch control both at the recipe level and at the equipment control level. A number of alternative ways of representing recipes are presented and discussed. The presented approaches comply with the S88.01 standard.
- It is shown how resource allocation can be integrated with recipe execution using ideas from concurrent programming and Petri Nets.

1.2 Published Papers

The work presented in this thesis is primarily based on the following conference presentations and journal articles:

- Johnsson C. and Årzén K.-E. (1994): "High-Level Grafcet and Batch Control", *Presented at ADPM'94 (Automation of Mixed Processes: Dynamical Hybrid Systems), Brussels, Belgium*
- Johnsson C. and Årzén K.-E. (1996): "Object-Tokens in High-Level Grafchart", *Presented at CIMAT'96 (Computer Integrated Manufacturing and Automation Technology), Grenoble, France*
- Årzén K.-E. and Johnsson C., and (1996): "Object-oriented SFC and ISA-S88.01 recipes", *Presented at World Batch Forum, Toronto, Canada*
- Johnsson C. and Årzén K.-E. (1996): "Batch Recipe Structuring using High-Level Grafchart", *Presented at IFAC'96 (International Federation of Automatic Control) , San Francisco, USA*

- Årzén K.-E. and Johnsson C. (1996): "Object-oriented SFC and ISA-S88.01 recipes", *ISA Transactions, Vol. 35, p. 237-244*
- Årzén K.-E. and Johnsson C. (1997): "Grafchart: A Petri Net/Grafcet Based Graphical Language for Real-time Sequential Control Applications", *Accepted for SNART'97 (Swedish Real-Time Systems Conference), Lund, Sweden*
- Johnsson C. and Årzén K.-E. (1998): "On recipe-based structures using High-Level Grafchart", *Submitted to ADPM'98 (Automation of Mixed Processes: Dynamical Hybrid Systems), Reims, France*
- Johnsson C. and Årzén K.-E. (1998): "Grafchart and its relations to Grafcet and Petri nets", *Submitted to INCOM'98 (Information Control Problems in Manufacturing), Nancy, France*

High-Level Grafchart has also been used in other areas of batch control. This work is only presented marginally in this thesis. More information can be found in the following conference presentations and journal articles:

- Johnsson C., Viswanathan S., Srinivasan R., Venkatasubramanian V. and Årzén K.-E. (1996): "A Model Based Framework for Automating Operating Procedure Synthesis for Batch Chemical Plants", *Presented at Process Plant Safety Symposium, Houston, USA*
- Viswanathan S., Johnsson C., Srinivasan R., Venkatasubramanian V. and Årzén K.-E. (1997): "Automating Operating Procedure Synthesis for Batch Processes: Part 1. Knowledge Representation and Planning Framework", *Submitted to Computers and Chemical Engineering*
- Johnsson C., Viswanathan S., Srinivasan R., Venkatasubramanian V. and Årzén K.-E. (1997): "Automating Operating Procedure Synthesis for Batch Processes: Part 2. Implementation and Application", *Submitted to Computers and Chemical Engineering*
- Simensen J., Johnsson C. and Årzén K.-E. (1996): "A Framework for Batch Plant Information Models", *Internal report, TFRT-7553, Dept. of Automatic Control, Lund Institute of Technology*

- Simensen J., Johnsson C. and Årzén K.-E. (1997): "A Multiple-View Batch Plant Information Model", *Accepted for PSE'97/ESCAPE-7, Trondheim, Norway*

1.3 Outline of the Thesis

The thesis consists of two parts. The first part, Chapter 2 – 6, describes programming languages. It ends with a presentation of High-Level Grafchart, which is the language used later in the thesis. The second part, Chapter 7–9, describes the main application of High-Level Grafchart, recipe-based batch control. It describes batch control systems in general and a recipe management system for batch processes in particular.

Chapter 2 – 5, present the foundation on which High-Level Grafchart is based. In Chapter 2, **Petri Nets** are presented. Petri Nets are a mathematical and graphical model. One main characteristic of the nets are their possibility to be theoretically analyzed. They can be used for simulation, verification and analysis of discrete systems. **Grafcet** is described in Chapter 3. Grafcet is a mathematical model aimed at formal specification and realization of programmable logic controllers (PLC). It uses a graphical and intuitively understandable syntax. Two international standards exist, describing Grafcet and its use in PLC-programming. In the standards, Grafcet is refered to as Sequential Function Charts (SFC) Today, Grafcet or SFC are well known and well accepted in industry. However, even though a strong relation exists between Petri Nets and Grafcet, Petri Nets are not known nor used in industry.

In order to model large systems efficiently, the Petri Net model has been enlarged and developed into High-Level Petri Nets. In Chapter 4, these nets, and other closely related **High-Level Nets** are described.

Chapter 5 describes **Grafchart**. Grafchart is the name of a mathematical sequential control model. It is based on Grafcet but aimed, not only at local level sequential control problems like PLC-programming, but also at supervisory sequential control applications. Grafchart is also the name of an implementation of the Grafchart model. The Grafchart toolbox is implemented in G2.

Chapter 6 presents **High-Level Grafchart**, an extended version of Grafchart. It combines the graphical features of Grafcet/SFC, with high-level programming languages constructs, object-oriented programming structures and ideas from high-level Petri Nets. High-Level Grafchart is, like Grafchart, both a mathematical model and an implementation. The presentation of High-level Grafchart focuses on the implementational aspects.

Chapter 7 – 9 present batch control systems, which is the main application area of High-Level Grafchart. Chapter 7 describes **batch control systems** in general terms. The chapter also gives an overview of ISA S88.01, a recent international standard for batch control systems. The description of how to produce a batch is called a recipe. Chapter 8 combines **High-Level Grafchart and batch control**. It is shown how the different features of High-level Grafchart can be used to implement a batch control system and how they fit the ISA S88.01 standard.

In Chapter 9 the different recipe structures, all implemented with High-level Grafchart, are presented and discussed. By combining different features of High-Level Grafchart, the recipe structures can be largely variated. They all still however, comply with the ISA S88.01 standard. Advantages and drawbacks of the different structures are given. The chapter also contains a discussion on resource allocation, and a presentation of different possible strategies is given. **Recipe structuring using High-Level Grafchart** allows an integration between Petri Nets based analysis methods and Grafcet/SFC based structuring and execution.

The last Chapter, Chapter 10, is dedicated to a summary of the thesis. Some **conclusions** are drawn and possible future work is proposed.

Happy Reading!

2

Petri Nets

Petri Nets were proposed by the German mathematician Carl Adam Petri in the beginning of the 1960s, [Petri, 1962]. C.A. Petri wanted to define a general purpose graphical and mathematical model describing relations between conditions and events. The mathematical modeling ability of Petri Nets makes it possible to set up state equations, algebraic equations, and other models governing the behavior of the modeled system. The graphical feature makes Petri Nets suitable for visualization and simulation.

The Petri Net model has two main interesting characteristics. Firstly, it is possible to visualize behavior like parallelism, concurrency, synchronization and resource sharing. Secondly, there exists a large number of theoretical methods for analysis of these nets. Petri nets can be used at all stages of system development: modeling, mathematical analysis, specification, simulation, visualization, and realization. Petri Nets have been use in a wide range of application areas, e.g., performance evaluation, [Molloy, 1982], [Sifakis, 1978], distributed database systems, [Ozsu, 1985], flexible manufacturing systems, [Murata *et al.*, 1986], [Desrochers and Al'Jaar, 1995], [Silva and Teruel, 1996], logic controller design, [Silva and Velilla, 1982], [Valette *et al.*, 1983], multiprocessor memory systems, [Kluge and Lautenbach, 1982], and asynchronous circuits, [Yoeli, 1987].

Petri Nets have during the years been developed and extended and several special classes of Petri Nets have been defined. These include, e.g.: generalized Petri Nets, synchronized Petri Nets, timed Petri Nets,

interpreted Petri Nets, stochastic Petri Nets, continuous Petri Nets, hybrid Petri Nets, coloured Petri Nets, object-oriented Petri Nets, and multidimensional Petri Nets.

In this chapter a brief overview of ordinary Petri Nets and generalized Petri Nets is given as well as a short description of other Petri Net classes. More detailed presentations can be found in [David and Alla, 1992], [Murata, 1989], [Peterson, 1981].

2.1 Basic Concepts

A Petri Net (PN) has two types of nodes, places and transitions, see Figure 2.1. A place is represented as a circle and a transition is represented as a bar or a small rectangle. Places and transitions are connected by arcs. An arc is directed and connects either a place with a transition or a transition with a place, i.e., a PN is a directed bipartite graph.

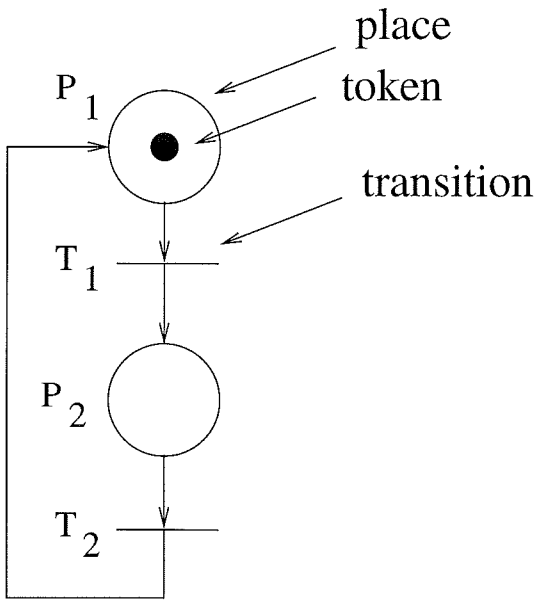


Figure 2.1 A Petri Net.

The set of places is called P and the set of transitions T. Each place contains a nonnegative integer of tokens. The number of tokens contained in a placed is denoted $M(P_i)$ or m_i. The marking of the net, M,

is defined by a column vector where the elements are the number of tokens contained in the corresponding place. The marking defines the state of the PN or the state of the system described by the PN. For the PN given in Figure 2.1 we have:

$$\begin{aligned} \text{Places:} \quad & P = \{P_1, P_2\} \\ \text{Transitions:} \quad & T = \{T_1, T_2\} \\ \text{Marking:} \quad & m_1 = 1,\ m_2 = 0 \\ & M = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned}$$

The set of input (upstream) transitions of a place P_i is denoted ${}^{\circ}P_i$ and the set of output (downstream) transitions is denoted P_i°. Similar notations exist for the input and output places of a transition.

A transition without an input place is called a source transition and a transition without an output place is called a sink transition.

Systems with concurrency can be modeled with Petri Nets using the and-divergence and and-convergence structure, see Figure 2.2 (left). Systems with conflicts or choices can be modeled using the or-divergence and or-convergence structure, see Figure 2.2 (right).

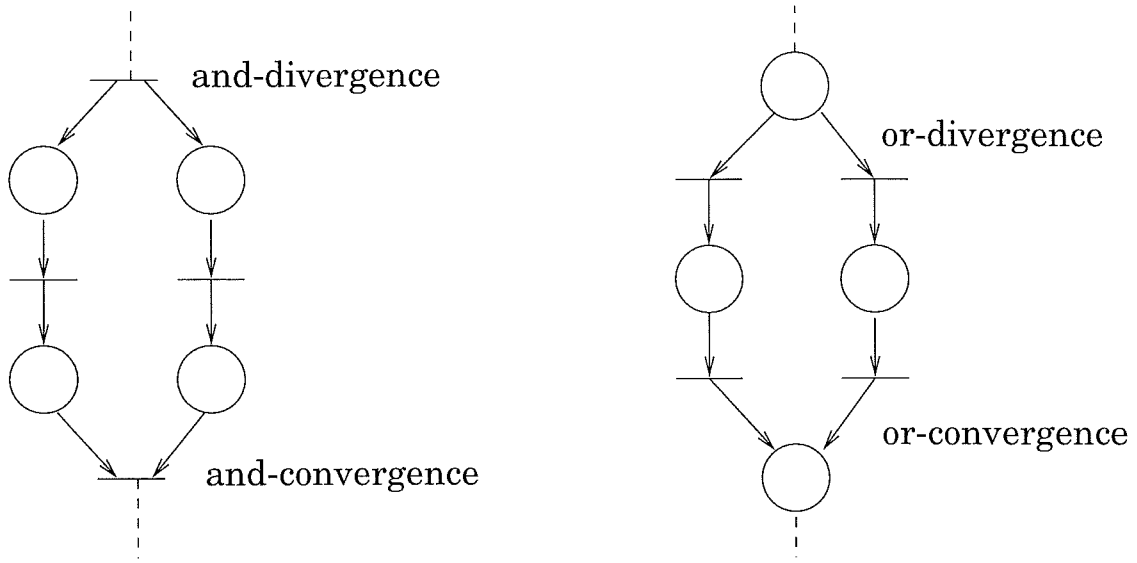


Figure 2.2 Concurrency and conflicts.

State graphs and event graphs are special classes of Petri Net structures. An unmarked Petri Net is a state graph (state machine) if and only if every transition has exactly one input and one output place. If the Petri Net is marked its behavior will be equivalent to a classical state machine if and only if it contains only one token. A Petri Net is an event graph (marked graph, transition graph) if and only if every place has exactly one input and one output transition. In a state graph parallelism cannot be modeled whereas alternatives or conflicts cannot be modeled in an event graph.

Some typical interpretations of transitions and places are shown in Table 2.1, [Murata, 1989].

Input Places	Transition	Output Places
Preconditions	Event	Postconditions
Input data	Computation step	Output data
Input signals	Signal processor	Output signals
Resources needed	Task or job	Resources released
Conditions	Clause in logic	Conditions
Buffers	Processor	Buffers

Table 2.1 Some typical interpretations of transitions and places in Petri Nets.

Dynamic Behavior

A transition is enabled if each of its input places contains at least one token. An enabled transition may or may not fire. The firing of a transition consists of removing one token from each input place and adding one token to each output place of the transition. The firing of a transition has zero duration.

An autonomous PN is a PN where the firing instants are either unknown or not indicated whereas a non-autonomous PN is a PN where the firing of a transition is conditioned by external events and/or time.

2.2 Formal Definition

An unmarked ordinary PN is a 4-tuple $Q =< P, T, Pre, Post >$ *where:*

- $P = \{P_1, P_2, ...P_n\}$ *is a finite, nonempty set of places.*
- $T = \{T_1, T_2, ..., T_m\}$ *is a finite, nonempty set of transitions.*
- $P \cap T = \varnothing$, *i.e., the sets* P *and* T *are disjoint.*
- $Pre : P \times T \rightarrow \{0, 1\}$ *is the input incidence function.*
- $Post : P \times T \rightarrow \{0, 1\}$ *is the output incidence function.*

A marked ordinary PN is a pair $R =< Q, M_0 >$ *in which* Q *is an unmarked ordinary PN and* M_0 *is the initial marking.*

$Pre(P_i, T_j)$ is the weight of the arc connecting place P_i with transition T_j. This weight is 1 if the arc exists and 0 if not. $Post(P_i, T_j)$ is the weight of the arc connecting transition T_j with place P_i. This weight is 1 if the arc exists and 0 if not.

EXAMPLE 2.2.1
Consider the Petri Net given in Figure 2.3.

The net is described by the 5-tuple

$$Q =< P, T, Pre, Post, M_0 >$$

where

$$\begin{aligned} P &= \{P_1, P_2, P_2, P_4\} \\ T &= \{T_1, T_2, T_3, T_4, T_5\} \end{aligned}$$

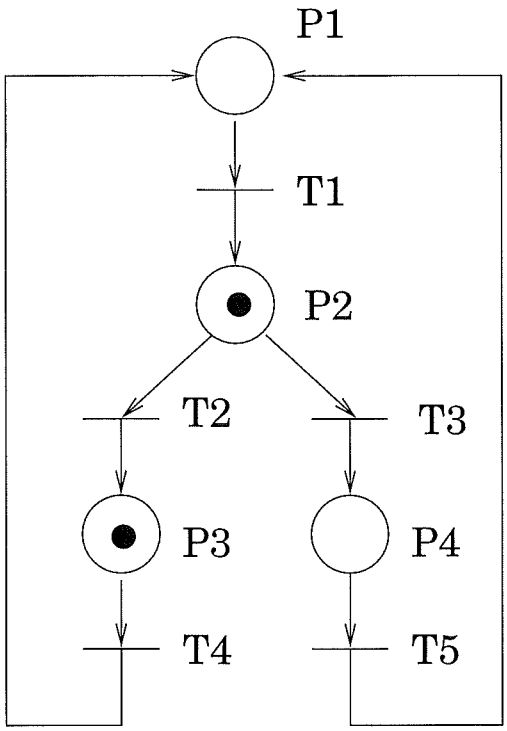


Figure 2.3 A Petri Net.

$$W^- = [Pre(P_i, T_j)] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$W^+ = [Post(P_i, T_j)] = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

#

2.3 Petri Net Properties

The marking of a PN is a column vector whose components are the marking of place P_i. The set of reachable markings from marking M_0 is denoted $*M_0$. The firing of transition T_i from marking M_i will result in marking M_{i+1}, this is denoted:

$$M_i[T_i > M_{i+1}$$

The firing of more than one transition is called a firing sequence and is denoted S.

Home State A PN has a home state M_h for an initial marking M_0 if, for every reachable marking $M_i \in *M_0$, a firing sequence S_i exists such that $M_i[S_i > M_h$.

Reversible A PN is reversible for an initial marking M_0 if M_0 is a home state.

Bounded A place P_i is bounded for an intital marking M_0 if there is a nonnegative integer k such that, for all markings reachable from M_0, the number of tokens in P_i is not greater than k (P_i is said to be k-bounded). A PN is said to be bounded for an initial marking M_0 if all the places are bounded for M_0 (the PN is k-bounded if all the places are k-bounded)

Safe A PN is safe for an initial marking M_0 if, for all reachable markings, each place contains at most one token, i.e., the net is 1-bounded.

Live A transition T_j is live for an initial marking M_0 if, for every reachable marking $M_i \in *M_0$, a firing sequence S from M_i exists, which contains transition T_j. A PN is live for an initial marking M_0 if all its transitions are live from M_0.

Deadlock-free A deadlock is a marking such that no transition is enabled. A PN is said to be deadlock-free for an initial marking M_0 if no reachable marking $M_i \in *M_0$ is a deadlock.

2.4 Petri Net Analysis Methods

There are three main methods for analyzing a PN, i.e., to find out if a certain property holds; (1) the reachability graph and the related coverability tree, (2) linear algebra, and (3) reduction techniques.

(1) Reachability graph and Coverability tree The basic and most simple method is to draw the reachability graph. In this graph the nodes correspond to the reachable markings and the arcs correspond to the firing of transitions. In Figure 2.4 a PN and its reachability graph is shown. This graph can be used to find out that this PN is safe, live, reversible and that it has two repetitive components $T_1T_2T_4$ and $T_1T_3T_5$. Another, equivalent name for the reachability graph is the marking graph.

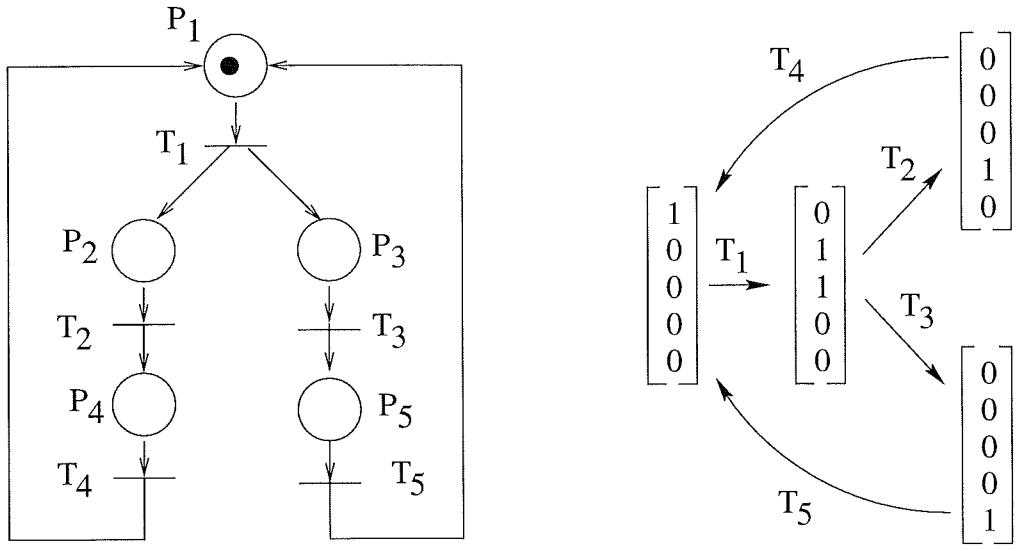


Figure 2.4 A Petri Net and its reachability graph.

In an unbounded net, the number of tokens in a place can be infinite. An unbounded net is represented by associating the symbol ω with the places that might have an infinitely large number of tokens. The resulting graph is now called the coverability tree, see Figure 2.5 (upper right). If the nodes that correspond to the same marking are merged together the coverability graph is obtained, see Figure 2.5 (lower right).

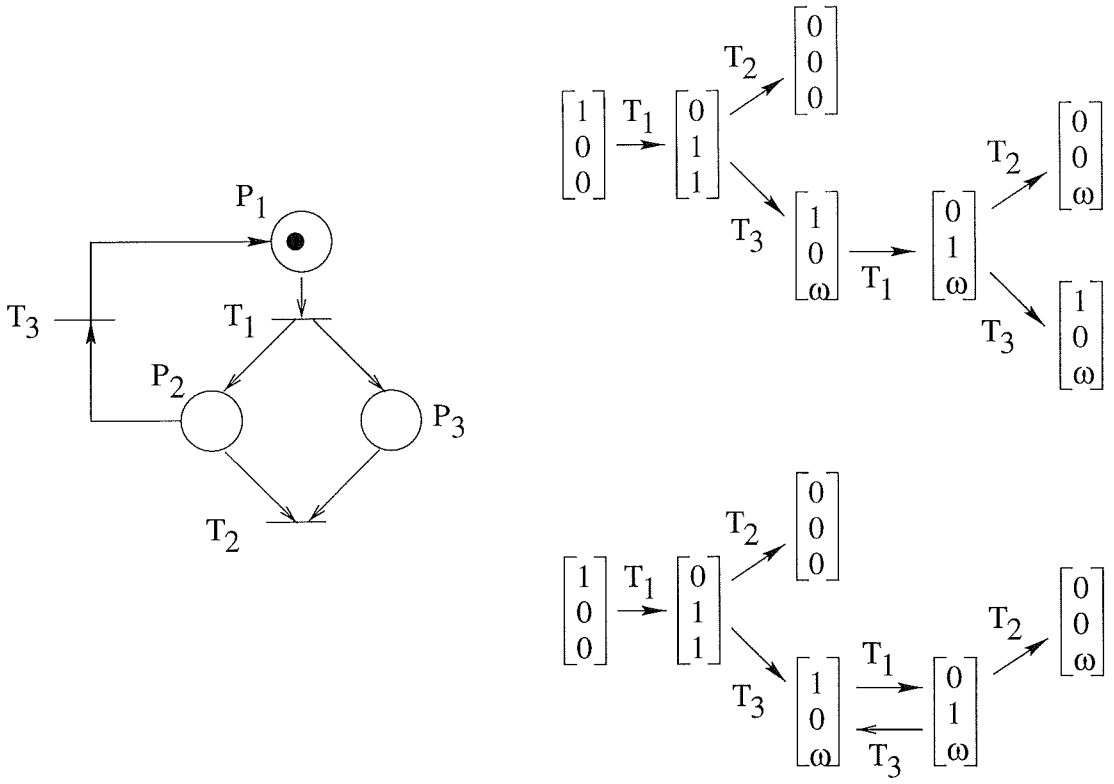


Figure 2.5 A Petri Net and its coverability graph (lower) and coverability tree (upper).

(2) Linear Algebra The linear algebra methods are mathematical methods used to determine the properties of a net. The fundamental equation of a net is given by:

$$M_k = M_i + W \cdot \underline{S}$$

where $\underline{S}$ is the characteristic vector of the firing sequence S that takes marking M_i to marking M_k. The j'th element in the column vector $\underline{S}$ corresponds to the number of firings of transition j in the sequence S. W is the incidence matrix defined as:

$$\begin{aligned} W &= W^+ - W^- \\ W^+ &= [Post(P_i, T_j)] \\ W^- &= [Pre(P_i, T_j)] \end{aligned}$$

The marking of a Petri Net can be changed through the firing of its transitions. If a deadlock situation does not occur the number of firings

is unlimited. However, not all markings can be reached and not all firing sequences can be carried out. The restrictions are given by the invariants of the net. A marking invariant is obtained if a weighted sum of the marking of a subset of the places in a net is always constant, no matter what the firing might be. The places contained in this subset is a conservative component and the vector containing the weights is the P-invariant. The conservative components of a net often have a physical interpretation. If the firing of a certain sequence of transitions results in the same marking as it was started from, the sequence is called a repetitive component. The characteristic vector of the firing sequence is the T-invariant.

The P-invariants and the T-invariants of a net can be deduced by linear algebra methods, [David and Alla, 1992].

(3) Reduction methods Although the construction of the reachability graph is an efficient way of determining the properties of a small-size PN it is not a suitable method when a net has a large number of reachable markings. However, reduction methods exist that transform a large size PN into a PN of a smaller size. The idea of the reduction methods is to successively apply local transformation rules that transform the net into a simpler net, i.e., into a net with a smaller number of places and transitions, while preserving the properties of the net that one wants to investigate.

Four reduction rules exist that preserves the properties: live, bounded, safe, deadlock-free, with home state and conservative. Two reduction methods exist that preserves the invariants. The reduction methods are described in [David and Alla, 1992].

2.5 Generalized Petri Nets

In a generalized PN weights (strictly positive integers) are associated with the arcs. $w(p_j, t_i)$ denotes the weight associated with the arc from place p_j to transition t_i and $w(t_i, p_k)$ denotes the weight associated with the arc from transition t_i to place p_k. In a generalized Petri Net a transition t_i is enabled if each input place p_j of t_i contains at least

$w(p_j, t_i)$ tokens. A firing of an enabled transition t_i consists of removing $w(p_j, t_i)$ tokens from each input place p_j of t_i and adding $w(t_i, p_k)$ tokens to each output place p_k of t_i, see Figure 2.6. If a weight is not explicitly specified it is assumed to be 1.

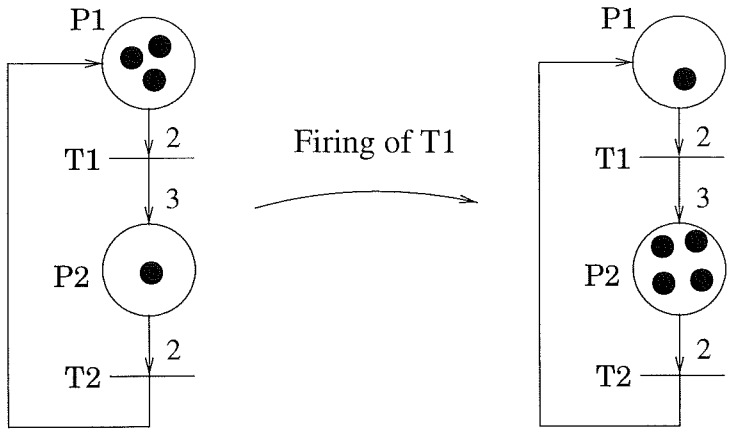


Figure 2.6 A Generalized Petri Net.

All generalized PN can be transformed into ordinary PN.

2.6 Other Petri Net classes

A number of special classes of Petri Nets have been defined. They are briefly described in this section.

Synchronized PN

In ordinary PN an enabled transition may or may not fire , i.e., the ordinary PN are asynchronous. In synchronized Petri Nets external events are associated with the transitions. If a transition is enabled and the associated event occurs, the transition will immediately fire. It is assumed that two external events can never occur simultaneously.

A synchronized PN is a triple $< R, E, Sync >$ *where:*

- *R is a marked PN*
- *E is the set of external events*
- *Sync is a function from T to* $E \cup \{e\}$*, where e is the always occurring event, i.e., an event that is always true.*

Timed PN

Using timed PN, systems with time dependent behavior can be described. The timing can either be associated with the place or with the transition. These nets are called P-timed PN and T-timed PN respectively.

A P-timed PN is a couple $<R, Time>$ such that:

- *R is a marked PN*
- *$Time$ is a function from the set P of places to the set of positive or zero numbers. $Time(P_i) = d_i =$ the timing associated with place P_i.*

When a token arrives in place P_i of a P-timed PN, the token must remain in this place at least d_i time units. During this time the token is unavailable. When time d_i has elapsed, the token becomes available again and can now participate in the enabling of a transition.

A T-timed PN is a couple $<R, Time>$ such that:

- *R is a marked PN*
- *$Time$ is a function from the set T of transitions to the set of positive or zero numbers. $Time(T_j) = d_j =$ timing associated with transition T_j.*

When a token arrives in a place of a T-timed PN, it directly enables the output transition of this place. Before the firing of the output transition can take place the token has to be reserved for at least d_j time units. During this time the token is unavailable for any other activity. When d_j time units has elapsed the transition can fire and the token is removed from the preceding place and placed in the succeeding place of the transition.

Interpreted PN

An interpreted PN is a synchronized, P-timed Petri Net. Associated with the places are operations $O = \{O_1, O_2, \cdots\}$. The interpreted PN

has a data processing part whose state is defined by a set of variables $V = \{V_1, V_2, \cdots\}$. This state is modified by the operations. The variables determine the value of the condition (predicate) C associated with the transition. The operation of an interpreted PN together with its environment is shown in the left part of Figure 2.7. The right part of the same figure shows that the timing d_i and the operation O_i are associated with the place P_i while the event E_j and the condition C_j are associated with transition T_j. The behavior of an interpreted Petri Net is strongly related to Grafcet.

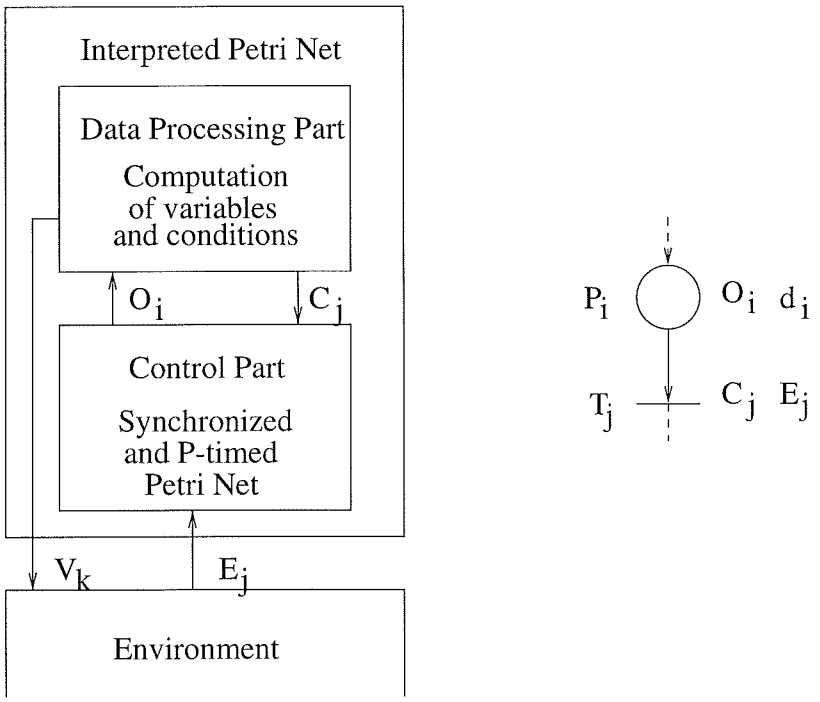


Figure 2.7 An interpreted Petri Net.

Stochastic PN

In a timed PN a fixed duration is associated with each place or with each transition. In stochastic PN a random time is associated with the transition. The most common hypothesis is that the timing is distributed according to an exponential law. The marking at time t, $M(t)$ of a stochastic PN is then an homogeneous Markov process. A Markov chain can thus be associated with every stochastic PN and the probabilities of states in stationary behavior can be calculated.

Continuous PN and Hybrid PN

In a continuous PN the numbers of tokens in a place is given by a real number. It is possible to put weights on the arcs and thereby have a generalized continuous PN, the weights can also be real numbers. A hybrid PN contains one discrete part and one continuous part.

2.7 Summary

Petri Nets can conveniently be used to model systems with e.g., concurrency, synchronization, parallelism and resource sharing. The graphical nature of Petri Nets also makes them suitable to use for visualization and simulation of systems. In addition to this, the nets can be theoretically analyzed with different analysis methods.

3

Grafcet

Grafcet was proposed in France in 1977 as a formal specification and realization method for logical controllers. The name Grafcet was derived from graph, since the model is graphical in nature, and AFCET (Association Française pour la Cybernétique Economique et Technique), the scientific association that supported the work.

During several years, Grafcet was tested in French industries. It fast proved to be a convenient tool for representing small and medium scale sequential systems. Grafcet was therefore introduced in the French educational programs and proposed as a standard to the French association AFNOR where it was accepted in 1982. In 1988 Grafcet, with minor changes, was also adopted by the International Electrotechnical Commission (IEC) as an international standard named IEC 848, [IEC, 1988]. In this standard Grafcet goes under the name Sequential Function Chart (SFC). Seven years later, in 1995, the standard IEC 1131-3, with Grafcet as essential part, arrived, [IEC, 1995]. The standard concerns programming languages used in Programmable Logic Controllers (PLC). It defines four different programming language paradigms together with SFC. No matter which of the four different languages that is used, a PLC program can be structured with SFC.

Because of the two international standards, Grafcet, or SFC, is today widely accepted in industry where it is used as a representation format for sequential control logic at the local PLC level.

In this chapter a brief overview of Grafcet is given. A more thoroughly presentation can be found in [David and Alla, 1992].

3.1 Syntax

Grafcet has a graphical syntax. It is built up by steps, drawn as squares, and transitions, represented as bars. The initial step, i.e., the step that should be active when the system is started, is represented as a double square. Grafcet has support for both alternative and parallel branches, see Figure 3.1.

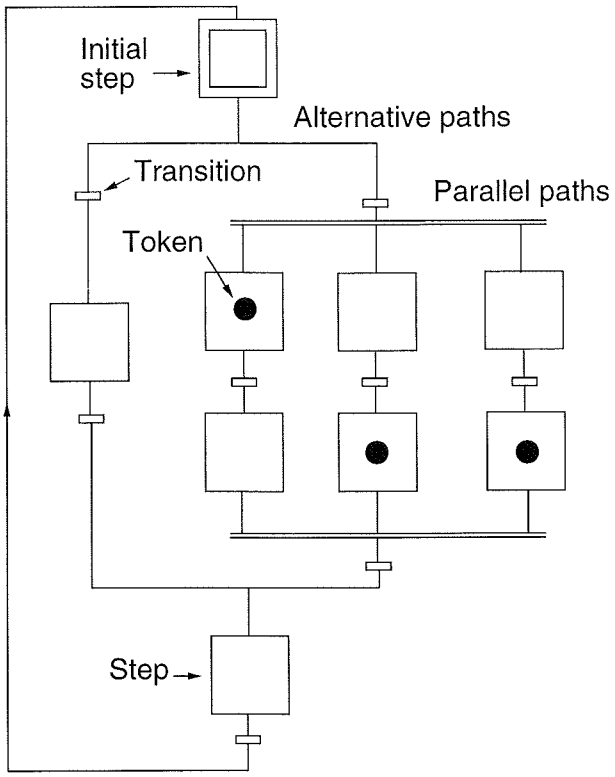


Figure 3.1 Grafcet graphical syntax.

Steps

A step can be active or inactive. An active step is marked with one (and only one) token placed in the step. The steps that are active define the situation or the state of the system. To each step one or several actions can be associated. The actions are performed when the step is active.

Transitions

Transitions are used to connect steps. Each transition has a receptivity. A transition is enabled if all steps preceding the transition are

active. When the receptivity of an enabled transition becomes true the transition is fireable. A fireable transition will fire immediately. When a transition fires the steps preceding the transition are deactivated and the steps succeeding the transition are activated, i.e., the tokens in the preceding steps are deleted and new tokens are added to the succeeding steps.

Actions

There are two major categories of actions: level actions and impulse actions. A level action is modeled by a binary variable and has a finite duration. The level action remains set all the time while the step, to which the action is associated, is active. A level action may be conditional or unconditional. An impulse action is responsible for changing the value of a variable. The variable can, but must not, be a binary variable. An impulse action is carried out as soon as the step changes from being inactive to active. A variable representing time may be introduced to create time-delayed actions and time-limited actions. A level action can always be transformed into an impulse action. The opposite is, however, not always true.

A situation can be stable or unstable. If the transition following a step is immediately fireable when the step becomes active, the situation is said to be unstable. An impulse action is carried out even if the situation is unstable whereas a level action is performed only if the situation is stable. Between two different external events it is assumed that there is always enough time to reach a stable situation.

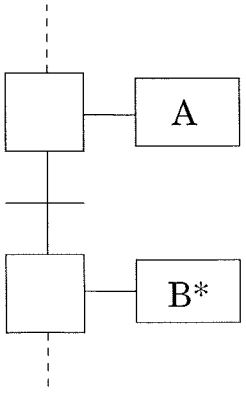


Figure 3.2 Level and impulse actions.

In Figure 3.2 two steps are shown. A level action, A, is associated with the upper step and an impulse action, B*, is associated with the lower step.

Receptivities

Each transition has a receptivity. A receptivity may either be a logical condition, an event, or an event and a condition. In Figure 3.3 three transitions and their receptivities are shown. The receptivity of the first transition is an event, $\uparrow x$. The receptivity of the second transition is a condition, y, and the receptivity of the last transition is a combination of a condition and an event, $x \cdot \uparrow z$.

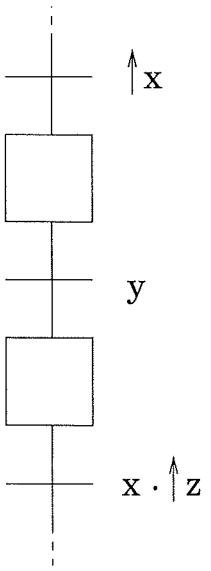


Figure 3.3 Three receptivities.

Macro Steps

To facilitate the description of large and complex systems macro steps can be used. A macro step is a step with an internal representation that facilitates the graphical representation and makes it possible to detail certain parts separately. A macro step has one input and one output step. When the transition preceding the macro step fires the input step of the macro step is activated. The transition succeeding the macro step does not become enabled until the execution of the macro step reaches its output step. The macro step concept is shown in Figure 3.4.

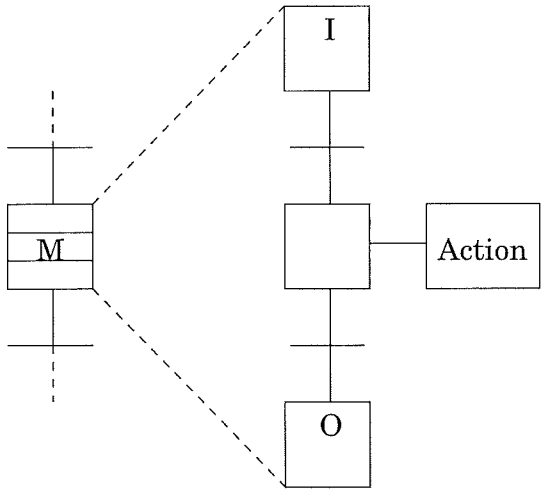


Figure 3.4 A macro step.

Dynamic behavior

The dynamic behavior of Grafcet is defined by five rules, [David, 1995].

1. The initial situation of a Grafcet is determined by its initial steps.
2. A transition is enabled if all of its previous steps are active. A enabled transition is fireable if its associated receptivity is true. A fireable transition is immediately fired.
3. Firing of a transition results in deactivation of its previous step and a simultaneous activation of its following steps.
4. Simultaneously fireable transitions are simultaneously fired.
5. If a step is to be simultaneously activated and deactivated it remains active.

EXAMPLE 3.1.1
In Figure 3.5 (left) a system consisting of a tank is shown. The tank has an inlet valve V_1 and an outlet valve V_2. There are two level sensors L_0 and L_1, one temperature sensor T and one heater Q.

The Grafcet for controlling the tank system is shown in Figure 3.5 (right). When the system is started, valve V_1 should open and the filling should start. When the level in the tank reaches L_0, the heating should start. The tank is now heated and filled in parallel. When the

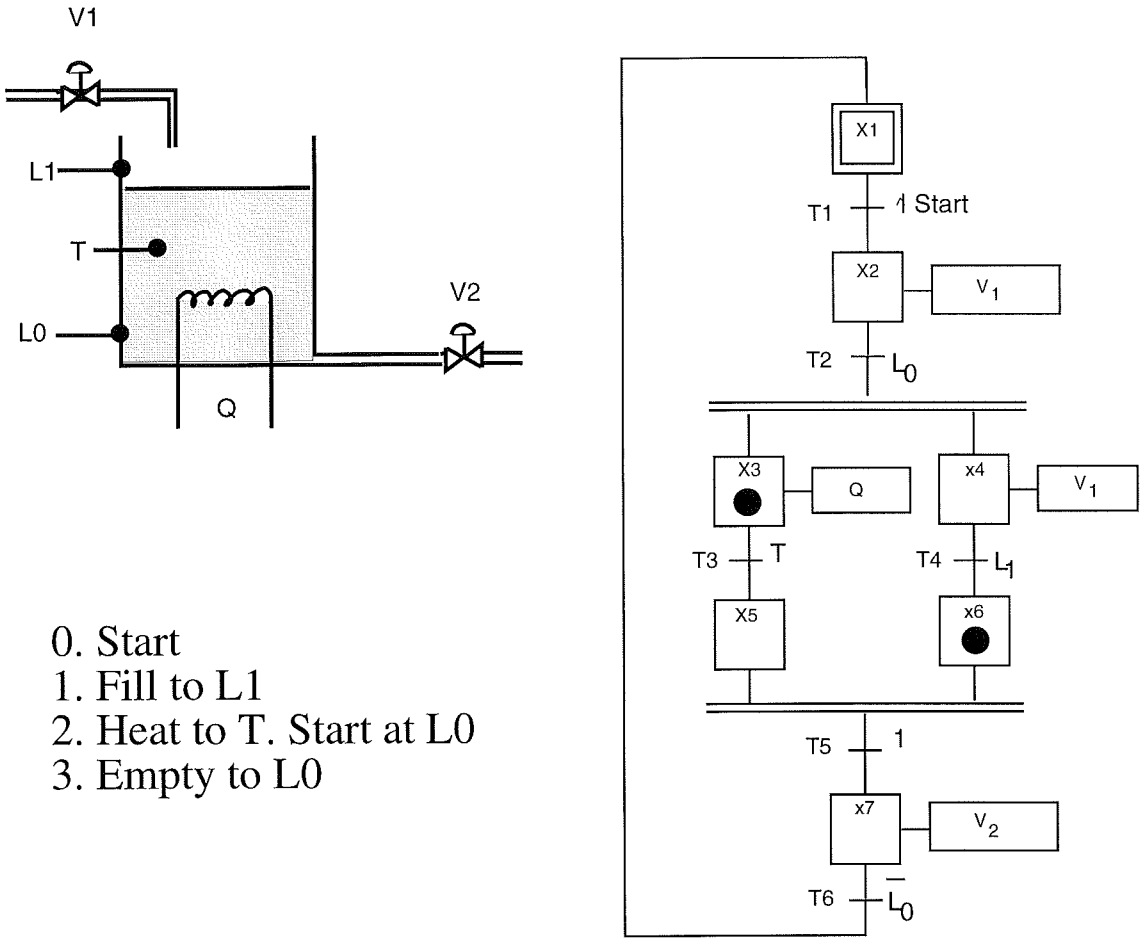


Figure 3.5 A tank example.

level in the tank reaches L_1 the filling is stopped and when the temperature reaches T the heating is stopped. When both the right level and the right temperature are reached, the tank is emptied. This is done by opening the outlet valve V_2. When the system is empty, the sequence can be restarted. All the actions shown in Figure 3.5 (right) are level actions, e.g., the valve V_2 is open (boolean variable V2=1) as long as step x7 is active.

3.2 Interpretation Algorithm

A Grafcet describes a logic controller (which is a sequential machine), i.e., it specifies the relation between the input sequence and the output

sequence. The interpretation of a Grafcet must be without ambiguity, i.e., the same input sequence applied to a Grafcet must result in the same output sequence, independently of the person interpreting the Grafcet. The output sequences must be equivalent both with respect to the relative order between the actions and with respect to the timing. The aim of the interpretation algorithm given below, [David and Alla, 1992], is to define exactly how a Grafcet should be interpreted so that there will never be any ambiguities. The algorithm describes the theoretical understanding of the behavior of the Grafcet.

The interpretation algorithm is based on two assumptions:

1. The system to be controlled by the Grafcet is assumed to be slower than the logic controller implemented by the Grafcet.
2. Two external events do never occur simultaneously.

Algorithm:

1. Initialization: activate the initial steps and execute the associated impulse actions. Go to Step 5
2. When a new external event occurs, determine the set T_1 of transitions fireable on occurrence of this event. If T_1 is not empty, go to Step 3. If T_1 is empty, modify, if necessary, the state of the conditional actions associated with the active steps. Go to Step 2 and wait for a new external event.
3. Fire all the fireable transitions. Go to Step 6 if the situation remains unchanged after this firing.
4. Execute the impulse actions associated with the steps that became active at Step 3.
5. Determine the set of transitions, T_2, that are fireable on the occurrence of the event e, i.e., the transitions that have a receptivity that is always true. Go to Step 3 if T_2 is not empty.
6. A stable situation is reached
   (a) Determine the set A_0 of the level actions which should be deactivated.

(b) Determine the set A_1 of the level actions that should activated.

(c) Set all the actions that belong to A_0 but not to A_1 to 0. Set all the actions that both belong to A_0 and to A_1 to 1. Go to Step 2.

The algorithm searches for stable situations (step 6). However, the algorithm is based on the assumption that a finite number of iteration always results in a stable situation. Therefore, it might seem as if it is not necessary to do the search. Another simpler algorithm exists where the search is not done.

However, the shorter algorithm does not give a correct result if the net contains unstable situations. Even though the shorter algorithm does not always give a correct output, this algorithm is the one used most often in industrial implementations of Grafcet, [Gaffe, 1996].

An interpretation algorithm that works in a synchronous manner has been developed, [Gaffe, 1996]. The output of a Grafcet interpreted with the synchronous algorithm will in most cases be equivalent to that of the algorithm with search for stability. However, since the output sometimes differ, a Grafcet together with a synchronous interpretation algorithm has been given a special name, SGrafcet.

Implementational aspects

The interpretation algorithm to chose when implementing a Grafcet is the algorithm where a search for stable situations is done. However, certain practical aspects must be taken into consideration if the real behavior should be consistent with the interpretation (theoretical behavior). The algorithm is based on the assumption that no external events can occur simultaneously. However, when a logic controller, i.e., a Grafcet, is implemented, two events may occur so close in time that there is no technical mean to distinguish which event occurred the first. This might cause problems in a conflict situation (alternative path). Situations like this should therefore be avoided. This is done by making the receptivities mutually exclusive.

3.3 Formal Definition

A Grafcet can formally be defined in many ways. The definition does only describe the structure of a Grafcet, its actions and its receptivities. To execute a Grafcet an interpretation algorithm must be applied. In other words, the formal definition defines the syntax whereas the interpretation algorithm defines the semantics.

A Grafcet can be defined as a 5-tuple $G =< \mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}, I >$ *where:*

- $\mathcal{V}_r$ *is the set of variables given by:* $\mathcal{V}_r = \{\mathcal{V}_{ext} \cup \mathcal{V}_{int}\}$. $\mathcal{V}_{ext}$ *is the set of variables originating from the plant.* $\mathcal{V}_{int}$ *is the variables corresponding to the internal states of the Grafcet.* $\mathcal{V}_{ext}$ *and* $\mathcal{V}_{int}$ *can either be conditions, i.e., variables that are true or false, or events, i.e., variables that change values.*
- $\mathcal{V}_a$ *is the set of variables issued to the plant.* $\mathcal{V}_a$ *can either be continuous, i.e., a variable that is set to true or false, or discontinuous, i.e., a variable that is given a new value.*
- $\mathcal{X}$ *is the set of steps,* $\mathcal{X} = \{x_1, x_2, x_3, ...\}$. *A step* $x_i \in \mathcal{X}$ *is defined by* $\{action(x_i)\}$, *where* $action(x_1) \in \mathcal{V}_a$ *defines the actions associated with the step* x_i.
- $\mathcal{T}$ *is the set of transitions,* $\mathcal{T} = \{t_1, t_2, t_3, ...\}$. *A transition* $t \in \mathcal{T}$ *is defined by the 3-tuple* $\{X_{PR}(t), X_{FO}(t), \varphi(t)\}$, *where* $X_{PR}(t)$ *is the set of previous steps of t,* $X_{FO}(t)$ *is the set of the following steps of t, and* $\varphi(t)$ *is the receptivity associated with t.* $X_{PR}(t) \in \mathcal{X}$, $X_{FO}(t) \in \mathcal{X}$ *and* $\varphi(t) \in \mathcal{V}_r$.
- I *is the set of initial steps,* $I \subseteq \mathcal{X}$.

EXAMPLE 3.3.1
The Grafcet given in Figure 3.5 is represented by the 5-tuple G.

$$G =< \mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}, I >$$

where:

$$\mathcal{V}_r = \{\uparrow Start, L_0, T, L_1, \overline{L_0}\}$$

$$\mathcal{V}_a = \{V_1, Q, V_2\}$$

$$\mathcal{X} = \{X1, X2, X3, X4, X5, X6, X7\}$$

$$\begin{aligned} action(X2) &= V_1 \\ action(X3) &= Q \\ action(X4) &= V_1 \\ action(X7) &= V_2 \end{aligned}$$

$$\mathcal{T} = \{T1, T2, T3, T4, T5, T6\}$$

$$\begin{aligned} X_{PR}(T1) &= X1 & X_{FO}(T1) &= X2 \\ X_{PR}(T2) &= X2 & X_{FO}(T2) &= \{X3, X4\} \\ X_{PR}(T3) &= X3 & X_{FO}(T3) &= X5 \\ X_{PR}(T4) &= X4 & X_{FO}(T4) &= X6 \\ X_{PR}(T5) &= \{X5, X6\} & X_{FO}(T5) &= X7 \\ X_{PR}(T6) &= X7 & X_{FO}(T6) &= X1 \end{aligned}$$

$$\begin{aligned} \varphi(T1) &= \uparrow Start \\ \varphi(T2) &= L_0 \\ \varphi(T3) &= T \\ \varphi(T4) &= L_1 \\ \varphi(T5) &= 1 \\ \varphi(T6) &= \overline{L_0} \end{aligned}$$

$$I = \{X1\}$$

#

If the macro steps are to be considered, the definition will be as follows:

A Grafcet, with macro steps taken into consideration, can be defined as a 6-tuple $G = < \mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}, \mathcal{M}, I >$, *where:*

- $\mathcal{V}_r$ *is defined as before.*
  $\mathcal{V}_r$ *can be divided into two subsets,* $\mathcal{V}_{r\mathcal{X}}$ *and* $\mathcal{V}_{r\mathcal{M}}$. $\mathcal{V}_{r\mathcal{X}}$ *are the variables associated with the transitions not included in the macro steps.* $\mathcal{V}_{r\mathcal{M}}$ *are the variables associated with the transitions included in the macro steps. The two subsets are not necessarily disjoint.*

- $\mathcal{V}_a$ *is defined as before.*
  $\mathcal{V}_a$ *can be divided into two subsets,* $\mathcal{V}_{a\mathcal{X}}$ *and* $\mathcal{V}_{a\mathcal{M}}$. $\mathcal{V}_{a\mathcal{X}}$ *are the variables associated with the steps not included in the macro steps.* $\mathcal{V}_{a\mathcal{M}}$ *are the variables associated with the steps included in the macro steps. The two subsets are not necessarily disjoint.*

- $\mathcal{X}$ *is defined as before.*
  $action(x_i) \in \mathcal{V}_{a\mathcal{X}}$.

- $\mathcal{T}$ *is defined as above.*
  $X_{PR}(t) \in \{\mathcal{X} \cup \mathcal{M}\}, X_{FO}(t) \in \{\mathcal{X} \cup \mathcal{M}\}$ *and* $\phi(t) \in \mathcal{V}_{r\mathcal{X}}$.

- $\mathcal{M}$ *is the finite set of macro steps,* $\mathcal{M} = \{M_1, M_2, \cdots\}$.
  A macro step M_i *is defined as a 7-tuple* $M_i = < \mathcal{V}_{ri}, \mathcal{V}_{ai}, \mathcal{X}_i, \mathcal{T}_i, \mathcal{M}_i, In_i, Out_i >$, *where:*

  - $\mathcal{V}_{ri}, \mathcal{V}_{ai}, \mathcal{X}_i, \mathcal{T}_i$ *are defined as* $\mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}$ *defined above.*
  - $\mathcal{M}_i$ *is the set of macro steps,* $\mathcal{M}_i = \{M_{i,1}, M_{i,2}, \cdots\}$. *The macro steps are not allowed to be infinitely recursive.*
  - In_i *is the input step of the macro step,* $In_i \subseteq \mathcal{X}_i$.
  - Out_i *is the output step of the macro step,* $Out_i \subseteq \mathcal{X}_i$.

  $\mathcal{V}_{r\mathcal{M}} = \{\mathcal{V}_{r1} \cup \mathcal{V}_{r2} \cup \cdots\}$.
  $\mathcal{V}_{a\mathcal{M}} = \{\mathcal{V}_{a1} \cup \mathcal{V}_{a2} \cup \cdots\}$.

- I *is the set of initial steps,* $I \subseteq \mathcal{X}$.

EXAMPLE 3.3.2
The Grafcet given in Figure 3.6 is represented by the 6-tuple G.

$$G = < V_r, V_a, \mathcal{X}, \mathcal{T}, \mathcal{M}, I >$$

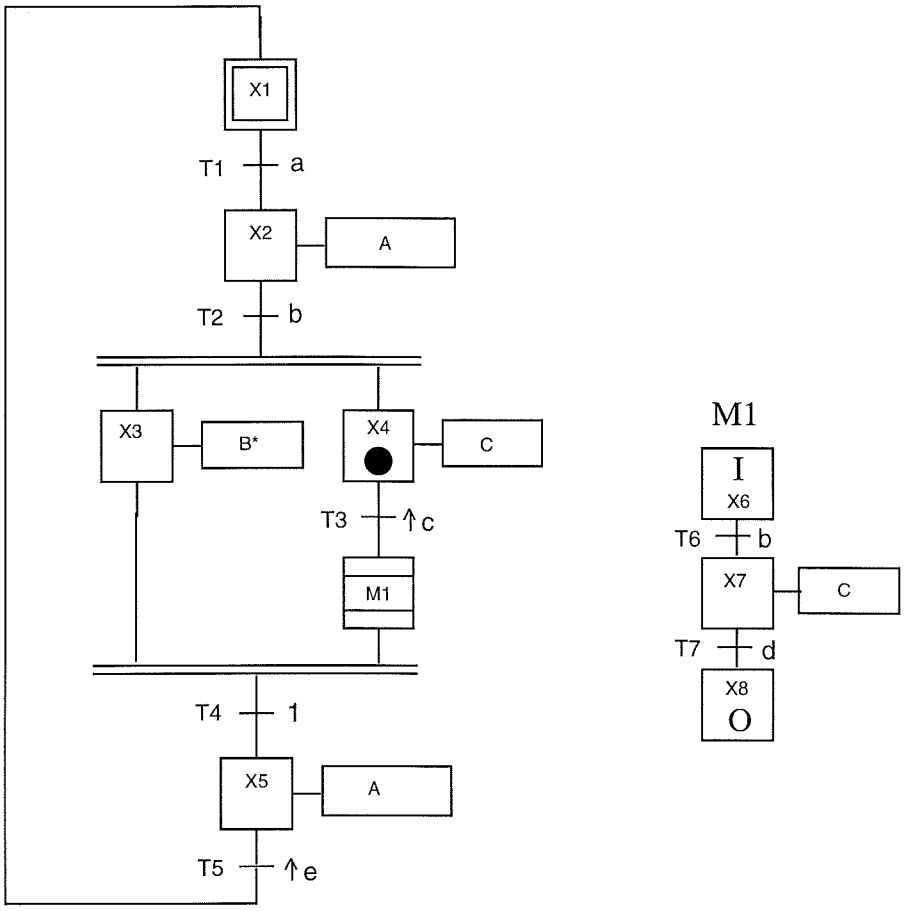


Figure 3.6 A Grafcet with a macro step.

$$\begin{aligned} V_r &= \{V_{rX} \cup V_{r\mathcal{M}}\} \\ V_{rX} &= \{a, b, \uparrow c, \uparrow e\} \end{aligned}$$

$$\begin{aligned} V_a &= \{V_{aX} \cup V_{a\mathcal{M}}\} \\ V_{aX} &= \{A, B*, C\} \end{aligned}$$

$$\mathcal{X} = \{X1, X2, X3, X4, X5\}$$

$$action(X2) = A \qquad action(X3) = B*$$
$$action(X4) = C \qquad action(X5) = A$$

$$\mathcal{T} = \{T1, T2, T3, T4, T5\}$$

$$X_{PR}(T1) = X1 \qquad X_{FO}(T1) = X2$$
$$X_{PR}(T2) = X2 \qquad X_{FO}(T2) = \{X3, X4\}$$
$$X_{PR}(T3) = X4 \qquad X_{FO}(T3) = M1$$
$$X_{PR}(T4) = \{X3, M1\} \qquad X_{FO}(T4) = X5$$
$$X_{PR}(T5) = X5 \qquad X_{FO}(T5) = X1$$

$$\varphi(T1) = a \qquad \varphi(T2) = b$$
$$\varphi(T3) = \uparrow c \qquad \varphi(T4) = 1$$
$$\varphi(T5) = \uparrow e$$

$$\mathcal{M} = \{M1\}$$
$$V_{r\mathcal{M}} = V_{r1} = \{b, d\}$$
$$V_{a\mathcal{M}} = V_{a1} = \{C\}$$
$$\mathcal{X}_1 = \{X6, X7, X8\}$$
$$action(X7) = C$$
$$\mathcal{T}_1 = \{T6, T7\}$$
$$X_{PR}(T6) = X6 \qquad X_{FO}(T6) = X7$$
$$X_{PR}(T7) = X7 \qquad X_{FO}(T7) = X8$$
$$\varphi(T6) = b \qquad \varphi(T7) = d$$
$$\mathcal{M}_1 = \varnothing$$
$$In_1 = X6$$
$$Out_1 = X8$$

$$I = \{X1\}$$

3.4 Grafcet vs Petri Nets

Grafcet and Interpreted Petri Nets, see Chapter 2.6, have several similarities:

1. Both models have two types of nodes: steps and transitions for Grafcet and places and transitions for Petri Nets.
2. In both models, the net execution is synchronized by external events.

However, there are also some differences:

1. The marking of a Grafcet is boolean whereas the marking of a Petri Net is numerical. However, a Petri Net like formalism with boolean marking, called Condition/Event Nets, has been defined by C.A. Petri [Petri, 1962].
2. All simultaneously fireable transitions will be simultaneously fired in a Grafcet whereas in an Interpreted Petri Nets the transitions will be fired in a sequence, called the complete firing sequence. If the sequence is not maximal all the fireable transitions will not be fired. This difference means that, for a Grafcet, in an or-divergence situation, with all receptivities being true, the transitions in both branches will be fired and both the "alternative" branches will be executed. This is often not the designers intention and it is recommended that these transitions are made mutually exclusive. An Interpreted Petri Net treats the situation by nondeterministically choosing one of the branches.
3. The conditions used in Grafcet can depend on the state of the marking, this is not the case in an Interpreted Petri Net.

The first two differences are the most fundamental ones. The third difference can be avoided by rewriting the Interpreted Petri Net.

3.5 Grafcet vs Finite State Machines

The classical models for describing sequential systems are state machines. Two different types exist; Mealy machines and Moore machines, [Mealy, 1955], [Moore, 1956]. The output from a Mealy-machine depends only on the internal state whereas the output from a Moore machine depends both on the internal state and the inputs.

Both Mealy and Moore machines can directly be transformed into a Grafcet whereas a transformation the other way is not always possible. Moreover, it can be shown that the number of steps (and transitions) required in a Grafcet is at most equal to the number of states (and transitions) required for the description by the corresponding state machine.

3.6 Summary

Grafcet was developed in France in the mid seventies. It was proposed as a formal specification and realization method for logical controllers. Grafcet has since become well known worldwide through the two international standards, IEC 848 from 1988 and IEC 1131-3 from 1995. In these standards Grafcet, with minor changes, is referred to as Sequential Function Charts (SFC). The aim of Grafcet, or SFC, in the standards has gradually shifted from being a representation format for logical controllers towards being a graphical programming language for sequential control problems at the local level. One main advantage of Grafcet, or SFC, is its simple and intuitively understandable graphical syntax. There are steps, representing the states and there are transitions, representing the change of states. Today Grafcet is widely used and very well accepted in industry.

4

High-Level Nets

If a system has several parts that are identical these parts should, using ordinary nets, be modeled by as many identical nets as there are parts. This kind of problem is of no importance for small systems but it might be catastrophic for the description of a large system. Real-world systems are often large and contain many parts which are similar but not identical [Jensen, 1992]. Using ordinary nets, these parts must be represented by disjoint subnets with a nearly identical structure. This means that the total net becomes very large and it becomes difficult to see the similarities between the individual subnets representing the similar parts. To make the representation more compact, efficient, and manageable the different identical parts could be modeled by one net where each part is represented by an individual token. In order to distinguish the different tokens, i.e., the different parts, an identifier is associated with each token. This is the main idea behind all High-Level Nets, [Jensen and Rozenberg, 1991].

In this chapter some different High-Level Nets are described. The different models presented are not supposed to give a complete overview of the field but a rough idea of what has been done in the area.

4.1 Coloured Petri Nets

Coloured Petri Nets exist in two main different versions. The first version was developed in 1981, [Jensen, 1981], and the second version,

which is richer and more advanced, was developed in 1986, [Jensen, 1992]. In both versions, the identifier, associated with each token is called the token colour. Petri Net models were the tokens can be identified, like coloured Petri Nets, are called High-Level Petri Nets.

Jensen compares the step from ordinary low-level Petri Nets to high-level nets with the step from assembly languages to modern programming languages. In low level nets there is only one type of token which means that the state of the place is described by an integer (or by a boolean). In high-level nets, each token can carry complex information or data. The state of the system can therefore be more precise [Jensen, 1992].

EXAMPLE 4.1.1
An example taken from [David and Alla, 1992] illustrates the idea of coloured Petri Nets. Figure 4.1 represents two identical systems. In each system, the truck can move either to the left or to the right. As soon as the truck reaches one end, it changes direction and sets of again towards the other end.

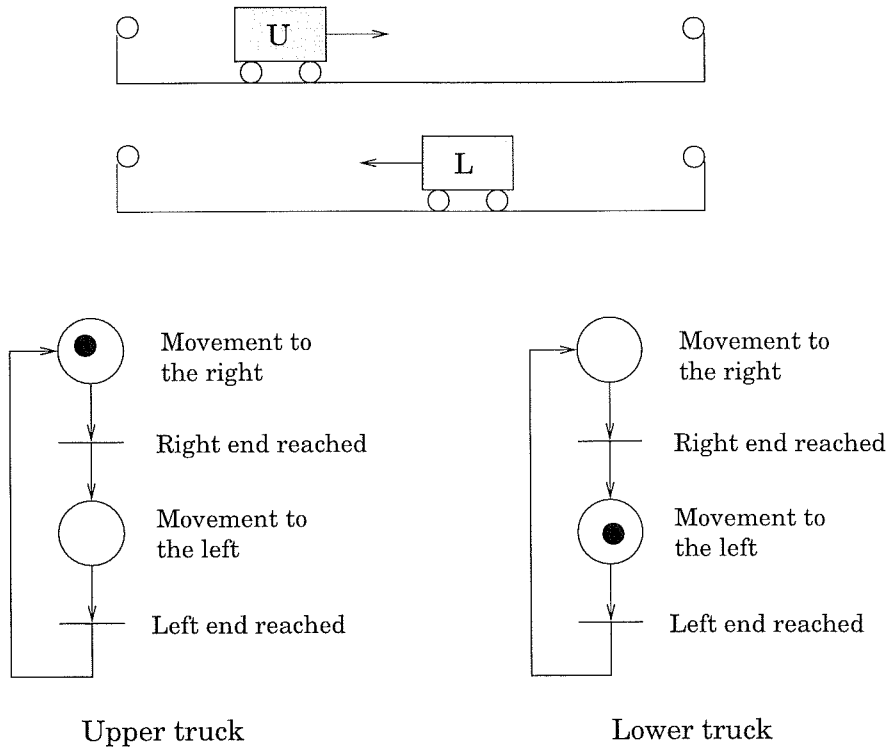


Figure 4.1 Two identical systems and their associated Petri Nets.

The two systems can be modeled by two separate nets, as indicated in Figure 4.1. Each net contains one token. The two nets can also be modeled in one single net as shown in Figure 4.2. In order to distinguish the two trucks from each other, the tokens are given individual names or colours. The token corresponding to the upper truck is labeled $< U >$ and the truck corresponding to the lower truck is labeled $< L >$.

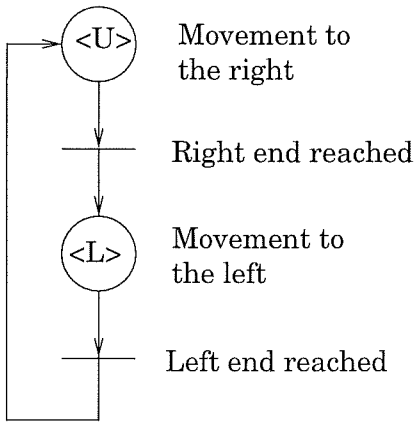


Figure 4.2 One coloured Petri Net representing two identical systems.

#

The operation performed when transforming an ordinary Petri Net to a Coloured Petri Net is called folding. The reverse operation is called unfolding.

The properties of a coloured Petri Net are the same as those for an ordinary Petri Net, i.e., firing sequences, bounded, live, deadlock-free, etc.

Coloured Petri Nets version 1981

To each transition a set of firing colours is associated. These represent the different firing possibilities. A transformation of colours may occur during the firing of a transition. To be able to represent colour transformations the concept of arc inscriptions is introduced. The arc inscriptions are functions associated with the input and the output arc of a transition. The functions on the input arcs determine the numbers and the colours of the tokens that will be removed from the input places when the transition fires with respect to a certain firing colour. Similarly, the functions on the output arcs determine the colours and

the number of the tokens that will be added in the output places of the transition. The identity function is associated to arcs without colour transformation. The identity function is however, very often not explicitly written out.

EXAMPLE 4.1.2
An example of a small coloured PN is given in Figure 4.3. In the figure it is shown how the colour functions, associated with the arcs, can be used.

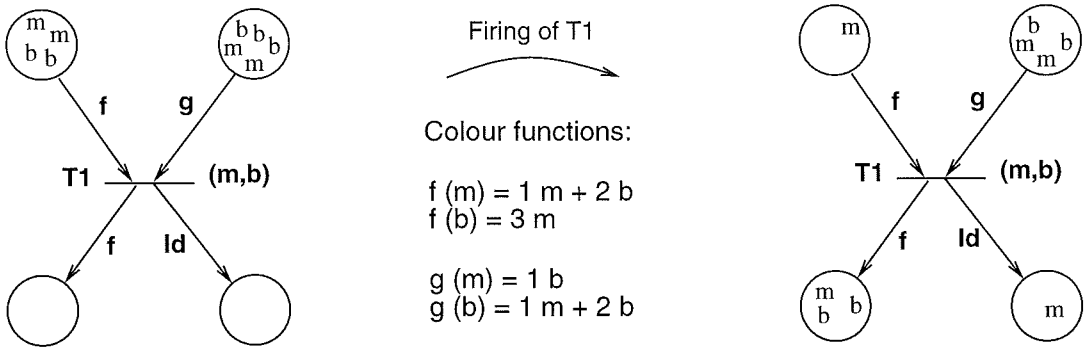


Figure 4.3 A simple coloured PN.

Transition $T1$ is enabled with respect to the firing colour m since the input places contains at least $f(m)$ and $g(m)$ tokens, respectively. The transition is not enabled with respect to the firing colour b. This is due to the left input place that does not fulfill the condition of containing $f(b)$ tokens. When the transition fires with respect to the firing colour m, $f(m)$ and $g(m)$ tokens will be removed from the respective input place and $f(m)$ and $Id(m) = 1m$ tokens will be added to the respective output place.

#

Complex colours consisting of two or more subcolours can be defined. This can further reduce the size of the model of a system. However, if the system is to much compactified, it becomes hard to read and understand.

EXAMPLE 4.1.3
In Figure 4.2 a single colour is used to identified each truck. The system can however be even more compactly described using complex colours.

The upper and the lower truck can both move either to the left or to the right. In Figure 4.2 the two directions are represented by two separate places. If each truck instead is represented by a complex colour where the first subcolour indicates if it is the upper, U, or lower, L, truck and the second subcolour indicates if the truck moves to the right, r, or to the left, l, the system can be described by only one place and one transition, see Figure 4.4.

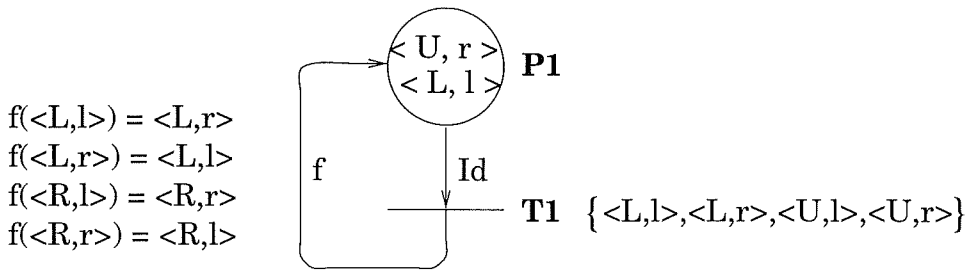


Figure 4.4 A coloured Petri Net with complex colours.

#

Formal definition

A coloured PN is a sextuple $R =< P, T, Pre, Post, M_0, C >$ *where:*

- P *is the set of places.*
- T *is the set of transitions.*
- Pre *and* $Post$ *are functions relating the firing colours to colours of the tokens.*
- M_0 *is the initial marking.*
- $C = \{C_1, C_2, \ldots\}$ *is the set of colours.*

EXAMPLE 4.1.4
The coloured Petri Net shown in Figure 4.4 can be described by the sextuple R.

$$R = \{P, T, Pre, Post, M_0, C\}$$

where

$$
\begin{aligned}
P &= \{P_1\} \\
T &= \{T_1\} \\
Pre &= [Id] \\
Post &= [f] \\
M_0 &= [<U,r> \quad <L,l>]^T \\
C &= \{<U,r>,<U,l>,<L,r>,<L,l>\}
\end{aligned}
$$

#

Coloured Petri Nets version 1986

The difference between the second version of Coloured Petri Nets and the first one is the representation of the arc inscriptions. Coloured Petri Nets version 1981, relies on a function representation. The arc inscriptions are functions that are associated with the input and output arcs of a transition. Associated with a transition is a set of firing colors that indicate the firing possibilities of a transition. Coloured Petri Nets version 1986 relies on an expression representation where arc expressions are used in combination with guards. It has been shown that the two representations can be transformed into each other.

The token colour is a data value, that might be of arbitrarily complex type, e.g., a record where the first place is an integer and the second place a boolean variable. Each place in the net have a colour set that specifies the possible colours of tokens that might reside in this place.

The concept of guards is introduced. A guard is a boolean expressions restricting the conditions under which the transition can fire. The guard must be fulfilled before the transition can fire. Each arc has an associated arc expression that determines which and how many tokens that will be affected by a transition firing. A declaration is associated with each net specifying the different colour sets and variables.

Presentations of practical applications with Coloured Petri Nets can be found in [Jensen, 1997]. A formal definition of Coloured Petri Nets version 1986 can be found in [Jensen, 1995].

EXAMPLE 4.1.5
A example, inspired by [Jensen, 1995], illustrates how this type of net works. In Figure 4.5 there are two types of processes, called p and q. Three q-processes start in place $p1$ and cycle through the places $(p1, p2, p3)$. Two p-processes start in place $p2$ and cycle through the places $(p2, p3)$. Each of the five processes is represented by a token, where the token colour is a pair such that the first element tells whether the token represents a p-process or a q-process and the second element is an integer telling how many full cycles that process has completed. In the initial marking there are three $(q, 0)$-tokens at place $p1$ and two $(p, 0)$-tokens at place $p2$. There are two different types of resources, one r-resource and three s-resources.

In the arc expressions and in the guards different variables are used. The variables are specified in the declaration that is associated with

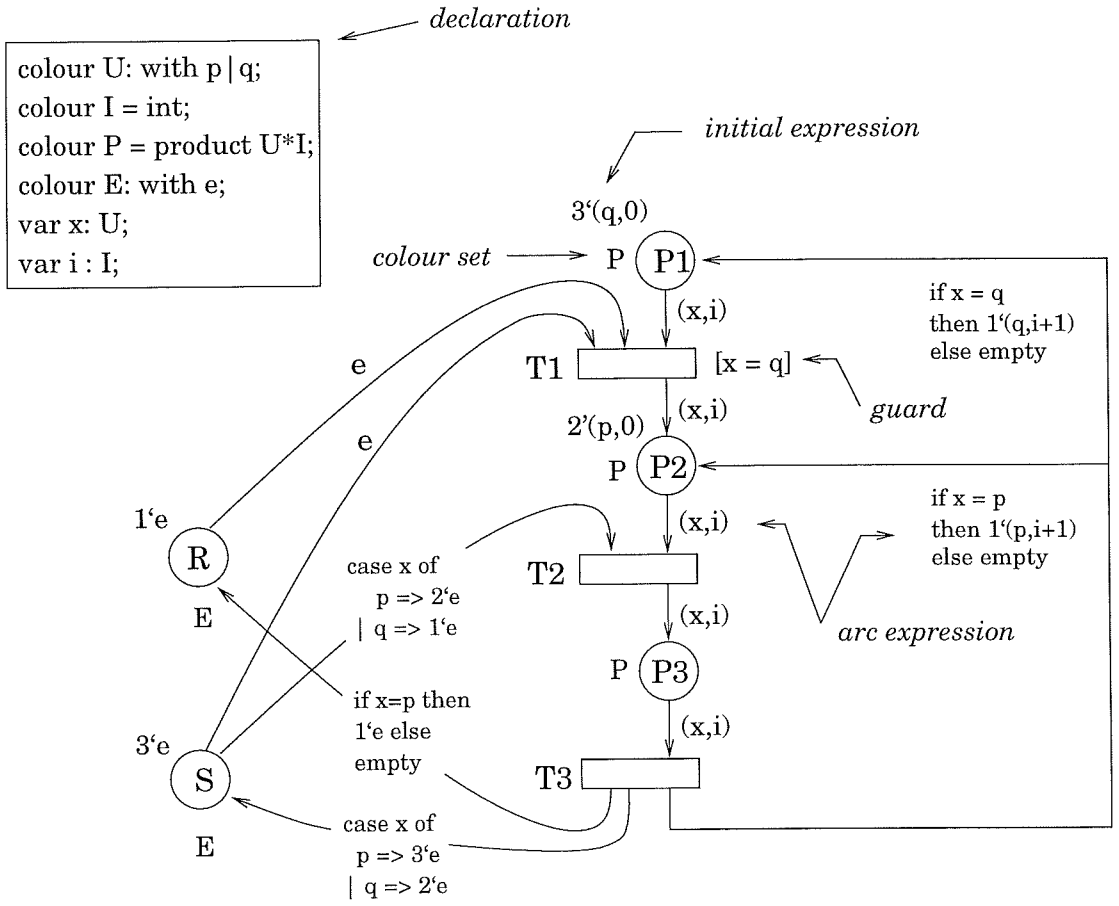


Figure 4.5 A coloured Petri Net describing a resource allocation system.

the net. The variable i is allowed to be of type I where I is a colour declared as an integer. The variable x is allowed to be of type U where U is a colour either equal to p or q. A guard might restrict the possible types of variables that can enable a transition. This is, e.g., the case for transition $T1$ where the variable x is only allowed to be of type q. The colour set, P, associated with the places, indicates the possible token colours allowed in a place. The colour P is composed of one part of type U and one part of type I. The resources are of token type E.

Transition $T1$ is enabled if there is a token of type (q, i) in place $P1$, a token of type e in place R and a token of type e in place S. When the transition fires the tokens that enabled the transition are removed and a token of type (q, i) is added in place $P2$. Transition $T3$ does not have a guard, this means that the transition can be enabled by a token of type (p, i) or a token of type (q, i) in place $P3$. If the transition is enabled with respect to a token of type (p, i), the firing of the transition will remove a token of type (p, i) from place $P3$ and add one token of type e in place R, three tokens of type e in place S and one token of type $(p, i + 1)$ in place $P2$. If the transition instead is enabled with respect to a token of type (q, i), the firing of the transition will remove a token of type (q, i) from place $P3$ and add two tokens of type e in place S and one token of type $(q, i + 1)$ in place $P1$.

#

A toolbox, called Design/CPN, has been developed for Coloured Petri Nets, [Met, 1993]. The user graphically draws the net and textually specifies the colour sets, arc expressions and guards. The inscription language of the toolbox is Standard ML, [Harper, 1986].

4.2 Coloured Grafcet

Coloured Grafcet is a graphical model similar to Grafcet. Coloured Grafcet was introduced by Agaoua in a PhD-thesis in 1987 [Agaoua, 1987]. It has also been used to realize a model for simulation and real-time control, [Suau, 1989].

Grafcet is used to control a system. In order to be able to control large systems composed of several identical, or similar, subsystems, the con-

cept of Coloured Grafcet is introduced, i.e., the main reason for introducing Coloured Grafcet is the same as that for the introduction of Coloured Petri Nets.

A Coloured Grafcet is built up by steps and transitions in the same way as Grafcet. Associated with the arcs are functions that specify the numbers and the colours of the tokens that should be removed or added to a step, the functions are called *Pre* and *Post*, respectively. A step may contain at most one token of each colour. The set of different token types is denoted C. All possible combinations of token types that may reside in a step is called C_S. Each action associated with a step is bound to a colour. When a token enters the step, the action, if any, associated with the step and with the colour of the token is effectuated. The action types are the same as those for Grafcet, see Chapter 3.1 (Actions). A transition can be enabled with respect to different colours. A transition can have several receptivities. Each receptivity is bound to a colour. The set of different receptivity types is called C_T and is a product of C and $\mathcal{R}$ where $\mathcal{R}$ is the set of conditions and events that enable a change of state. The colour-function is denoted C and is defined as $C = \{C_S \cup C_T\}$.

The new concepts in Coloured Grafcet, compared to Grafcet, are the introduction of colours and the colour-functions.

EXAMPLE 4.2.1
Consider the case were there are two machines, M_1 and M_2. Each machine can be in either of two states: idle or busy. If the machine is in the state idle and the button start is pushed the machine switches state and starts to work. If the machine is in state busy and the button stop is pushed the machine stops working and becomes idle. The two machines can be modeled by two separate Grafcets, see Figure 4.6 (left), or by one Coloured Grafcet, see Figure 4.6 (right).

#

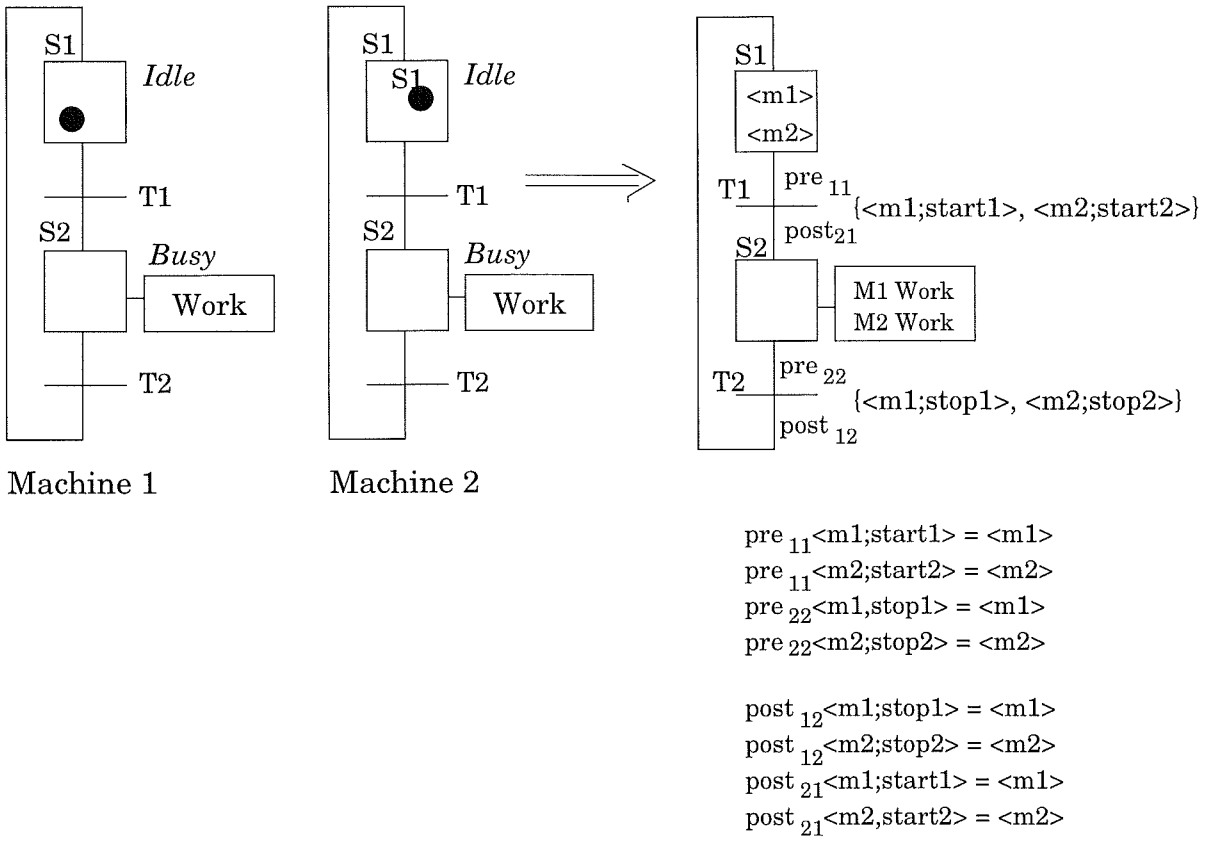


Figure 4.6 Two Grafcets transformed into a Coloured Grafcet.

Formal Definition

A Coloured Grafcet, [Agaoua, 1987], is a 6-tuple

$$CG = < S, T, C, Pre, Post, M_0 >$$

where:

- $S = \{S_1, S_2, ...S_n\}$ *is a finite, nonempty, set of steps.*
- $T = \{T_1, T_2, ..., T_m\}$ *is a finite, nonempty, set of transitions.*
- $S \cap T = \emptyset$, *i.e., the sets* S *and* T *are disjoint.*

- *C is the colour-function defined for $S \cup T$ in the non-empty set $\{CS_i \cup CT_i\}$ such that $CS_i \in C_S$ and $CT_j \in C_T$. The elements of $\{CS_i \cup CT_j\}$ are called colours.*
- *Pre is the input incidence function defined for $S \times T$ such that pre_{ij} is a function $CS_i \rightarrow CT_j$.*
- *$Post$ is the output incidence function defined for $S \times T$ such that $post_{ij}$ is a function $CT_j \rightarrow CS_i$.*
- *M_0 is the initial marking defined for S such that $\forall S_i \in S$, $M_0(S_i) \in C_S$.*

EXAMPLE 4.2.2
The Coloured Grafcet in Figure 4.6 (right) is described by the sextuple:

$$CG = < S, T, C, Pre, Post, M_0 >$$

where

$$\begin{aligned}
S &= \{S_1, S_2\} \\
T &= \{T_1, T_2\} \\
C &= \{C_S \cup C_T\} \\
C &= \{<m1>, <m2>\} \\
C_S &= \{<m1>, <m2>, <m1> + <m2>\} \\
\mathcal{R} &= \{start1, start2, stop1, stop2\} \\
C_T &= \{<m1; start1>, <m1; start2>, <m1; stop1>, <m1; stop2>, \\
&\quad <m2; start1>, <m2; start2>, <m2; stop1>, <m2; stop2>\} \\
Pre &= \begin{bmatrix} pre_{11} & 0 \\ 0 & pre_{22} \end{bmatrix} \\
Post &= \begin{bmatrix} 0 & post_{12} \\ post_{21} & 0 \end{bmatrix} \\
M_0 &= [<m1> + <m2> \quad 0]^T
\end{aligned}$$

#

4.3 Object Petri Nets and LOOPN

The difference between ordinary Petri Nets (PN) and Coloured Petri Nets (CPN) is the expressive comfort. A Coloured Petri Net can be transformed into an ordinary PN, but the structure of the CPN is more compact and most often easier to read and understand. CPN therefore constitute a significant advance over PN, but the absence of powerful structuring primitives is still a weakness, [Jensen, 1990].

Object Petri Nets are a class of Petri Nets where object-oriented ideas are applied to Coloured Petri Nets, [Lakos, 1994]. Two class hierarchies exists: one for tokens and one for subnets or modules. So far, the object Petri Nets have only focused on object-oriented structuring of the token types. The components of a class are drawn within a frame marked with the class name. A place is a data field which can supply values. All data fields are drawn as circles. Transitions and functions are drawn as rectangles. The export of fields and functions from a class is indicated by an undirected arc from the object to the class boundary. A class can be made a subclass of another class and thereby inheriting from it. The Object Petri Nets have retained the traditional Petri Net style with tokens as passive data items and their life cycles specified by the global control structure of the net. This means that, unlike ordinary object-oriented programming, it is not possible to let the call to a function, or method, depend on the type of data within the token and it is neither possible to affect the response of an operation. The Object Petri Nets can be proven to be behaviorally equivalent to Coloured Petri Nets and thus also to ordinary Petri Nets. This means that the analysis techniques developed for PN can be applied.

Object Petri Nets provides a simple way to model multi-level systems, i.e., systems where the components that move through the system have their own internal life cycles. Real-life situations generally have a number of layers of data activity, [Lakos, 1994]. For example, in modeling a traffic intersection, the cars which move through the intersection can be considered as data objects. The cars also have internal activities such as consumption of petrol, mechanical failure, etc, which may be of interest in the simulation.

EXAMPLE 4.3.1
A small example will illustrate how the OBJSA nets work. Figure 4.7 illustrates a truck that can move either to the left or to the right. The truck is driven by driver. Inside the truck there is food. As soon as the driver becomes hungry, he stops the truck and starts to eat, when he is no longer hungry he continues to drive.

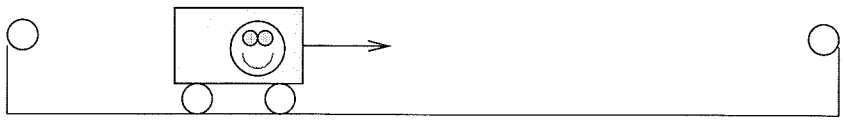


Figure 4.7 A truck and a driver.

The token type for a driver is called Driver and is shown in Figure 4.8 (left). The driver contains data e.g., an identity number given by an integer. The state of the driver is described by the net named Driver-Action shown in Figure 4.8 (middle). The net has two functions, Drive and Eat, that are exported. The two functions depend on the marking of place $P1$ and $P2$ respectively, as shown by a special arc, known as the compound arc. This arc has the effect of binding the variable y to the marking of $P1$ or $P2$ and then the function returns the boolean value $y = 1$. The token moving around in this net is of the type Driver.

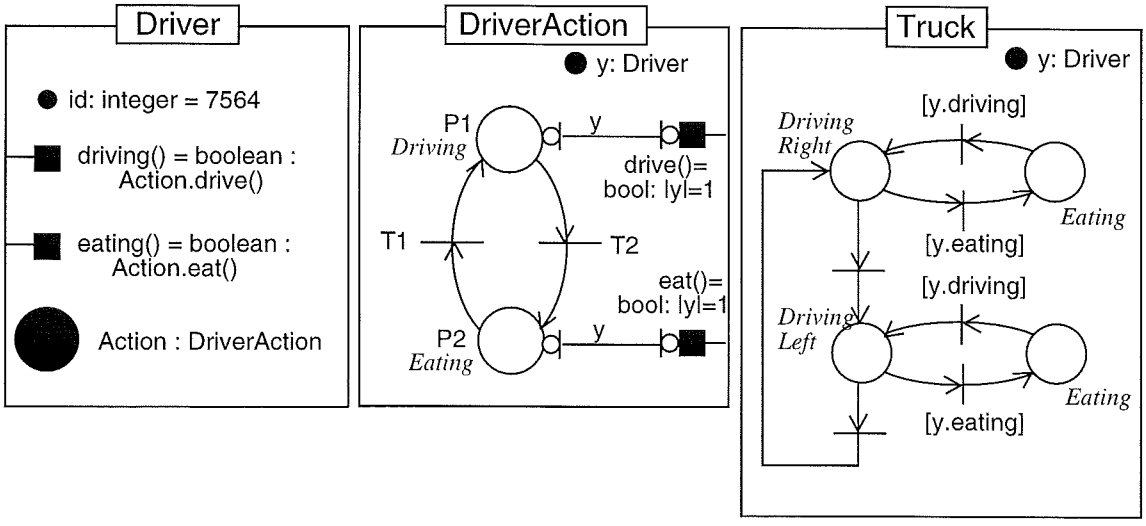


Figure 4.8 Class definitions for Driver, DriverAction and Truck.

The net describing the truck is shown in Figure 4.8 (right). The token moving around in this net is of type Driver. Associated with the tran-

sitions are receptivities from which it is possible to check the data of the token.

#

Modeling a multi-level system with CPN will often result in a big and complicated net whereas the modeling with OPN will be easier and the resulting structure of the net becomes easy to read and understand.

The formal definition of OPN as well as the relation to CPN are given in [Lakos, 1994]. LOOPN++, a textual language for Object Petri Nets is defined in [Lakos and Keen, 1994].

4.4 OBJSA Nets

High-Level Nets all have individual tokens. In the OBJSA Nets this is combined with algebraic specification techniques, [Battiston *et al.*, 1988]. OBJSA net systems is a class of high level Petri Nets. The OBJSA nets can be decomposed into state-machine components. The individual tokens are defined as abstract data types. To modify the data a language called OBJ2 or OBJ3 is used.

EXAMPLE 4.4.1
A small example will demonstrate the idea behind the OBJSA Nets. The example is a modified version of an example first published in [Battiston *et al.*, 1988].

Consider a system that consists of one sender (S) and one receiver (R). The receiver has a mailbox (M) consisting of a one-cell buffer where the sender asynchronously put messages. The receiver asynchronously reads the messages and removes them from the mailbox. The system can be modeled by an ordinary PN or by a coloured PN (CPN). In the CPN the tokens carry an identifier permitting the tokens to be distinguished from each other. The messages from a sender can be represented by the token $< msg >$ where msg is the message to be sent. The tokens representing the messages in the mailbox is either of the type empty $<->$ or of the type message $<msg>$.

However, usually mailboxes are not one-cell buffers but list of messages. This can conveniently be modeled using algebraic nets, i.e., nets

where the information associated with the token is algebraically specified and algebraically modified. Figure 4.9 shows the sender and the mailbox represented by an algebraic net. The token representing the mailbox is now a list, either empty $<eL>$ or nonempty $<L>$. When a new message comes the mailbox modifies its list of messages by placing the new message in the end of the old list. This operation is performed by the @-operator in the OBJ3 language. There exists a large number of operators defined in the OBJ3 language e.g., $tail(L)$, an operator that returns all elements in the list L but the first.

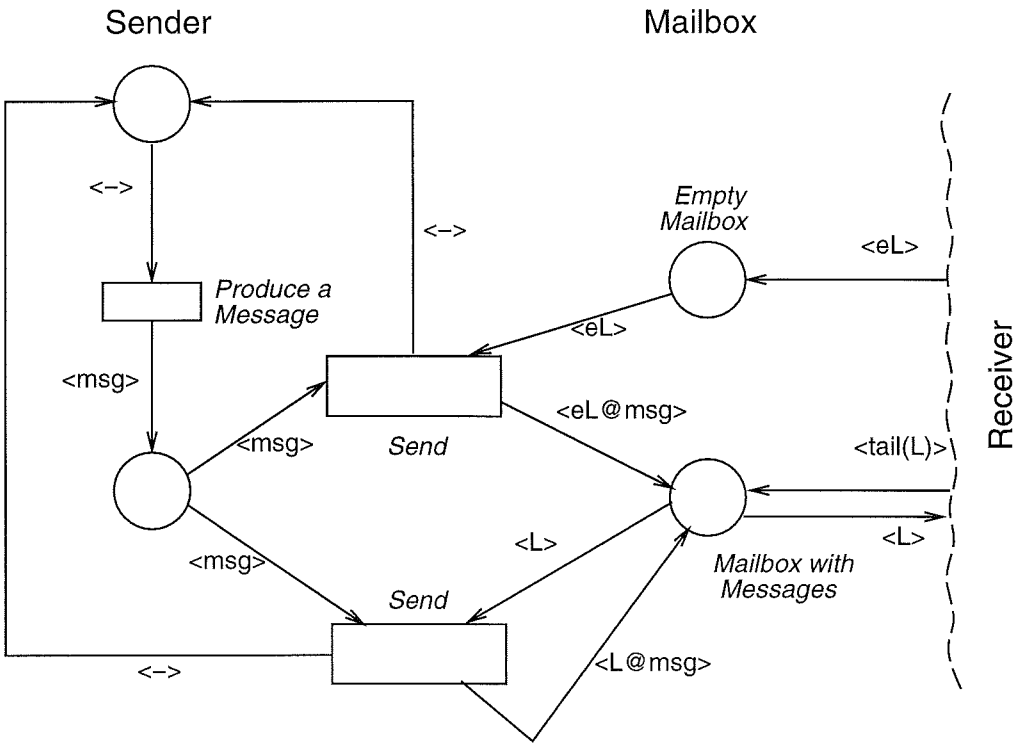


Figure 4.9 A sender and a receiver represented by an algebraic net.

#

A formal definition a more thoroughly presentation of the OBJSA Nets and the OBJ2 and OBJ3 languages can be found in [Battiston *et al.*, 1988].

4.5 Other High Level Languages influenced by Petri Nets

Petri Nets have also influenced some other high-level programming languages.

Cooperative Objects

Cooperative Objects is an object-oriented language influenced by the language Eiffel and by High-Level Petri Nets, [Bastide *et al.*, 1993]. Cooperative Objects aims at modeling a system at various abstraction levels as a collection of concurrent objects. The objects cooperate in a well defined way to provide the services expected from the system.

In this language concurrency, object behavior and inter-object communication are described in terms of High-Level Petri Nets. The language relies on a client-server organization of objects and it retains the most important features in object-oriented languages: classification, encapsulation, inheritance, instantiation and dynamic use relationship.

In Eiffel, the contract that the server object should fulfill for its client is expressed by a set of preconditions, postconditions and invariants. In Cooperative Objects, the contract is expressed by a High-Level Petri Net. Also the implementation of a class is given by a High-Level Petri Net as well as the semantics of a service invocation.

The language can be used both for specification of a concurrent class and for implementation. Results from the PN theory can be used to ensure the compatibility between a specification and its implementation.

Moby

MOBY (MOdellierung von BürosYstemen) is a tool which supports modeling and analysis of systems by hierarchical timed high level Petri nets with objects, [Fleischhack and Lichtblau, 1993]. The tool should assist in the modeling, analysis and simulation of processes in various application areas. The tool consists of editors for the specification of the net part, and of a simulator for the validation of constructed models. The simulator contains a conflict resolution component, which reduces the combinatorial complexity of possible behaviors of a model by using specific knowledge about the system, [Fleischhack and Lichtblau, 1993].

The nets considered in MOBY combines features from algebraic high level nets and coloured Petri Nets with concepts of time and hierarchy. The tokens are structured in an object oriented manner. A transition may be refined and the can be seen as a kind of macro expansion. From a transitions subnets can be called.

To each place in an object net one of the four types: multiset, stack, queue or priority queue, is assigned. This means that the objects in a place are handled according to the rules of the corresponding data type. Multiset places may contain tokens belonging to different types whereas other places are bound to one token type.

Arcs are either of standard type or of inhibitor type. In order to activate a transition, inhibitor arcs require that its input place does not contain certain objects. The arcs can be either activating or consuming. Activating arc behave like standard ones as far as the activation is concerned, however, the firing of the transition does not remove objects from the place. Consuming arcs enable a transition to empty a place.

GINA

Gina is an object-oriented language concept for parallel, hierarchically structured, data driven programs. It is also a Petri Net language based on dynamically modified, interpreted nets, [Sonnenschein, 1993].

4.6 Summary

Real-world systems are often very large and do often contain many parts which are similar but not identical. Using ordinary low level nets, these parts must be represented by disjoint subnets with nearly identical structures. This means that the total net becomes very large and it becomes difficult to see the similarities between the individual subnets representing similar parts. To make the representation more compact, efficient and manageable the different identical parts could be modeled by one net where each part is represented by an individual token. To be able to distinguish the different tokens, i.e., the different parts, an identifier is associated with each token.

5

Grafchart

Grafcet, or Sequential Function Charts (SFC), has been widely accepted in industry as a representation format for sequential control logic at the local level through the standards IEC 848 and IEC 1131-3, as described in Chapter 3. There is, however, also a need for a common representation format for the sequential elements at the supervisory control level. Supervisory control applications receive increasing attention both from the academic control community and the industry. The reasons for this are the increasing demands on performance, flexibility, and safety caused by increased quality awareness, environmental regulations and customer-driven production.

Sequential elements show up in two different situations in supervisory control. The first situation arises due to the fact that the processes in the process industry are typically of a combined continuous and sequential nature. All processes have different operation modes. In the simplest case these can consist of start-up, production and shut-down. The second situation concerns the case when the problem that the supervisory system should solve itself can be decomposed into sequential steps.

Grafcet was developed for logic controllers. It is a mathematical model whose semantics is determined by an interpretation algorithm. Since Grafcet was developed for sequential control logic at the local level it lacks many features needed to structure and implement the complex applications that are found on the supervisory control level.

Grafchart is the name of a mathematical sequential control model. It

is based on Grafcet but aimed, not only at local level applications, but also at supervisory level applications. Grafchart has been developed at Lund Institute of Technology, Sweden, since 1991, [Årzén, 1991], [Årzén, 1994b], [Årzén, 1993].

Grafchart is also the name of an implementation of the Grafchart model in G2, an object-oriented graphical programming environment, [Moore *et al.*, 1990], see Appendix A. By using Grafchart the same language can be used both on the local control level and on the supervisory control level, see Figure 5.1.

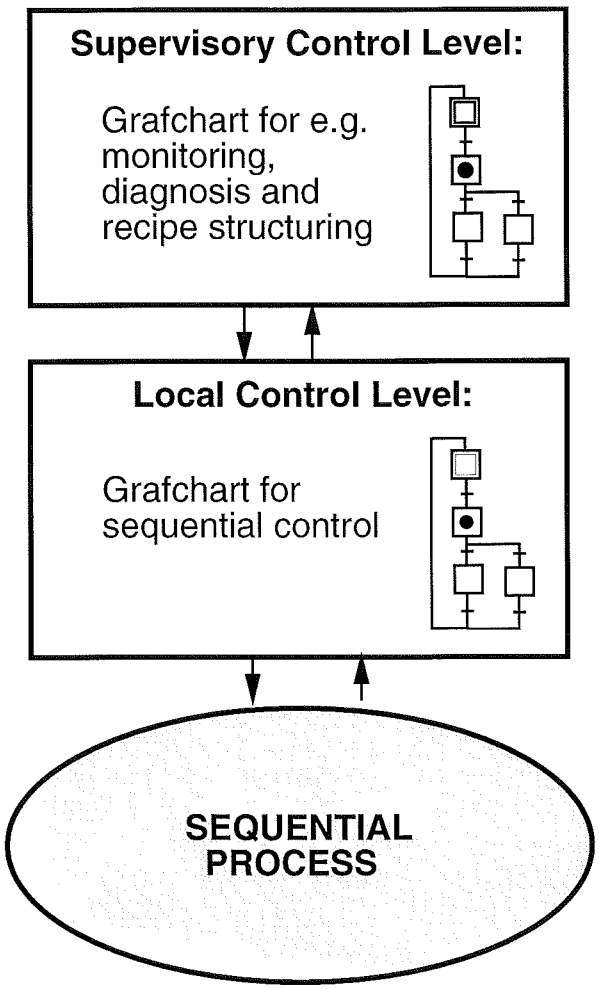


Figure 5.1 Supervision of sequential processes.

This chapter begins with a presentation of the Grafchart model and its graphical elements, followed by a description of the G2 implementation of Grafchart.

5.1 Graphical Language Elements

The graphical syntax of Grafchart is similar to that of Grafcet. It supports alternative branches and parallel branches. The graphical language elements of Grafchart are: steps, transitions, macro steps, procedure steps, process steps, Grafchart procedures and Grafchart processes. Each element is represented by an object. The icon of this object defines the graphical presentation of the language element. The language element are interconnected using graphical connections.

Grafchart processes

An entire function chart can be represented as a Grafchart process object, see Figure 5.2. The function chart is encapsulated by the Grafchart process object. In the G2 implementation the function chart is placed on the subworkspace, see Chapter 5.8, of the Grafchart process.

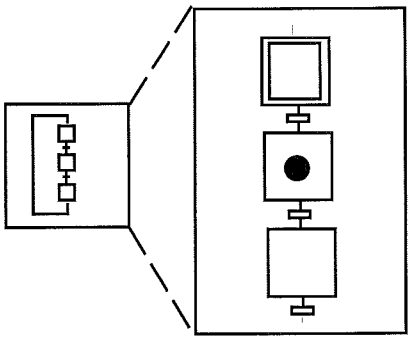


Figure 5.2 A Grafchart process.

A Grafchart can be closed or open, as shown in Figure 5.3. However, the function chart always starts with one or several initial steps.

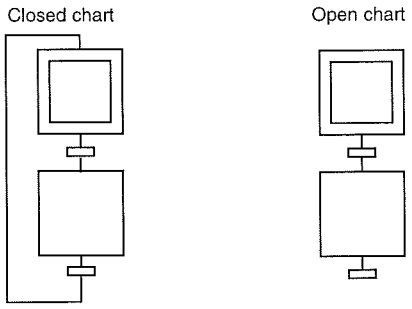


Figure 5.3 One closed and one open function chart.

Steps

A step is an object that has actions associated with it. A step is represented as a square, see Figure 5.4. An active step is indicated by a token placed in the step. A step is shown in Figure 5.4.

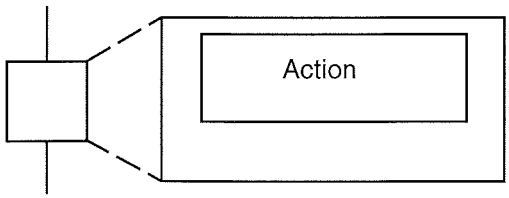


Figure 5.4 A step.

Initial step

The graphical representation of an initial step, i.e. a step that should be active when the function chart is started, is a double square, see Figure 5.5.

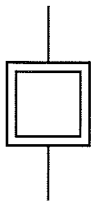


Figure 5.5 An initial step.

Transitions

A transition is represented by an object. The graphical representation of a transition is shown in Figure 5.6. Each transition has two attributes: `event` and `condition`. These are used for the receptivity of the transition.

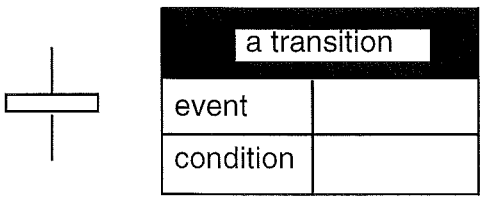


Figure 5.6 A transition.

Parallel bars

A parallel bar is used to indicate the beginning and the end of a parallel branch. To indicate the beginning of such a branch a parallel bar of type and-divergence (parallel split) is used and to indicate the end a parallel bar of type and-convergence (parallel join) is used, see Figure 5.7.

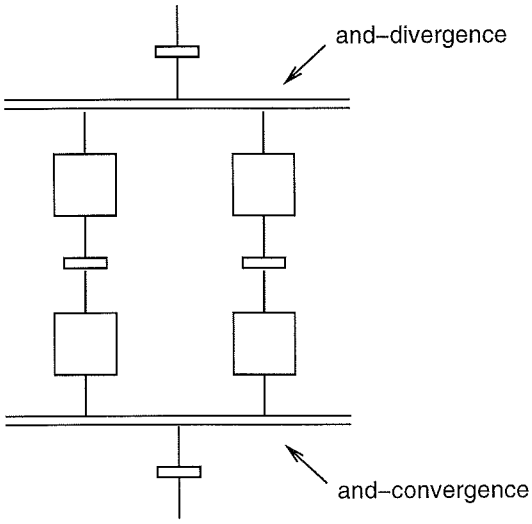


Figure 5.7 A parallel branch.

Macro steps

Macro steps are used to represent steps that have an internal structure of (sub)steps, transitions and macro steps. In the G2 implementation a macro step is represented by an object that has a subworkspace. The internal structure is placed on the subworkspace. In Figure 5.8, a macro step and its internal structure is shown.

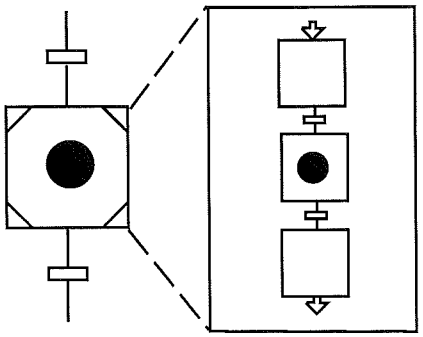


Figure 5.8 A macro step.

Enter steps and Exit steps

Special step objects, called enter-step and exit-step, are used to indicate the first and the last sub-step of a macro step. When the transition preceding a macro step becomes true, the enter-step upon the subworkspace of the macro step and all the transitions following that enter-step are activated. The transitions following a macro step will not become active until the execution of the macro step has reached its exit-step. An enter step and an exit step are shown in Figure 5.9.

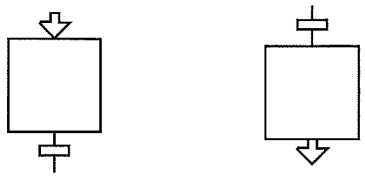


Figure 5.9 An enter step (left) and an exit step (right).

Grafchart procedures

Sequences that are executed in more than one place in a function chart can be represented as Grafchart procedures, see Figure 5.10. The Grafchart procedure object has a subworkspace on which the procedure body is placed. The enter-step and exit-step objects are used to indicate the first and the last sub-step of a procedure. Only one enter step is allowed in each Grafchart procedure.

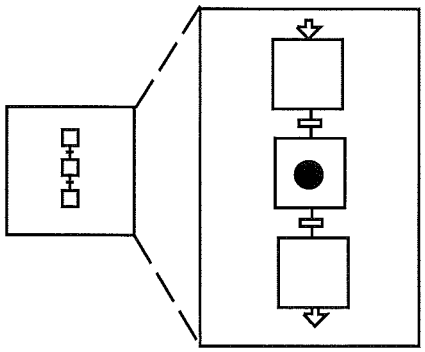


Figure 5.10 A Grafchart procedure and its subworkspace.

The Grafchart procedures are reentrant. Each procedure invocation executes in its own local copy of the procedure body. This makes recursive procedure calls possible.

Procedure steps

A Grafchart procedure is started from a procedure step. The procedure step has a `procedure` attribute in which the name of the Grafchart procedure that should be called is specified. Immediately when a procedure step is activated, the Grafchart procedure is called and a new token is placed in the enter step of the Grafchart procedure. The transitions following a procedure step do not become active until the execution of the Grafchart procedure has reached its exit step. In Figure 5.11 a procedure step is shown.

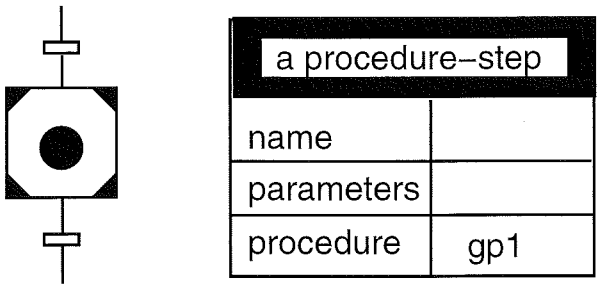


Figure 5.11 A Procedure step and its attribute.

Process steps

A procedure step is the equivalent of a procedure call in an ordinary programming language. Sometimes it is useful to start a procedure as a separate execution thread, i.e., to start the procedure as a separate process. This is possible with the process step, see Figure 5.12. The transitions after a process step become enabled as soon as the execution has started in the Grafchart procedure. An outlined circle token is shown in the process step as long as the process is executing. It is possible to have more than one process executing at the same time from the same process step.

Tokens

Tokens are represented as black filled circles. They act as boolean indicators indicating whether or not a step is active.

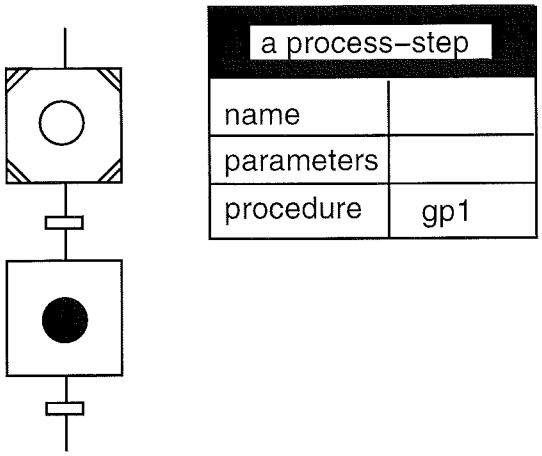


Figure 5.12 A Process step and its attribute.

5.2 Actions and Receptivities

Actions are associated with steps and receptivities are associated with transitions.

Actions

The main difference between Grafcet and Grafchart concerns the way actions are represented. Grafcet was developed as a logical controller. The actions that can be performed in a step are of a boolean or impulse nature. Two different kind of actions exist: level actions and impulse actions. The actions can be conditional or unconditional and it is possible to introduce a time-delay, see Chapter 3.1. In Grafchart, the actions, that can be associated with a step, are more general, they can be compared with the statements of a conventional programming language.

Four different types of actions exists: always, initially, finally and abortive. An initially action is executed once immediately when the step becomes active. This gives a behavior similar to the impulse action in Grafcet. A finally action is executed once immediately before the step is deactivated. An abortive action is executed once immediately before the step is aborted. An always action is executed periodically while the step is active. All actions can be conditional or unconditional. Which actions that may be performed in a step action depends on the underlying implementation language. The minimum requirement is

that it should be possible to assign values to variables and that is should be possible to execute macro actions, see Chapter 5.8.

Actions can also be associated to macro steps and Grafchart procedures. In this case, the initially actions are executed before the actions of the enter step of the macro step or the Grafchart procedure and the finally actions are executed after the actions of the exit step of the macro step or the Grafchart procedure. Always actions are executed periodically all the time while the macro step or Grafchart procedure is active. Abortive actions that are associated with a macro step, are executed if the execution of the macro step is aborted for some reason. Abortive actions can also be associated with a Grafchart procedure and they are executed if the Grafchart procedure is aborted.

All the actions types of Grafchart are of impulse nature. This is however, not a restriction compared with Grafcet, since all level actions in Grafcet can be transformed into corresponding impulse actions.

Receptivities

Each transition has two attributes, `condition` and `event`. These are used to enter the event expression and/or the logical condition telling when the transition should fire. The event expression and the logical condition are expressed in the underlying implementation language.

5.3 Error Handling

A common problem with Grafcet is how the logic for the normal operating sequence best should be separated from the error detection and error recovery logic. A separation is necessary, otherwise, the function chart will rapidly be very large, its clear structure will disappear and the function chart will be hard to read and understand.

Grafchart contains a number of assisting features for separation of error handling logic and normal operating logic.

Exception transitions

An exception transition is a special type of transition that only may be connected to macro steps and procedure steps. An ordinary transition connected after a macro step will not become enabled until the

execution has reached an exit step. An exception transition is enabled all the time that the macro step is active. If the exception transition condition becomes true or the exception transition event occurs, the exception transition will fire, abortive actions, if any, will be executed, and the step following the exception transition will become active. Exception transitions cannot be connected to a process step. The reason for this is that a Grafchart procedure can be started more than once from such a step and it is therefore not clear which ones of the procedures that should be aborted.

Macro steps and procedure steps remember their execution state from the time they were aborted and it is possible to resume them in that state. An exception transition is shown in Figure 5.13. Exception transitions do not exist in Grafcet. It was first proposed in [Årzén, 1991]. It has proved to be very useful when implementing error handling.

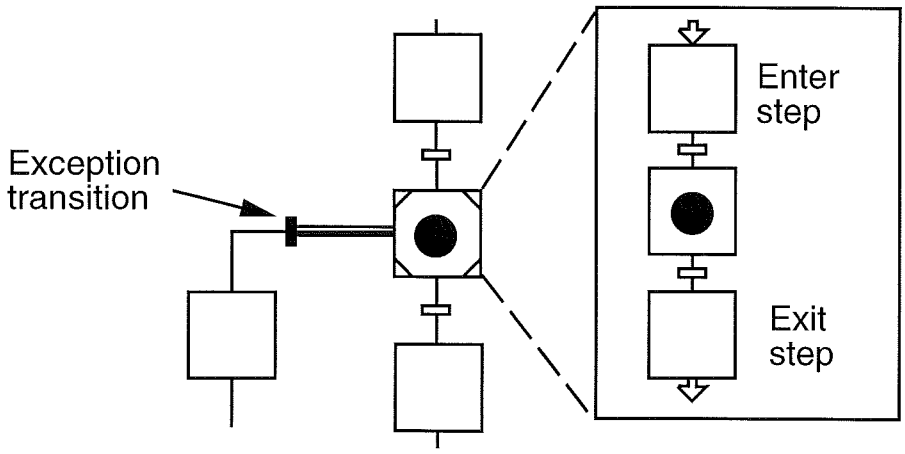


Figure 5.13 Macro step with exception transition.

Connection posts

A connection post is a special G2 item used to break a graphical connection, e.g., a connection between a step and a transition. This enhances the readability of the chart and can be used to separate error handling from normal operating logic. The use of connection posts is shown in Figure 5.14. The normal operating sequence has been separated from the error handling logic. Connection posts with the same name establish a logical connection.

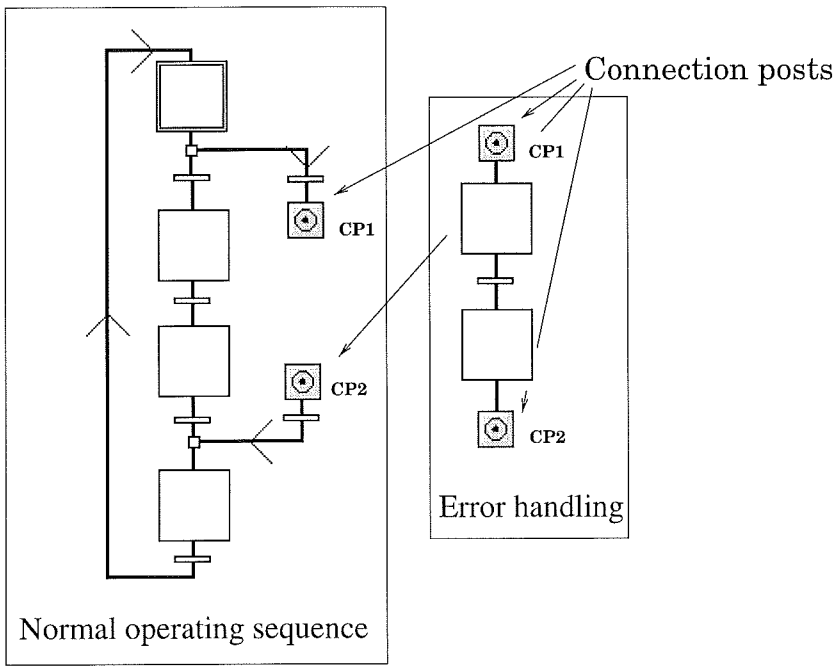


Figure 5.14 Two nets connected by connection posts.

Macro actions

Grafchart contains a number of macro actions. The macro actions affect the operation of an entire function chart. The macro actions can be called from step actions. There exist macro actions that cause a step to be deactivated, force a step to be active, force a transition to fire, abort an entire chart, freeze a transition or a whole chart, release a transition or a whole chart, and resume a macro step. Similar macro actions exist in Grafcet, [David and Alla, 1992]. The macro actions can be used to separate the error handling logic from the normal operation logic into separate Grafcharts without loosing the interactions between the two parts.

Step fusion sets

Grafchart also supports step fusion sets, [Jensen and Rozenberg, 1991]. Each step in a fusion set represents one view of the same conceptual step. Using fusion sets a step can have multiple graphical representations. When one of the steps in the fusion set becomes active (inactive) all the steps in the set will become activated (inactivated). Fusion sets can conveniently be used to separate normal operation logic from er-

ror handling logic. A small example of a step fusion set is shown in Figure 5.15.

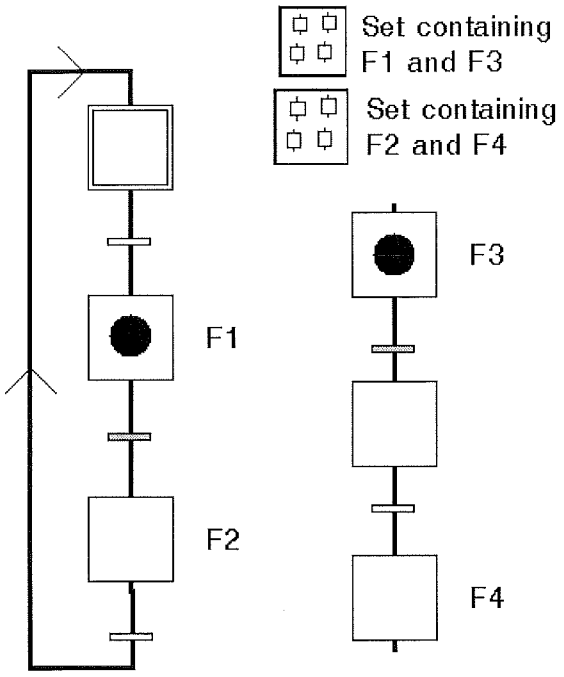


Figure 5.15 A step fusion set.

5.4 Dynamic Behavior

The dynamic behavior of Grafchart is given by the following rules.

Fireable transitions

.

A transition is fireable if and only if

- All the steps preceding the transition are `deactiveable`.
- The receptivity of the transition is true.

A step, initial step, enter step, exit step, and process step is deactiveable if and only if it is active.

A macro step is deactiveable if and only if an exit step upon the subworkspace of the macro step is active.

A procedure step is deactiveable if and only if the exit step in the Grafchart procedure started by the procedure step is active.

An exception transition is fireable if and only if

- The macro step or the procedure step preceding the exception transition is active.
- The receptivity of the exception transition is true.

A macro step is active if at least one of the steps upon the subworkspace of the macro step is active.

A procedure step is active if at least one of the step in the Grafchart procedure started by the procedure step is active.

Firing of a transition

The firing of a transition consists of deactivating the steps preceding the transition, de-enabling all transitions following the preceding steps, activating the steps succeeding the transition and enabling the transitions following the succeeding steps.

Firing rules

The firing rules that apply to Grafchart are similar to those of Grafcet.

- Rule 1: All fireable transitions are immediately fired.
  This rule applies both to Grafcet and to Grafchart.
- Rule 2: Several simultaneously fireable transitions are simultaneously fired.
  In Grafcet, this can cause problems in an or-divergence situation since the transitions in more than one branch can be simultaneously enabled. In this case all enabled transitions will fire and several different branches will be executed. This is often not the intention of the designer and it is therefore recommended that these transitions are made mutually exclusive. For Grafchart,

the rule applies in all situations except for the or-divergence situation where Grafchart treats the situation by nondeterministically choosing one of the branches. In order to have an interpretation algorithm without ambiguities, situations like this must be avoided. It is therefore required, also for Grafchart, that the user makes all branches of an or-divergence situation mutually exclusive.

- Rule 3: When a step must be simultaneously activated and deactivated, it remains active.
  This rule apply to Grafcet but not to Grafchart. For Grafcet, this means that the level actions associated with the step remain constantly active and the impulse actions are not carried out a second time. For a Grafchart, however, the finally actions and the initially actions will be executed.

5.5 Interpretation Algorithm

The interpretation algorithm of Grafchart describes the theoretical understanding of a Grafchart and guarantees that everyone faced with the same Grafchart understands it in the same way.

As in the case of Grafcet, multiple interpretation algorithms or semantics are possible for Grafchart. The following algorithm corresponds to the search for stability algorithm for Grafcet, see Chapter 3.2.

Algorithm

1. Initialization. Activate the initial steps and execute the associated initially actions. Go to Step 13.

2. Wait for a new external event. It might be necessary to start or stop the execution of the always actions associated with the active steps since the conditional part of the always action might change.

3. When a new external event occurs, determine the set of the fireable transitions T1. If T1 is empty go to Step 2.

4. Stop executing the always actions of the steps (and their activated substeps) preceding the transitions in T1. If T1 contains no exception transitions go to Step 10.

5. Determine the set T2, which is the set of exception transitions contained in the set T1, T2 $\subset$ T1. Let T1 be the set of all transitions contained in the set T1, except the exception transitions, i.e. T1:=T1\ T2.

6. Carry out the abortive actions associated with the steps (and their substeps) preceding the transitions of T2.

7. Fire the transitions in T2.

8. Carry out the initially actions of the steps (and their activated substeps) succeeding the transitions in T2.

9. Let T1 be the set of transitions contained in T1 and that are still enabled. If T1 is empty go to Step 13.

10. Carry out the finally actions of the steps (and their activated substeps) preceding the transitions in T1.

11. Fire the transitions in T1.

12. Carry out the initially actions of the steps (and their activated substeps) succeeding the transitions in T1.

13. Determine the set of fireable transitions and exception transitions T1. If T1 is non empty go to Step 5.

14. A stable situation is reached. Start executing the always actions associated with the active steps (and their activated substeps). Go to step 2.

5.6 Formal Definition

A Grafchart can be defined as an 9-tuple

$$G = < \mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}, M, \mathcal{P}_{procedure}, \mathcal{P}_{process}, \mathcal{GP}, I >$$

where:

- $\mathcal{V}_r$ is the set of variables given by: $\mathcal{V}_r = \mathcal{V}_{ext} \cup \mathcal{V}_{int}$.
  $\mathcal{V}_{ext}$ is the set of variables originating from the plant. $\mathcal{V}_{int}$ is the variables corresponding to the internal states of the Grafchart. $\mathcal{V}_r$ can be divided into three, not necessarily disjoint, subsets.

  $$\mathcal{V}_r = \{\mathcal{V}_{r\mathcal{X}} \cup \mathcal{V}_{r\mathcal{M}} \cup \mathcal{V}_{r\mathcal{GP}}\}$$

  - $\mathcal{V}_{r\mathcal{X}}$ are the variables associated with the transitions not included in a macro step or a Grafchart procedure.
  - $\mathcal{V}_{r\mathcal{M}}$ are the variables associated with the transitions included in a macro step.
  - $\mathcal{V}_{r\mathcal{GP}}$ are the variables associated with the transitions included in a Grafchart procedure.

- $\mathcal{V}_a$ is the set of variables issued to the plant. $\mathcal{V}_a$ can be divided into three, not necessarily disjoint, subsets.

  $$\mathcal{V}_a = \{\mathcal{V}_{a\mathcal{X}} \cup \mathcal{V}_{a\mathcal{M}} \cup \mathcal{V}_{a\mathcal{GP}}\}$$

  - $\mathcal{V}_{a\mathcal{X}}$ are the variables associated with the steps not included in a macro step or a Grafchart procedure.
  - $\mathcal{V}_{a\mathcal{M}}$ are the variables associated with the steps included in a macro step.
  - $\mathcal{V}_{a\mathcal{GP}}$ are the variables associated with the steps included in a Grafchart procedure.

- $\mathcal{X}$ is the set of steps, $\mathcal{X} = \{x_1, x_2, x_3, ...\}$.
  A step $x_i \in \mathcal{X}$ is defined by $\{action(x_i)\}$, where $action(x_1) \in \mathcal{V}_{a\mathcal{X}}$ defines the actions associated with the step x_i.

- $\mathcal{T}$ is the set of transitions, $\mathcal{T} = \{t_1, t_2, t_3, ...\}$.
  A transition $t \in \mathcal{T}$ is defined by the 3-tuple:

$$\{X_{PR}(t), X_{FO}(t), \varphi(t)\}$$

  - $X_{PR}(t)$ is the set of previous steps of t,
    $X_{PR}(t) \in \{\mathcal{X} \cup \mathcal{M} \cup \mathcal{P}_{procedure} \cup \mathcal{P}_{process}\}$
  - $X_{FO}(t)$ is the set of the following steps of t,
    $X_{FO}(t) \in \{\mathcal{X} \cup \mathcal{M} \cup \mathcal{P}_{procedure} \cup \mathcal{P}_{process}\}$
  - $\varphi(t)$ is the receptivity associated with t,
    $\varphi(t) \in \mathcal{V}_{r\mathcal{X}}$.

- $\mathcal{M}$ is the set of macro steps, $\mathcal{M} = \{M_1, M_2, \cdots\}$.
  - $\mathcal{V}_{r\mathcal{M}}$ is the set of variables associated with the transitions in $\mathcal{M}$
  - $\mathcal{V}_{a\mathcal{M}}$ is the set of variables associated with the macro steps and the steps in $\mathcal{M}$.

- $\mathcal{P}_{procedure}$ is the set of procedure steps, $P = \{P_1, P_2, \cdots\}$.
  The name of the procedure that should be called from the procedure step is given by $Procedure(P_i) \in \mathcal{GP}$.

- $P_{process}$ is the set of process steps, $P = \{P_1, P_2, \cdots\}$.
  The name of the procedure that should be called from the process step is given by $Process(P_i) \in \mathcal{GP}$.

- $\mathcal{GP}$ is the set of Grafchart Procedures, $\mathcal{GP} = \{GP_1, GP_2, \cdots\}$ that can be called from the Grafchart.
  - $\mathcal{V}_{r\mathcal{GP}}$ is the set of variables associated with the transitions in $\mathcal{GP}$
  - $\mathcal{V}_{a\mathcal{GP}}$ is the set of variables associated with the Grafchart procedure and the steps in $\mathcal{GP}$.

- I is the set of initial steps, $I \subseteq \mathcal{X}$.

A macro step M_i is defined as a 10-tuple

$$M_i = < \mathcal{V}_{ri}, \mathcal{V}_{ai}, \mathcal{X}_i, \mathcal{T}_i, \mathcal{M}_i, \mathcal{P}_{procedure,i}, \mathcal{P}_{process,i}, \mathcal{GP}_i, Enter_i, Exit_i >$$

where:

- $\mathcal{V}_{ri}, \mathcal{V}_{ai}, \mathcal{X}_i, \mathcal{T}_i$ are defined as $\mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}$, for a Grafchart.
- $\mathcal{M}_i$ is the finite set of macro steps, $\mathcal{M}_i = \{M_{i,1}, M_{i,2}, \cdots\}$. The macro steps are not allowed to be infinitely recursive. The actions associated with a macro step $M_{i,j} \in \mathcal{M}_i$ are given by the set $action(M_{i,j}) \in \mathcal{V}_{a\mathcal{M}_i}$.
  - $\mathcal{V}_{r\mathcal{M}_i} = \{\mathcal{V}_{ri1} \cup \mathcal{V}_{ri2} \cup \cdots\}$.
  - $\mathcal{V}_{a\mathcal{M}_i} = \{\mathcal{V}_{ai1} \cup \mathcal{V}_{ai2} \cup \cdots \cup action(M_{i1}) \cup action(M_{i2}) \cup \cdots\}$.
- $\mathcal{P}_{procedure,i}$ is the finite set of Procedure steps, $\mathcal{P}_{procedure} = \{P_1, P_2 \cdots\}$.
- $\mathcal{P}_{process,i}$ is the finite set of Process steps, $\mathcal{P}_{process} = \{P_1, P_2 \cdots\}$.
- $\mathcal{GP}_i$ are the set of Grafchart procedures that can be called from the set of M_i.
- $Enter_i$ is the enter step of the macro step, $Enter_i \subset \mathcal{X}_i$.
- $Exit_i$ is the exit step of the macro step, $Exit_i \subset \mathcal{X}_i$.

A Grafchart procedure, GP_i is defined by the 10-tuple

$$GP_i = < \mathcal{V}_{ri}, \mathcal{V}_{ai}, \mathcal{X}_i, \mathcal{T}_i, \mathcal{M}_i, \mathcal{P}_{procedure,i}, \mathcal{P}_{process,i}, \mathcal{GP}_i, Enter_i, Exit_i >$$

where:

- $\mathcal{V}_{ri}, \mathcal{V}_{ai}, \mathcal{X}_i, \mathcal{T}_i$ are defined as $\mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}$ for a Grafchart.
- $\mathcal{M}_i$ is the finite set of macro steps, $\mathcal{M}_i$ is defined as for a Grafchart.
- $\mathcal{P}_{procedure,i}$ is the finite set of Procedure steps, $\mathcal{P}_{procedure} = \{P_1, P_2 \cdots\}$.
- $\mathcal{P}_{process,i}$ is the finite set of Process steps, $\mathcal{P}_{process} = \{P_1, P_2 \cdots\}$.
- $\mathcal{GP}_i$ are the finite set of Grafchart procedures. $\mathcal{GP}_i = \{GP_{i,1}, GP_{i,2}, \cdots\}$. The calls to Grafchart procedures are not allowed to be infinitely recursive. The actions associated with a Grafchart procedure $GP_{i,j} \in \mathcal{GP}_i$ are given by the set $action(GP_{i,j}) \in \mathcal{V}_{a\mathcal{GP}_i}$.
  - $\mathcal{V}_{r\mathcal{GP}_i} = \{\mathcal{V}_{ri1} \cup \mathcal{V}_{ri2} \cup \cdots\}$.
  - $\mathcal{V}_{a\mathcal{GP}_i} = \{\mathcal{V}_{ai1} \cup \mathcal{V}_{ai2} \cup \cdots \cup action(GP_{i,1}) \cup action(GP_{i,2}) \cup \cdots\}$.
- $Enter_i$ is the enter step of the Grafchart procedure, $Enter_i \subset \mathcal{X}_i$.
- $Exit_i$ is the exit step of the Grafchart procedure, $Exit_i \subset \mathcal{X}_i$.

EXAMPLE 5.6.1
The Grafchart function chart in Figure 5.16 is given by the tuple

$$G = < \mathcal{V}_r, \mathcal{V}_a, \mathcal{X}, \mathcal{T}, M, \mathcal{P}_{procedure}, \mathcal{P}_{process}, \mathcal{GP}, I >$$

where:

$$\begin{aligned} \mathcal{V}_r &= \{\mathcal{V}_{rX} \cup \mathcal{V}_{r\mathcal{M}} \cup \mathcal{V}_{r\mathcal{GP}}\} \\ \mathcal{V}_{rX} &= \{a, b, c, \uparrow d\} \end{aligned}$$

$$\begin{aligned} \mathcal{V}_a &= \{\mathcal{V}_{aX} \cup \mathcal{V}_{a\mathcal{M}} \cup \mathcal{V}_{a\mathcal{GP}}\} \\ \mathcal{V}_{aX} &= \{Initially\ A,\ Finally\ C,\ Initially\ C\} \end{aligned}$$

$$\begin{aligned} \mathcal{X} &= \{x1, x2, x3, x4\} \\ & action(x2) = Initially\ A \qquad action(x3) = Finally\ C \\ & action(x4) = Initially\ C \end{aligned}$$

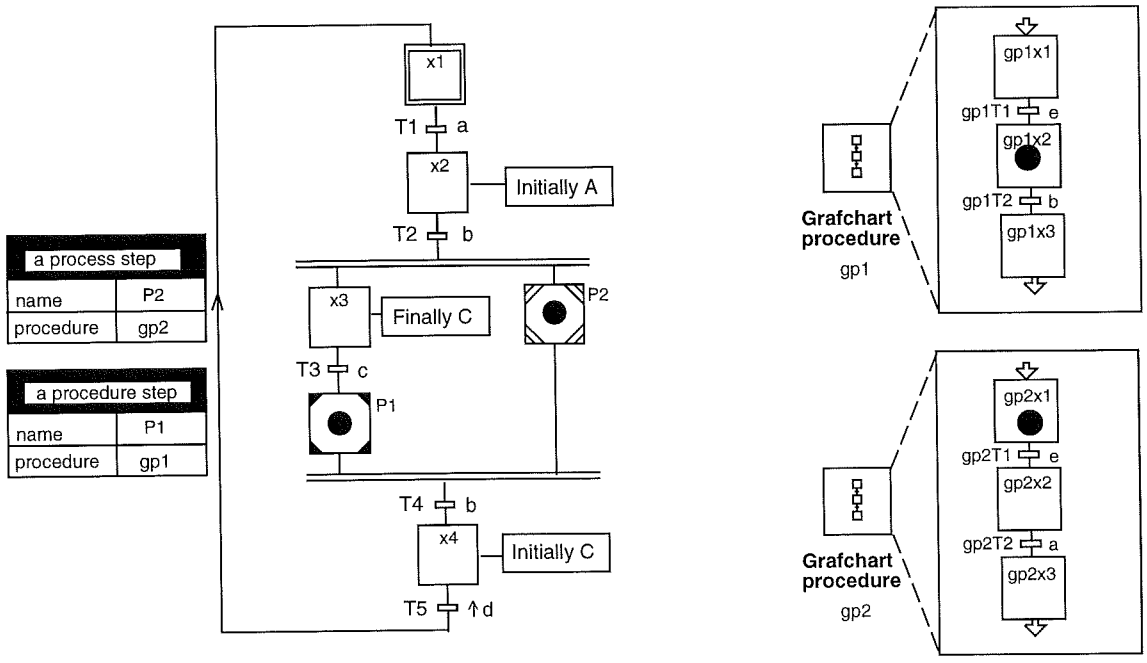


Figure 5.16 A Grafchart.

$$
\begin{aligned}
\mathcal{T} &= \{T1, T2, T3, T4, T5\} \\
&\quad X_{PR}(T1) = x1 \qquad X_{FO}(T1) = x2 \qquad \phi(T1) = a \\
&\quad X_{PR}(T2) = x2 \qquad X_{FO}(T2) = \{x3, P2\} \qquad \phi(T2) = b \\
&\quad X_{PR}(T3) = x3 \qquad X_{FO}(T2) = P1 \qquad \phi(T3) = c \\
&\quad X_{PR}(T4) = \{P1, P2\} \qquad X_{FO}(T4) = x4 \qquad \phi(T4) = b \\
&\quad X_{PR}(T5) = x4 \qquad X_{FO}(T5) = x1 \qquad \phi(T5) = \uparrow d \\
\mathcal{M} &= \varnothing
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{P}_{procedure} &= \{P1\} \\
&\quad Procedure(P1) = gp1
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{P}_{process} &= \{P2\} \\
&\quad Process(P2) = gp2
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{GP} &= \{gp1, gp2\} \\
&\quad \mathcal{V}_{r\mathcal{GP}} = \{\mathcal{V}_{r\ gp1} \cup \mathcal{V}_{r\ gp2}\} \\
&\quad \mathcal{V}_{a\mathcal{GP}} = \{\mathcal{V}_{a\ gp1} \cup \mathcal{V}_{a\ gp2}\ \}
\end{aligned}
$$

$$I = x1$$

The Grafchart Procedure $gp1$ in Figure 5.16 is given by the tuple

$$gp1 = < \mathcal{V}_{r\ gp1}, \mathcal{V}_{a\ gp1}, \mathcal{X}, \mathcal{T}, \mathcal{M}, \mathcal{P}_{procedure}, \mathcal{P}_{process}, \mathcal{GP}, I >$$

where:

$$
\begin{aligned}
\mathcal{V}_{r\ gp1} &= \{e, b\} \\
\mathcal{V}_{a\ gp1} &= \varnothing \\
\mathcal{X} &= \{gp1x1, gp1x2, gp1x3\}
\end{aligned}
$$

$$\mathcal{T} = \{gp1T1, gp1T2\}$$
$$X_{PR}(gp1T1) = gp1x1 \quad X_{FO}(gp1T1) = gp1x2 \quad \phi(gp1T1) = e$$
$$X_{PR}(gp1T2) = gp1x2 \quad X_{FO}(gp1T2) = gp1x3 \quad \phi(gp1T2) = b$$

$$\begin{aligned} \mathcal{M} &= \varnothing \\ \mathcal{P}_{procedure} &= \varnothing \\ \mathcal{P}_{process} &= \varnothing \\ \mathcal{GP} &= \varnothing \\ Enter &= gp1x1 \\ Exit &= gp1x3 \end{aligned}$$

The Grafchart Procedure $gp2$ in Figure 5.16 is given by the tuple

$$gp2 = < \mathcal{V}_{r\ gp2}, \mathcal{V}_{a\ gp2}, \mathcal{X}, \mathcal{T}, \mathcal{M}, \mathcal{P}_{procedure}, \mathcal{P}_{process}, \mathcal{GP}, I >$$

where:

$$\begin{aligned} \mathcal{V}_{r\ gp2} &= \{e, a\} \\ \mathcal{V}_{a\ gp2} &= \varnothing \\ \mathcal{X} &= \{gp2x1, gp2x2, gp2x3\} \\ \mathcal{T} &= \{gp2T1, gp2T2\} \end{aligned}$$
$$X_{PR}(gp2T1) = gp1x1 \quad X_{FO}(gp2T1) = gp1x2 \quad \phi(gp2T1) = e$$
$$X_{PR}(gp2T2) = gp1x2 \quad X_{FO}(gp2T2) = gp1x3 \quad \phi(gp2T2) = a$$

$$\begin{aligned} \mathcal{M} &= \varnothing \\ \mathcal{P}_{procedure} &= \varnothing \\ \mathcal{P}_{process} &= \varnothing \\ \mathcal{GP} &= \varnothing \\ Enter &= gp2x1 \\ Exit &= gp2x3 \end{aligned}$$

5.7 Grafchart vs Grafcet

The structure of a Grafchart can be transformed into that of a Grafcet. A Grafchart and a Grafcet are identical if, applied to the same input sequence, their output sequences are identical.

The transformation consists of two steps. First, the graphical elements must be changed to those of Grafcet. Secondly, the action types of Grafchart must be changed to those of Grafcet. Special care has to be taken to structures effected by the third firing rule.

Graphical element transformation

If the Grafchart contains only steps and transitions, the structure of the corresponding Grafcet will be identical to that of the Grafchart.

If the Grafchart contains macro steps, the macro step icon of Grafchart must be changed to that of Grafcet. The enter step and the exit step of the initial structure of the macro step must be replaced by an input step and output step of Grafcet. Alternatively, the macro step icon can be replaced by its internal structure in which the enter step and exit steps are replaced by ordinary steps, i.e., an in-line expansion is performed.

If the Grafchart contains procedure steps these should be replaced by a Grafcet macro step. The internal structure of the macro step should be that of the Grafchart procedure called by the procedure step.

If the Grafchart contains process steps, these should be replaced by a parallel construct (and-divergence) with two branches. One branch contains an empty step. The step is followed by the transition succeeding the process step. The second branch contains an empty step followed by an alternative structure (or-divergence) where each path contains a macro step followed by a sink transition. The macro steps are all identical and their internal structure is that of the Grafchart procedure called by the process step. The number of paths in the alternatives structure are equal to the number of processes that can run at the same time. This number has to be finite. The transformation of a process step is shown in Figure 5.17.

If the macro step or procedure step has an exception transition connected to it, the macro step icon or procedure step icon has to be re-

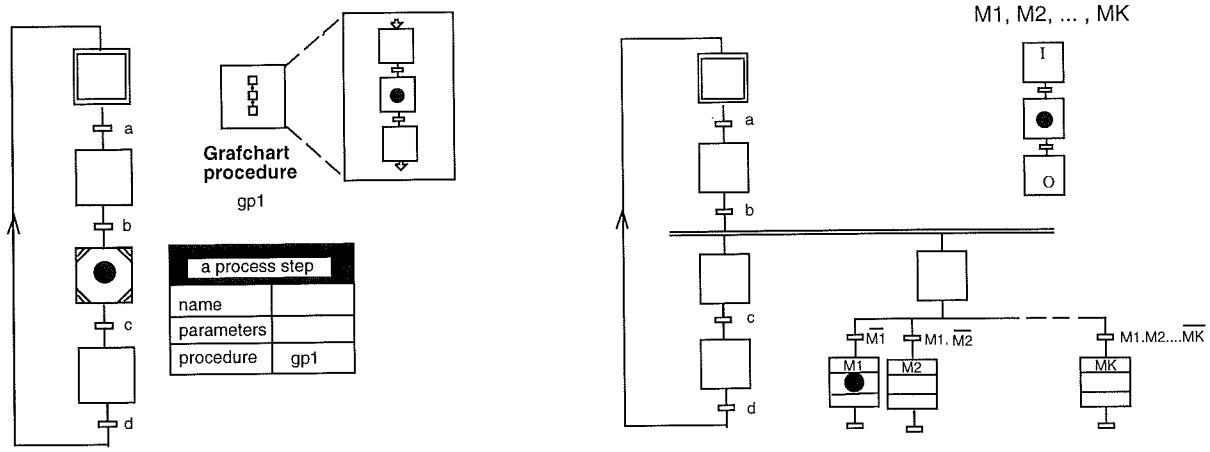


Figure 5.17 A Grafchart (left) and an identical Grafcet (right).

placed by its internal structure. To each of the internal steps an extra transition has to be added containing the events and conditions of the exception transition.

Action transformation

Each step in Grafchart, that contains actions can be replaced by a larger number of steps with actions in Grafcet, see Figure 5.18.

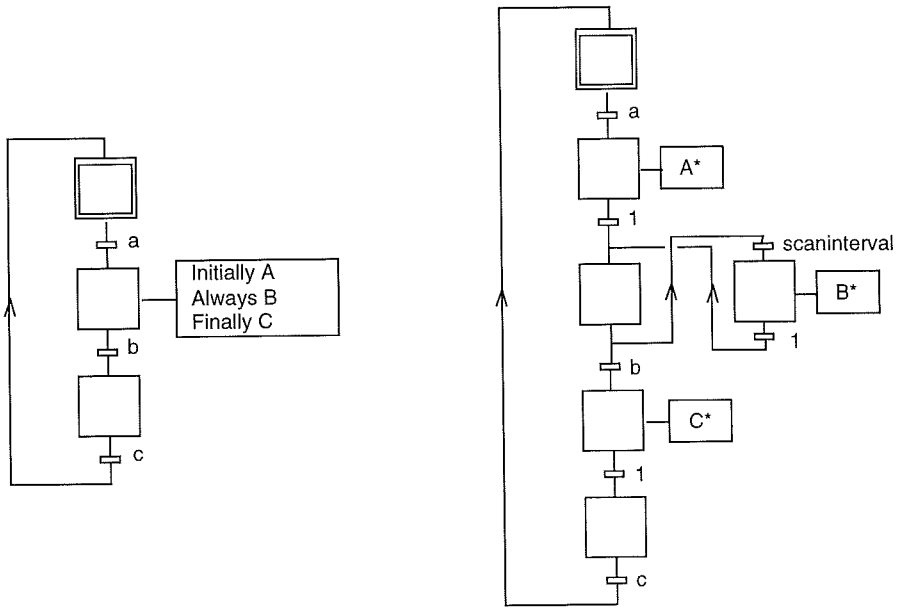


Figure 5.18 A Grafchart (left) and an identical Grafcet (right).

The transformation of a macro step with actions, or procedure step or process step for which the corresponding Grafchart procedure has actions, is possible but results in a huge and complex Grafcet.

Firing rules

The firing rules are not exactly identical in Grafchart and in Grafcet. A step that has to be simultaneously activated and deactivated remains active in Grafcet. This means that an impulse action associated with such a step in Grafcet will not be executed whereas an initially action associated with such a step in Grafchart will be executed. Some Grafchart structures therefore has to be extended with an extra step in the corresponding Grafcet, see Figure 5.19. The activation of the extra step assures that the step containing the impulse action is deactivated. A transition with a receptivity that is always true follows the extra step. When this transition fires the step with the impulse action is activated again and its impulse action is executed.

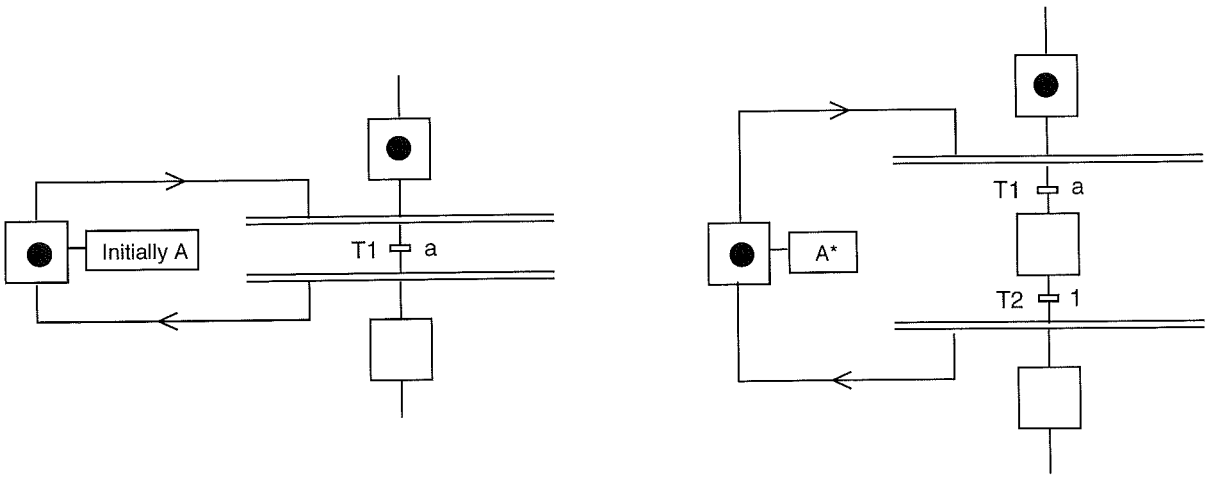


Figure 5.19 A Grafchart and an identical Grafcet.

5.8 The G2 Implementation

An implementation of Grafchart has been made in G2, an object oriented programming language, [Moore *et al.*, 1990], see Appendix A. The Grafchart objects are implemented as G2 objects and the connections are represented by G2 connections.

The implementation of Grafchart is based on the G2 concept of activatable subworkspaces. A workspace is visualized as a virtual, rectangular window upon which various G2 items such as rules, procedures, objects, displays, and interaction buttons can be placed. A workspace can also be attached to an object, see Figure 5.20. In this case the workspace is called a subworkspace of that object. When a subworkspace is deactivated, all the items on the workspace are inactive and invisible to the G2 run-time system.

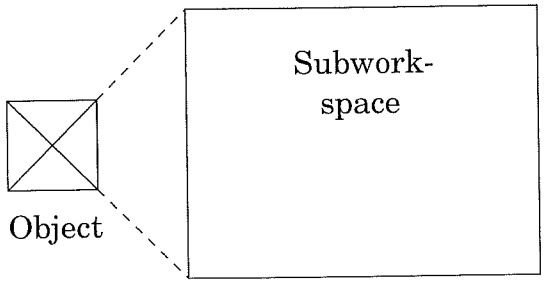


Figure 5.20 An object and its subworkspace.

Steps and actions

The actions associated with a step are placed on the subworkspace of the step. The actions are represented by G2 rules. The subworkspace of the step is only active when the step itself is active, this means that the rules, placed on the subworkspace, are only executed when the step is active. The actions that may be performed are the action types provided by G2. For example, it is possible to assign values to variables, to start procedures, to create and delete objects, hide and show information, perform animation actions, etc.

The actions are written in the G2's built in rule language. Sometimes, however, the syntax of G2 can be somewhat annoying. To facilitate for the user, the actions are therefore entered as action templates. These are text strings that during compilation are translated into the corresponding G2 rules. The entire syntax of G2 can be used in the action templates together with additional constructs of syntactical sugar nature. For example, the sometimes long and complicated syntax of certain G2 expressions can be replaced by a shorter and simpler expression using a Pascal-like dot notation, e.g. instead of referring to an attribute of an object using the standard syntax of G2 as `the`

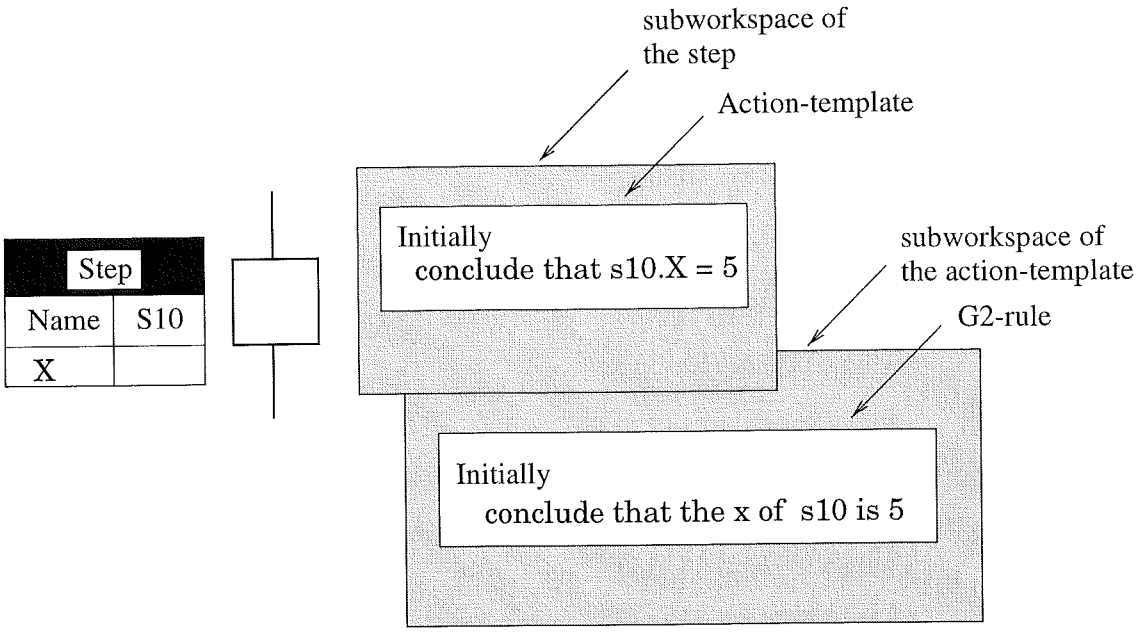


Figure 5.21 A step, an action template and the corresponding G2 rule.

`attribute1 of object1` the shorter notation `object1.attribute1` can be used. In Figure 5.21 a step and its subworkspace is shown. On the subworkspace an initially action template is placed. During compilation a subworkspace of the action template is created where the corresponding G2 rule is placed. In Figure 5.21, the subworkspace of the action template and the corresponding G2 rule is shown under the subworkspace of the step.

Transitions and receptivities

The condition and event of a transition is entered as attributes of the transition. During compilation the attributes are automatically translated into an appropriate G2 rule (compare step actions), which is placed on the subworkspace of the transition. The subworkspace is only active when the transition is enabled. A transition that only contains a logical condition is translated into a scanned `when` rule with the shortest scan interval possible. A transition that contains only an event expression and/or an event expression and a logical condition is translated into a `whenever` rule that will fire asynchronously when the event occurs.

When the transition fires, i.e., when the G2 rule condition becomes true

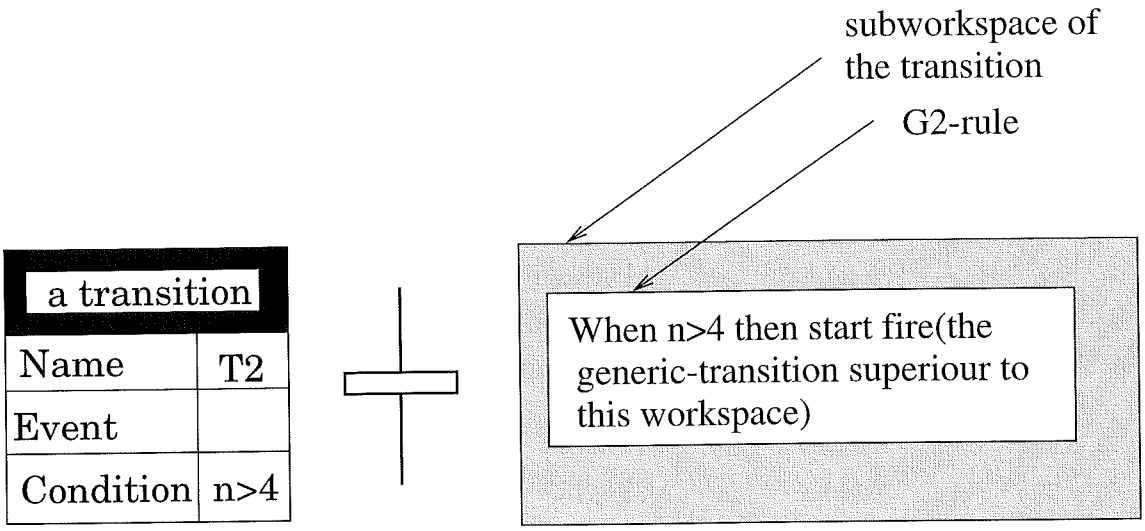


Figure 5.22 A transition, its attributes and the corresponding G2 rule.

and/or the event occurs, the G2 rule starts a procedure that takes care of the activation and deactivation of steps and transitions.

In Figure 5.22 a transition is shown together with its `event` and `condition` attributes. The corresponding G2 rule is placed on the subworkspace of the transition as shown in the same figure.

Class hierarchy

The objects in Grafchart are defined in classes. The relation between different classes is described by a class hierarchy. Attributes and methods can be associated with a class. A class defined above another class is called a superclass and a class defined below another class is called a subclass. G2 supports, like most object-oriented languages, inheritance and polymorphism.

The graphical language elements in Grafchart are defined in a class hierarchy, shown in Figure 5.23.

During compilation the action templates defining the actions of a step and the attributes defining the receptivity of a transition are translated into the corresponding G2 rules. Each class has a method, called compile, that describes how this class should be translated. The translation of a function chart is initialized by a call to one procedure called `compile`. This procedure traverses the function chart and calls the `compile`-method of each object in the function chart.

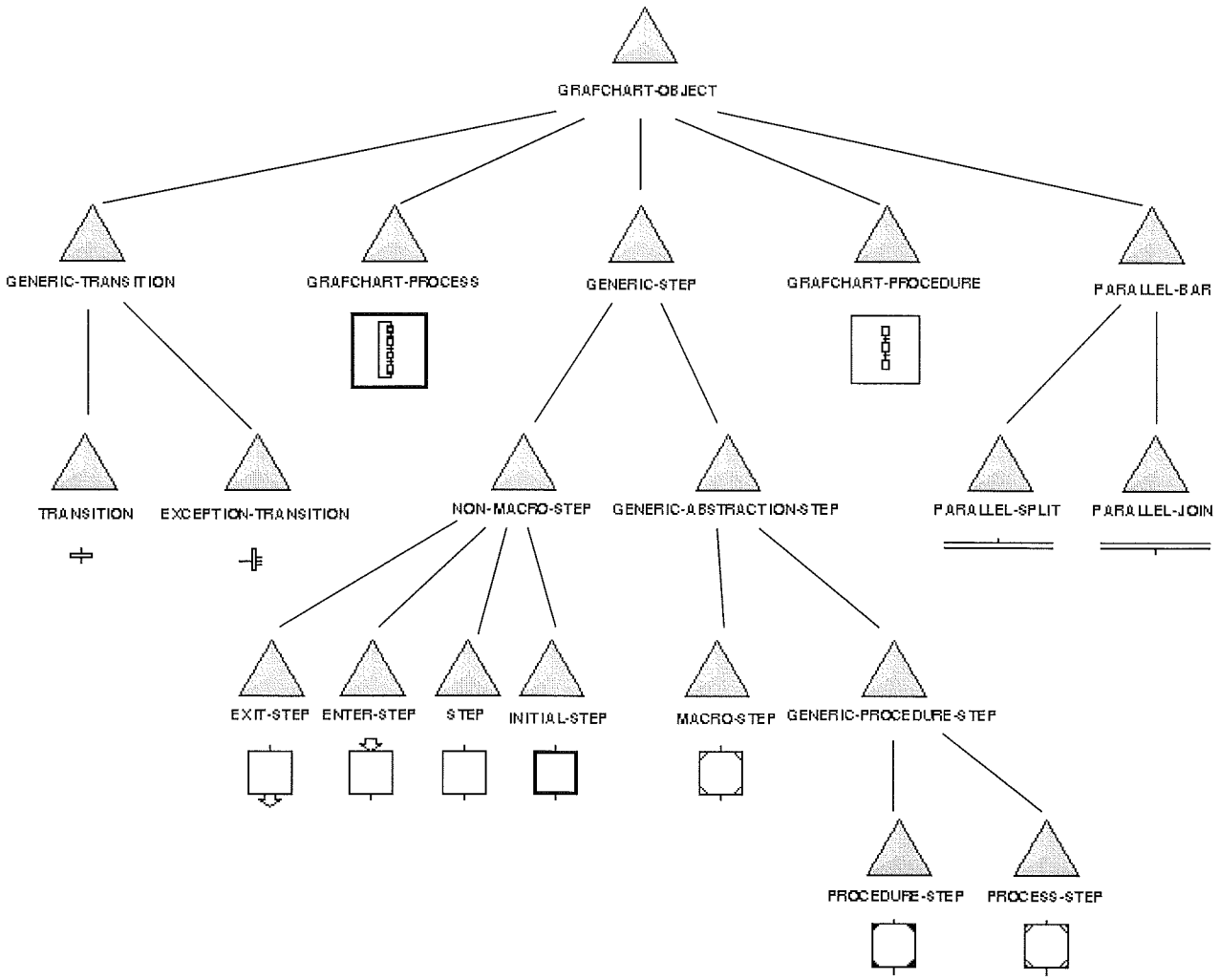


Figure 5.23 The class hierarchy in Grafchart.

Interpretation algorithm

The interpretation algorithm used in the implementation of Grafchart is based on the one given in Chapter 5.5. However, the algorithm given in Chapter 5.5 is a global algorithm whereas the one implemented in G2 is local. When a new external event occurs, the global algorithm searches through all receptivities to see which transitions that have become fireable. Since the implementation is based on activatable subworkspaces and G2 rules such a search is not performed. When a new event occurs or a condition becomes true the the G2 rule, that corresponds to a receptivity, immediately becomes true and a G2 procedure,

called `fire` is automatically started, see Figure 5.22. This procedure takes care of the deactivation and activation of steps and the enabling and de-enabling of transitions. The local algorithm that is used in the implementation is more efficient and less time demanding than the global algorithm. In almost all cases the behavior is equivalent to the behavior of the global algorithm.

EXAMPLE 5.8.1
Figure 5.24 shows a part of a function chart. The function chart has two steps, named $S1$ and $S2$, and two transitions, named $T1$ and $T2$. Step $S1$ is active and step $S2$ is inactive.

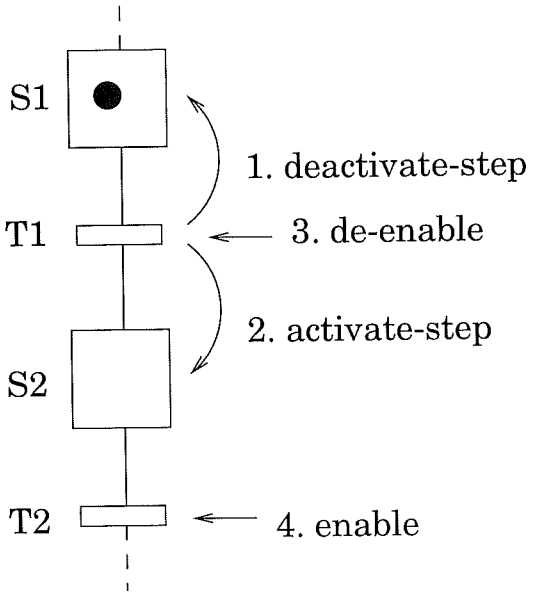


Figure 5.24 Firing of transition $T1$.

When the receptivity of transition $T1$ becomes true the procedure `fire` is started. The procedure calls, in the given order, the following methods: `deactivate-step`, `activate-step`, `de-enable` and `enable`. In Figure 5.24 this corresponds to deactivation of step $S1$, activation of step $S2$, de-enabling of step $T1$ and enabling of step $T2$.

#

Firing of a transition The deactivation and activation of steps as well as the de-enabling and enabling of transitions can be more com-

plicated if there are more than one input or output step or transition and/or if a parallel path or an alternative path is affected by the firing.

When a transition is fired, the following will happen (action execution not taken into account).

1. Deactivate-step
   (a) If the Grafchart object preceding the transition is a parallel-join the deactivation will be done for all Grafchart objects preceding the parallel-join.
   (b) If the Grafchart object preceding the transition is a macro step the deactivation will done for the macro step and the exit step of the macro step.
   (c) If the Grafchart object preceding the transition is a procedure step the deactivation will be done for the procedure step and for the exit step of the corresponding Grafchart procedure.
   (d) If the Grafchart object preceding the transition is a process-step, it will not be deactivated.
   (e) If the Grafchart object preceding the transition is a step, the step will be deactivated.

2. Activate-step
   (a) If the Grafchart object succeeding the transition is a parallel-split the activation will be done for all Grafchart objects succeeding the parallel-split.
   (b) If the Grafchart object succeeding the transition is a macro step the activation will be done for the macro step and for the enter step of the macro step.
   (c) If the Grafchart object succeeding the transition is a procedure step the activation will be done for the procedure step and for the enter step of the corresponding Grafchart procedure.
   (d) If the Grafchart object succeeding the transition is a process-step the activation will be done for the process step, if it is

not already active, and for the enter step of the corresponding Grafchart procedure.

(e) If the Grafchart object succeeding the transition is a step, the step will be activated.

3. De-enable

(a) The de-enable message is sent to the Grafchart object preceding the transition.

i. If, however, this Grafchart object is a parallel-join the de-enable message will be sent to each Grafchart object preceding the parallel-join.

(b) This object transmits the message to each succeeding Grafchart object.

i. If, however, this Grafchart object is a parallel-join the de-enable message will be sent to each Grafchart object succeeding the parallel-join.

(c) The Grafchart object is now a transition, this transition is de-enabled.

4. Enable

(a) The enable message is sent to the Grafchart object succeeding the transition.

i. If, however, this Grafchart object is a parallel-split the enable message will be sent to each Grafchart object succeeding the parallel-split.

(b) This object transmits the message to each succeeding Grafchart object.

i. If, however, the Grafchart object is a parallel-join the enable call will be sent to each Grafchart object succeeding the parallel-join.

(c) The Grafchart object is a transition, this transition is enabled.

i. If, however, this Grafchart object preceding the transition is a parallel-join the transition will only be enabled if all input places to the parallel-join are active.

EXAMPLE 5.8.2
Consider the function chart in Figure 5.25. Transition $T1$ and $T2$ are enabled. If transition $T1$ fires, the following will happen.

1. Deactivate $S1$.
2. Activate the parallel bar $PB1$, i.e activate the steps $S3$ and $S4$.
3. De-enable the transitions $T1$ and $T2$.
4. Enable the transitions $T3$ and $T4$. Transition $T5$ will not be enabled since $S5$ is not active.

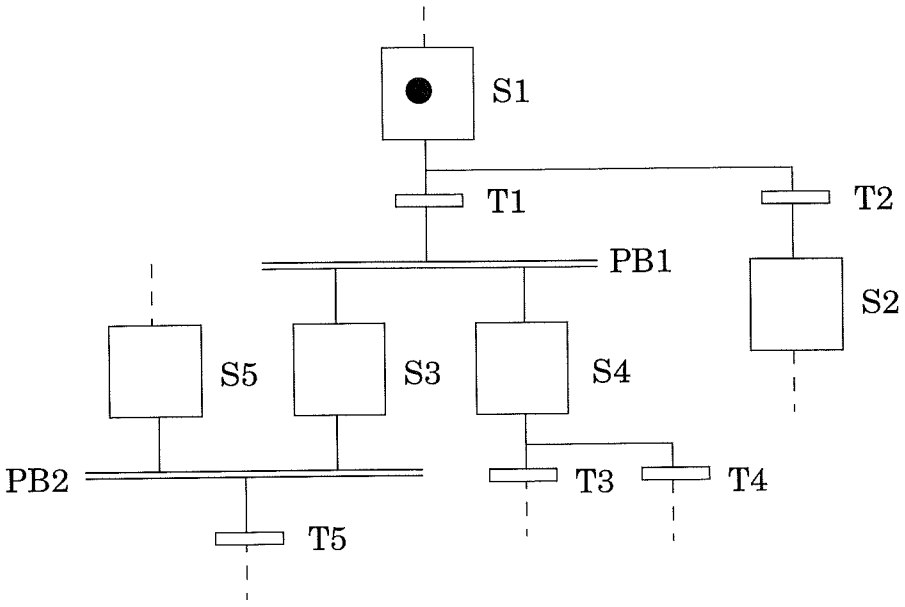


Figure 5.25 The firing of a transition.

5.9 Applications

Grafchart has been used in some applications.

Training simulator

Grafchart has been used to implement a prototype of a training simulator for a sugar crystallization process, [Nilsson, 1991]. In this application Grafchart is used both to structure the simulation model of the process and to implement the control system of the process.

Hydrogen balance advisory control

Grafchart has been used to implement a knowledge-based system, (KBS), that generates on-line advice for operators regarding the distribution of hydrogen resources at the Star Enterprise Delaware City Refinery [Årzén, 1994a]. The system uses KBS techniques coupled with numerical optimization. The specific problem that is solved is to meet the needs of the hydrogen consuming units in the refinery while minimizing the hydrogen that is wasted. A catalytic reformer unit and a continuous catalytic reformer unit produce hydrogen as by-products. A hydro cracker unit consumes high purity hydrogen and vents low purity hydrogen. Hydrogen from these units is used to satisfy the needs of the hydrogen consuming hydro treaters, sulphur recovery, methanol, and naphtalene units. Any additional hydrogen needs must be met by a hydrogen production unit.

Flexible manufacturing cell

In [López González *et al.*, 1994] a system is described were Grafchart is used to implement a flexible manufacturing cell. Grafchart is used in a four-layered hierarchical structure to represent the plant-wide operating phases of the control system, to describe the sequences of tasks to be executed to manufacture the parts, to describe the tasks at the workstation level and, finally, to describe the different services offered by the device drivers in the cell.

5.10 Summary

Grafchart is the name of a mathematical sequential control model and a toolbox. Grafchart is based on a syntax similar to that of Grafcet. The graphical elements are: steps, transitions, macro steps, procedure steps, process steps, Grafchart procedures and Grafchart processes. Grafchart is aimed at sequential control application at local and supervisory level.

6

High-Level Grafchart

High-Level Grafchart (HL-Grafchart) is an extended version of Grafchart, [Johnsson and Årzén, 1996b], [Årzén, 1996b], [Årzén, 1994a]. It combines the graphical language of Grafcet/SFC with high-level programming language constructs and ideas from High-Level Petri Nets. This widely increases the expression power and structuring facilities of Grafchart.

High-Level Grafchart adds four new features to Grafchart;

1. Parameterization
2. Methods and message passing
3. Object tokens
4. Multi-dimensional charts

High-Level Grafchart is available in two different versions. The first, called HL-Grafchart I, uses only the first two of the four new features, i.e., HL-Grafchart I is closer in spirit to Grafchart. The second version, HL-Grafchart II, uses all four features and is closer in nature to High-Level Petri Nets with the graphical syntax of Grafcet.

Grafchart is the name both of a formal model and of a G2 implementation of this model. The same situation applies to HL-Grafchart. In this chapter the focus will be the G2 implementation of HL-Grafchart.

6.1 Parameterization

Parameterization denotes the possibility for an object to have parameters. In HL-Grafchart, Grafchart processes, Grafchart procedures, macro steps and steps can be parameterized.

Ordinary Grafchart has no means for parameterization. This means that, e.g., the rules within a step have to be specific, i.e., they have to contain references to global variables and objects. This makes it difficult to reuse steps from one application to another. In High-Level Grafchart this is resolved by utilizing the fact that the steps are objects defined by a class definition. The user can specialize the step class by making a subclass in which additional attributes can been added. These attributes act as parameters which can be referenced from the actions within the instances of the step subclass.

The same technique is used for Grafchart processes, Grafchart procedures and macro steps. These object classes can be specialized by adding attributes which act as parameters. The parameters of a Grafchart process, Grafchart procedure or a macro step can be referenced from within all steps and all transitions placed on the subworkspace of the object. To reference a parameter, a special Pascal-like dot notation is introduced.

- `sup.attribute1`
  refers to the `attribute1` attribute visible in the current context.
- `sup.attribute1^`
  refers to the object named by the `attribute1` attribute visible in the current context.
- `sup.attribute1^.attribute2`
  refers to the `attribute2` attribute of the object given by the `attribute1` attribute visible in the current context.

Consider the example shown in Figure 6.1. The class named `fill-tank` is a specialization of a macro step with two new attributes: `tank` and `limit`. `FT1` is an instance of `fill-tank`. A fill-tank macro step contains the logic for the control and monitoring of the filling of a tank. The

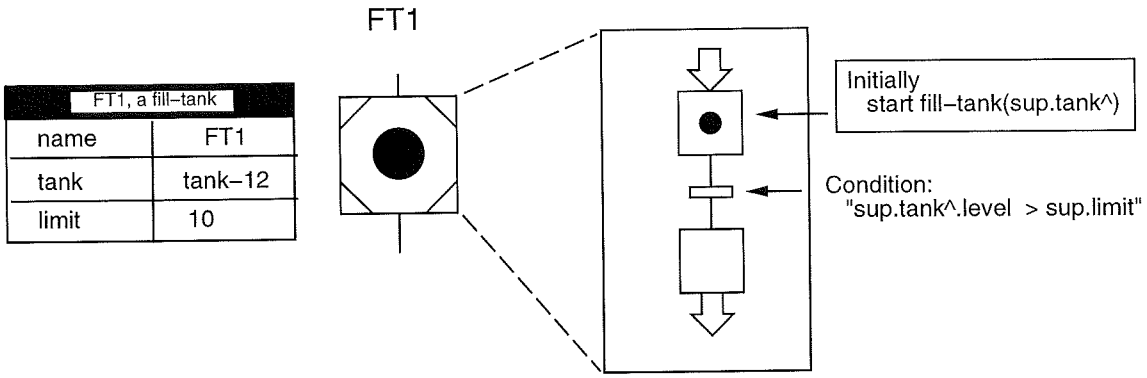


Figure 6.1 Parameterization of a macro step.

`tank` attribute contains the name of the tank that should be filled, i.e., the value of this attribute acts as a pointer. The `limit` attribute contains the limit up to which the tank should be filled. The macro step contains an enter-step that contains a rule that initiates the filling. This rule refers to the value of the tank attribute, i.e., tank-12, using the notation `sup.tank^` (sup is short for superior). Similarly the transition condition refers to the level of the tank referenced by the tank attribute and to the value of the limit attribute. The `sup.attribute` notation is translated and replaced by a corresponding G2 expression during compilation as described in Chapter 5.8.

Lexical scoping is used when searching for the attribute of an object replacing the `sup` notation. Consider the example shown in Figure 6.2. The macro step M1 has an attribute named x with value 156. The substructure of the macro step contains among other a transition and another macro step, M2. The macro step M2 has an attribute named x with value 12. The substructure of M2 contains a transition that reference to an x-attribute using the `sup.X` notation. The reference will be to the x-attribute of the macro step M2. The transition placed on the subworkspace of the macro step M1 also refers to an x-attribute using the `sup.X` notation. In this case the reference will be to the x-attribute of M1, see Figure 6.2. If, however, the macro step M2 would not have had an attribute named x, the `sup.X` reference of the transition of M2 would have refered to the x attribute of M1 as well.

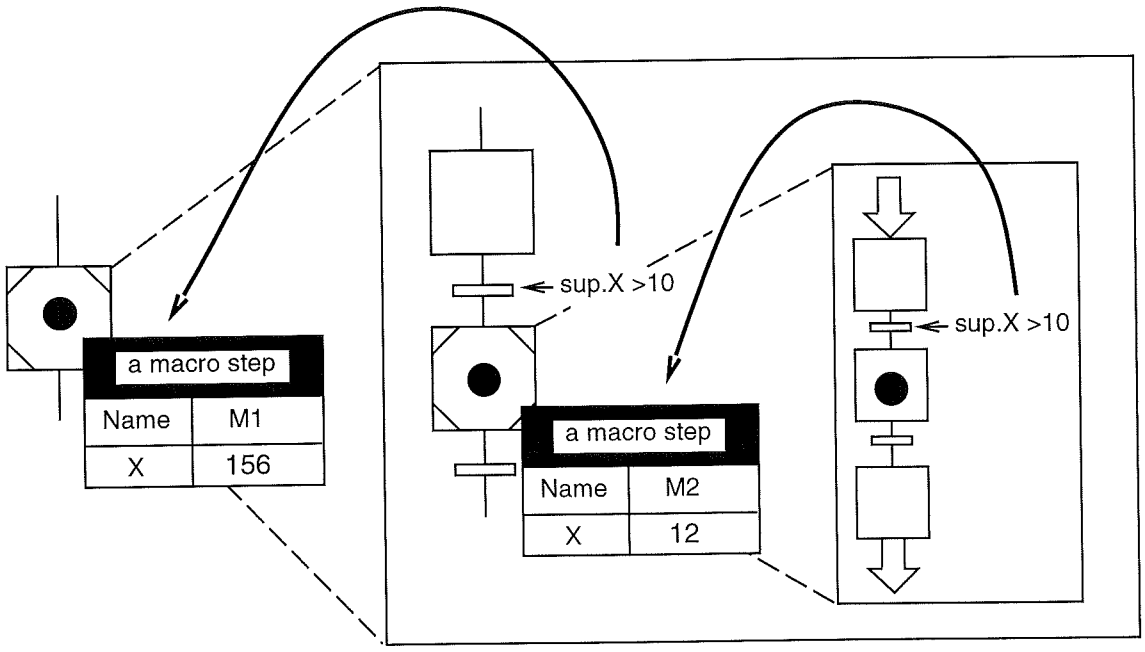


Figure 6.2 Lexical scoping.

Grafchart Procedure Parameterization

A Grafchart procedure is started from a process step or a procedure step. The actual values of the parameters of a Grafchart procedure are set in the procedure step or the process step. The procedure or process step has an attribute named `procedure` in which the name of the Grafchart procedure to call is given. They also have an attribute named `parameters` which contains a list of assignments to the formal parameters of the Grafchart procedures being called.

The assigned values can either be constants (numbers, strings or symbols), as in Figure 6.3 or the value of a parameter that is visible in the process or procedure step context, as in Figure 6.4. Using the latter form, it is possible for a Grafchart procedure to return values to the process or procedure step. In order to specify the direction in which the parameter is passed, one of the keywords `IN`, `OUT` or `INOUT` is added after the value of the parameter. It is also possible to determine which Grafchart procedure that should be called from the value of a parameter. In Figure 6.4 the parameters `sup.val`, `sup.v` and `sup.q` are all visible in the context of the parameter step. The parameter `proc` is also a parameter visible in the context of the procedure step, this param-

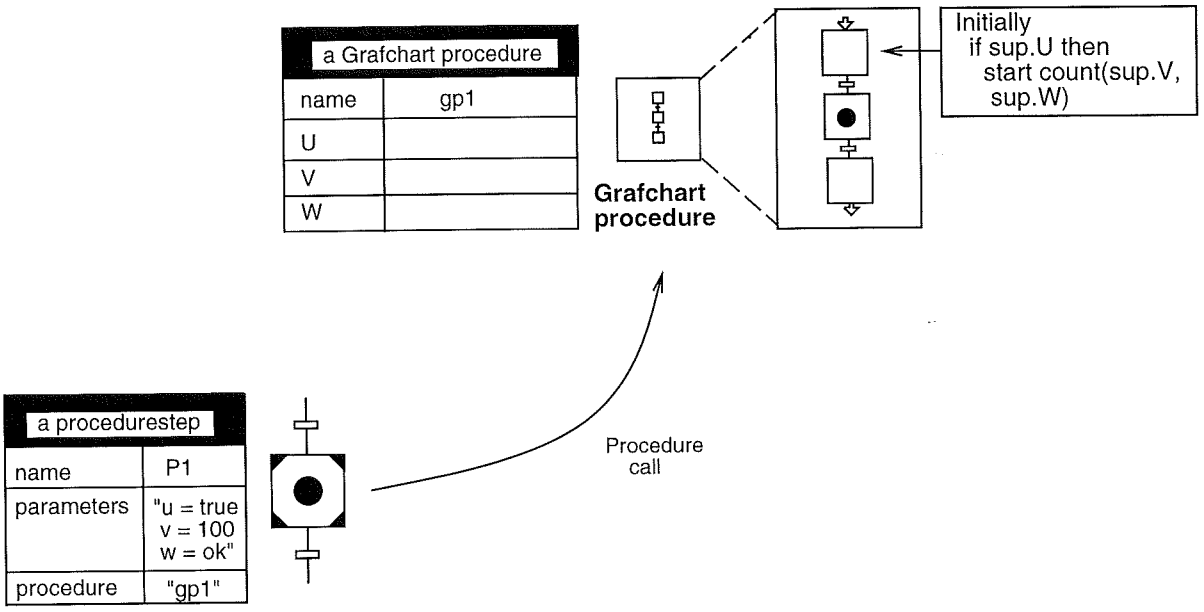


Figure 6.3 Parameterization of a Grafchart procedure.

eter determines the Grafchart procedure that should be called. When the Grafchart procedure is called `U` is assigned the value of `sup.val` and `W` is assigned the value of `sup.q`. When the execution of the Grafchart procedure ends, `sup.v` is assigned the value of `V`, and `sup.q`, is assigned the value of W.

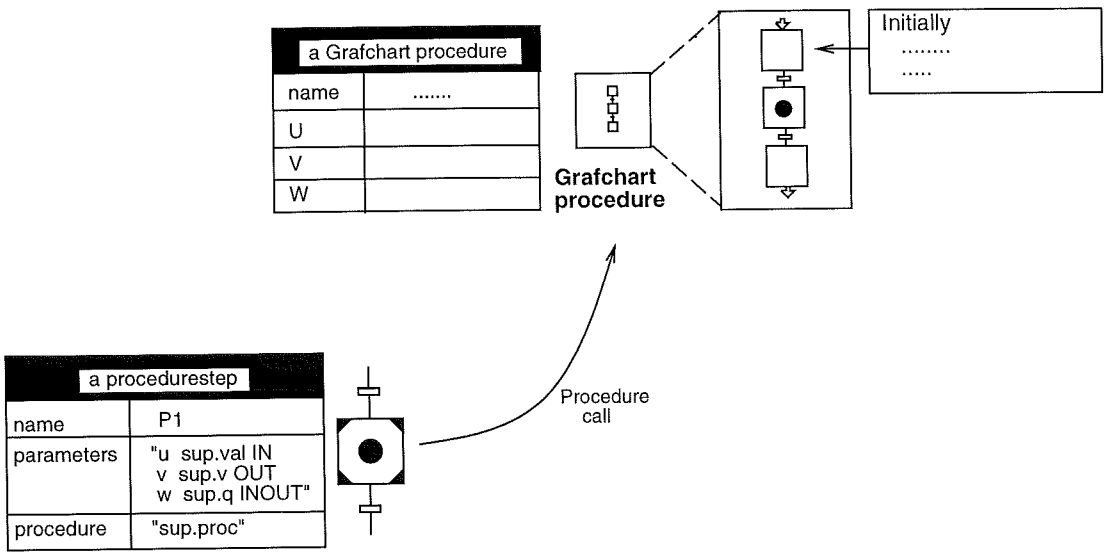


Figure 6.4 Parameterization of a Grafchart procedure.

Implementation and compilation

During compilation the action templates are translated into the corresponding G2 rules and the sup notations are changed to the correct G2 expression. An example of this is shown in Figure 6.5. The step, named $s10$, has an attribute x. From the action associated with this step one would like to set the global variable n to the value of the attribute x. Both the action template written by the user and the corresponding automatically generated G2 rule are shown in the figure.

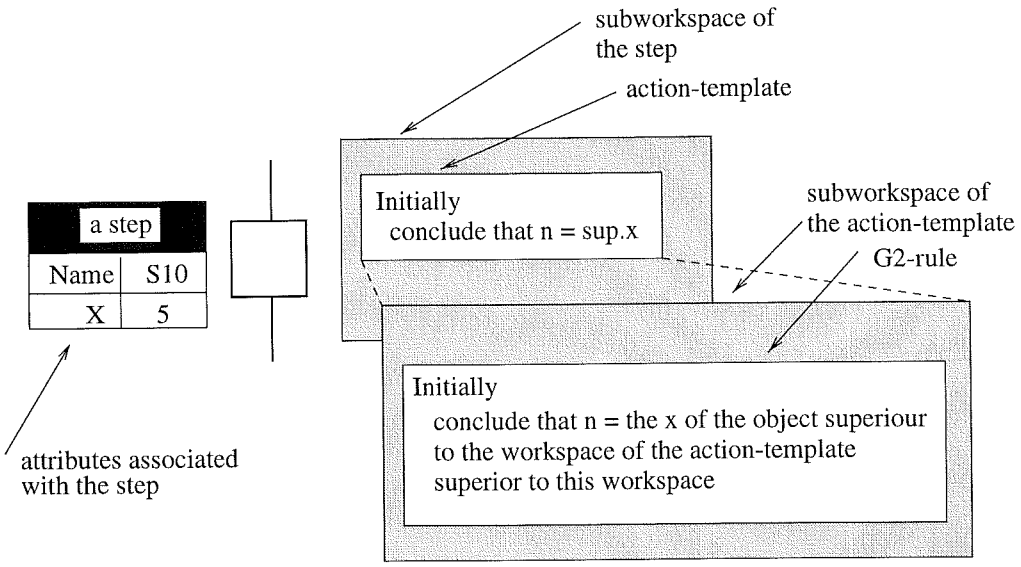


Figure 6.5 Compilation of steps and transitions.

Since lexical scoping is used, the user does not have to be concerned about which object that has the attribute x. If, e.g., step $s10$ would have been a step inside a Grafchart procedure, then x could instead have been an attribute of that Grafchart procedure. When referring to x from the action of the step, the user would have written exactly the same string but the translated G2-rule would have been different.

6.2 Methods and Message Passing

Methods and message passing are supported by allowing Grafchart procedures to be methods of general G2 objects. For example, an object representing a batch reactor can have Grafchart methods for charging,

discharging, agitating, heating etc. Inside the method body, it is possible to reference the object itself and the attributes of this object using a Smalltalk influenced notation.

- `self`
  refers to the object that the method belongs to, i.e., this object.
- `self.attribute1`
  refers to the `attribute1` attribute of this object.
- `self.attribute1^`
  refers to the object named by `attribute1` of this object.
- `self.attribute1^.attribute2`
  refers to the `attribute2` attribute of the object named by the `attribute1` attribute of this object.

References to attributes of the object that the method belongs to can be combined with parameter references using the `sup.attribute` notation.

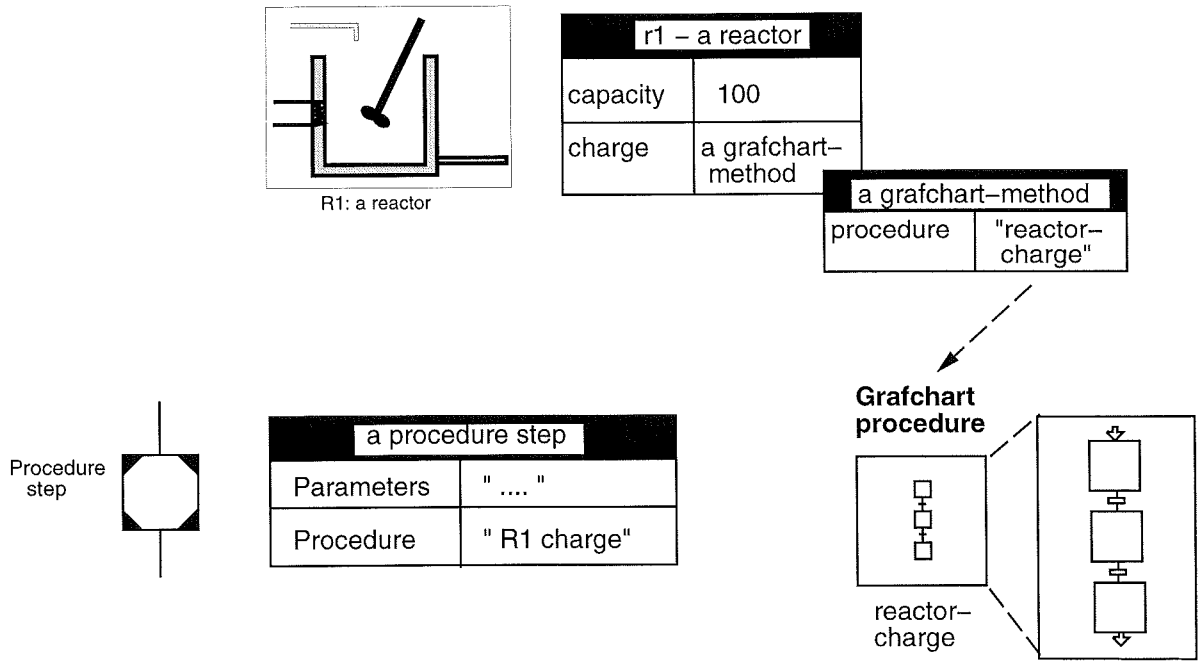


Figure 6.6 Grafchart methods.

The method of an object is called through a procedure or process step in the same way as if the procedure was stand-alone. Instead of giving a procedure reference, the procedure that will be called is determined by an object reference and a method reference. An example of a Grafchart method is shown in Figure 6.6. The reactor object `R1` contains the method `charge`. The method is implemented by the Grafchart procedure `reactor-charge`. The procedure step invokes the `charge` method of the `R1` object. The method that will be called can also be determined by parameters.

6.3 Object Tokens

In ordinary Grafchart a token is simply a boolean indicator telling whether a step is active or not. In High-Level Grafchart, the object token feature allows a token to be an object. This means that the token is defined in the class hierarchy and subclasses can be made where new attributes can be added. To indicate if a token is present in a step or not a grafchart marker is used. The grafchart marker appears as a black or a coloured filled circle and it acts as a pointer to an object token. The grafchart marker is implemented in the class hierarchy and subclasses, with different-coloured icons, can be created. The pointer between the grafchart marker and the object token is implemented as a G2 relation. The object token also contains an attribute specifying the type of the grafchart marker that should be used to animate the object token. Since the object token is never visible for the HL-Grafchart user its icon is the default G2 icon. The reason for letting the grafchart marker be a pointer to an object token with attributes and not letting the grafchart marker itself contain the attributes is the way parallel structures are handled, see Chapter 6.3 (Parallelism).

The class definitions of grafchart marker and object token are shown in Figure 6.7. The grafchart-marker class has a subclass named grafchart-marker-x and the object-token class has a subclass called object-token-A. Each grafchart-marker-x points at an instance of an object-token-A.

The attributes of an object token can be referenced from the actions of the step that the corresponding grafchart marker is placed in and

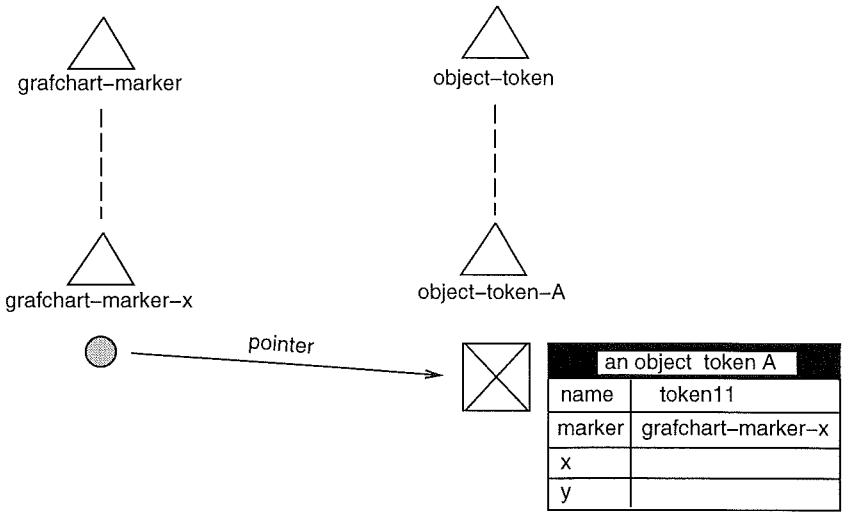


Figure 6.7 A grafchart marker and an object token.

from the receptivities of the transitions that the grafchart marker is currently enabling. There might be several grafchart markers, of the same or of different classes, pointing to different object tokens, in the same step.

One of the ideas of introducing object tokens is that the description of a system can be made more compact. Imagine a system with two identical tanks which both should be filled with water, heated and then emptied. The tanks operate independently of each other. Each tank system can be controlled by a separate Grafchart and information about e.g., the temperature, the level etc., can be stored as attributes of the entire chart, see Figure 6.8.

If however, HL-Grafchart is used, the two tanks can be controlled by one chart in which there are two object tokens, one for each tank. The information about, e.g., the temperature, the level etc., can no longer be stored in the attributes of the chart since they do not have the same values for the two tank systems. Instead the information is stored as attributes of the object tokens, see Figure 6.9.

The corresponding grafchart markers can move around in the function chart independently of each other as in Figure 6.9 where the behavior of one tank does not depend on that of the other tank. However, applications also exist where the object tokens are not independent of each other, e.g., a production line where the different parts move

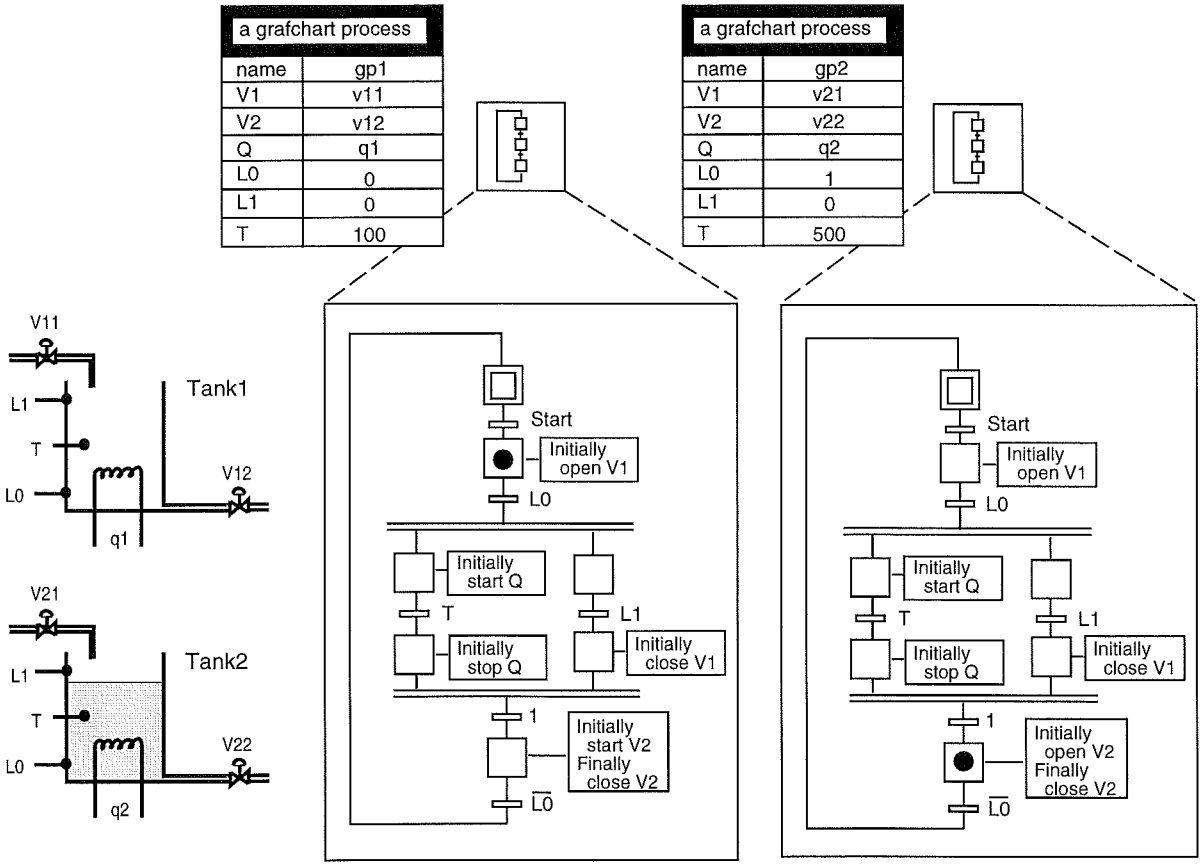


Figure 6.8 A system composed of two identical tanks, each controlled by a Grafchart.

between different stations but where it is impossible for one part to pass another. This means that grafchart markers in a chart can not pass each other and a grafchart marker may therefore have to wait for another grafchart marker to leave a step. Another example with dependencies between the object tokens is batch production. The batches in a batch production cell are effected by different recipe operations and they share the same equipment. One batch might, e.g., have to wait for an other batch to release a resource. High-Level Grafchart can be used to model structures with dependency between the object tokens.

A grafchart marker, or a grafchart marker together with its object token, will most often be refered to by the shorter term `token`. Only when we want to stress the fact that the grafchart marker and the object token are two different objects their correct names will be used.

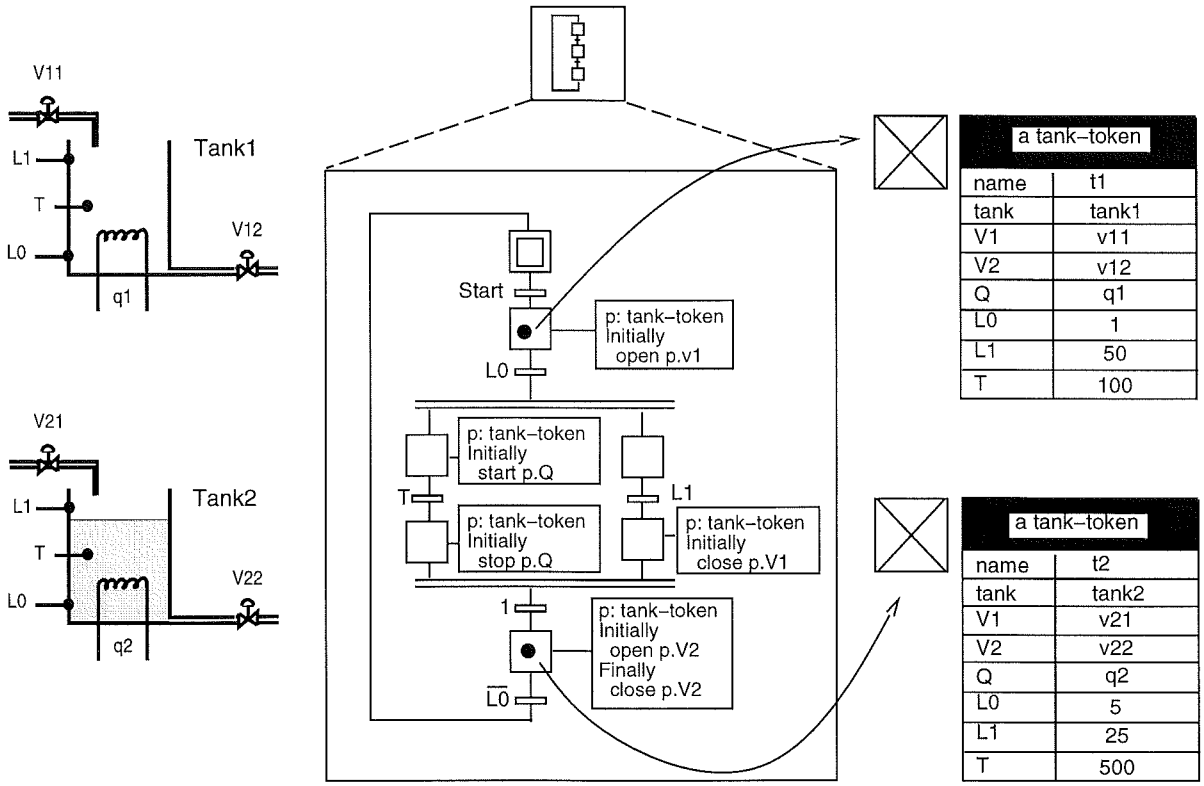


Figure 6.9 A system composed of two identical tanks, controlled by one HL-Grafchart.

Steps and Actions

A step may contain several tokens of the same or of different classes. To each step, actions can be associated. The action types are the same as in Grafchart, i.e., initially, finally, always and abortive. The difference is that in High-Level Grafchart each action is associated with a token class. An initially (finally) action is executed when an instance of its token class enters (leaves) the step. An always action is executed when an instance of its token class is present in the step. The action may contain conditions that depend on the presence of tokens of other classes and on the values of their attributes.

In Figure 6.10 a step and its associated actions are shown. Two tokens are placed in the step, one of class `P-token` and one of class `Q-token`. One `initially` and one `always` action are associated with the token class `P-token` and one `finally` action is associated with the token class `Q-token`.

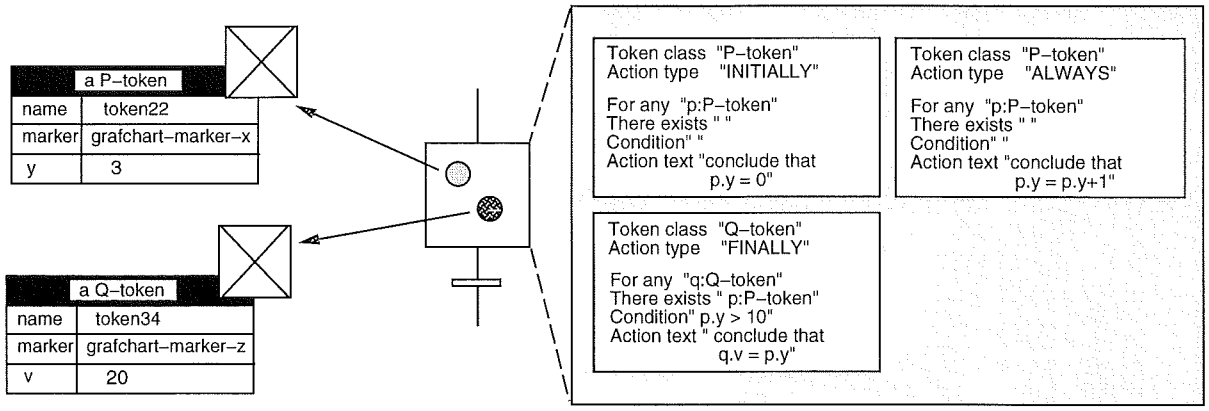


Figure 6.10 Step actions.

The action-templates are built up by six different parts. In the `token class` part, the class of the token to which the rule applies is specified. The `action type` indicates if the action is an initially, finally, always or abortive action. In the `for any` part, the token to which the rule applies is given a temporary name, this name can be used in the action-template to refer to the token. The `there exists` part can be used to check the presence of other tokens and the `condition` part can used to check if certain conditions apply. The last part of the action-template is the `action text` where the action that should be performed is specified.

When a token of type `P-token` enters the step in Figure 6.10 its y attribute is assigned the value zero, as specified by the initially action-template. If the token does not directly leave the step, the always action-template will assure that the y attribute is incremented periodically. The finally action-template associated with the token class `Q-token` specifies what should happen with a token of type `Q-token` when it leaves the step. If a token of type `P-token` exists in the step, and, if the value of the y attribute of the `P-token` is greater than ten, then, when a token of type `Q-token` leaves the step, the value of its v attribute will be assigned the same value as the value of the y attribute of the `P-token`.

The actions are, during compilation, translated into the corresponding G2 rules.

Transitions and Receptivities

Each transition has a receptivity for each token class that the transition can be enabled by. The condition and/or the event of the receptivity may refer to the attributes of the token class instance that enables the transition. It may also refer to the presence of other tokens in the input step and the value of their attributes.

In Figure 6.11 a transition and its receptivities are shown. The transition has two receptivities, one associated with the token class `P-token` and one associated with the token class `Q-token`.

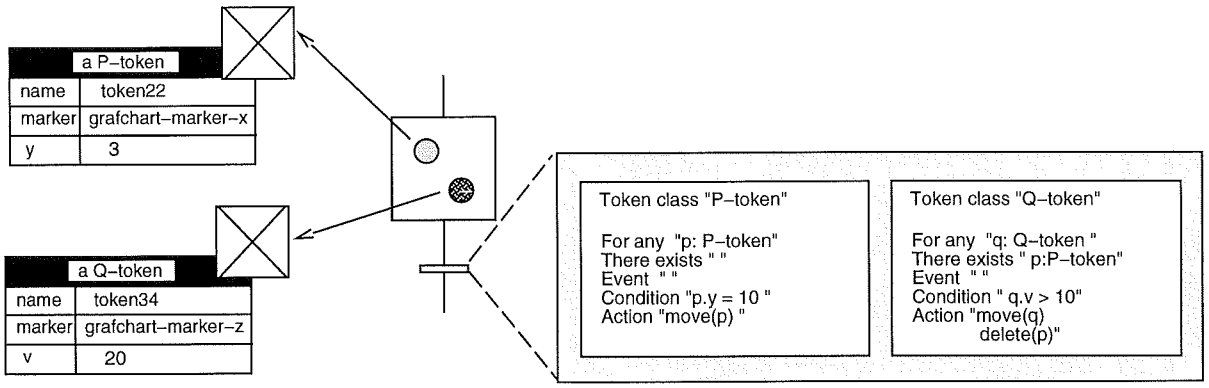


Figure 6.11 Transition with multiple receptivities.

The receptivity template is built up by six parts. The `token class`, `for any` and `there exists` parts are used in the same way as those of the action-template. The `event` and `condition` parts are used to specify the event and conditions of the receptivity. When a transition becomes fireable an operation is performed on the tokens that are referenced by the receptivity. The operation is specified in the `action` part of the receptivity-template.

In the standard case the operation would be to move the token from the input step to the output step. However, it is also possible to delete and create tokens and to change the value of the attributes of the tokens. The latter is useful primarily for initializing the values of a newly created token. The operations can also be more complex, e.g., an attribute of the token can be changed at the same time as the token is moved from the input step to the output step, another token placed in the input step can be deleted or a new token can be created and placed in the output step. It is also possible to move, not only the

token that enables the transition but also to move one or several of the tokens refered to in the condition or event part of the receptivity.

The action `move(p)` implies that the grafchart marker that is placed in the input step of the transition and that points on the object token named `p` is moved from the input step of the transition to the output step. The action `create(P-token)` implies that an instance of a P-token is created. A grafchart marker, of the class that can animate a P-token is also created and placed at the output step of the transition. The pointer from the grafchart marker to the P-token is created. The action `delete(p)` deletes the grafchart marker that is placed in the input step of the transition and that points to the object token named `p` and deletes the object token named *p*.

In Figure 6.11, two tokens are placed in the step preceding the transition, one token of class `P-token` and one token of class `Q-token`. The transition is enabled with respect to both token classes but it is only fireable with respect to the token of class `Q-token`. When the transition fires the token of class `Q-token` will be moved from the input step to the output step of the transition and the token of class `P-token` will be deleted.

Parallelism

The reason for having a grafchart marker that acts as a pointer to the corresponding object token is the way parallel structures are handled.

When the transition preceding a parallel-split (and-divergence) is fired, a grafchart marker is moved from the input step of the transition, to the first step in each parallel branch. The grafchart markers, that are added in a step, are all copies of the grafchart marker removed from the input step. The grafchart markers in all the parallel branches will therefore point to the same object token according to Figure 6.12. If the value of an attribute is changed in one of the branches it will directly be visible in all branches.

The transition after a parallel-join (and-convergence) is only enabled with respect to a token class if all the input steps of the transition contain grafchart markers that point at the same object token. When the transition is fired, one grafchart marker from one of the preceding steps in the parallel branch is moved to the output step, the rest

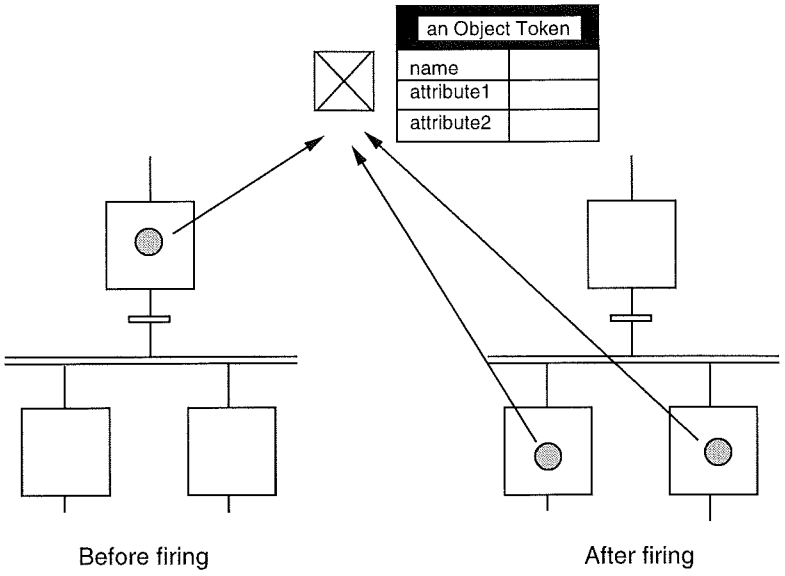


Figure 6.12 An and-divergence situation.

of the grafchart markers are deleted. Since all grafchart marker, in the parallel branch, point to the same object token it does not matter which one that is moved and which is deleted. The parallel-join (and-convergence) situation is shown in Figure 6.13.

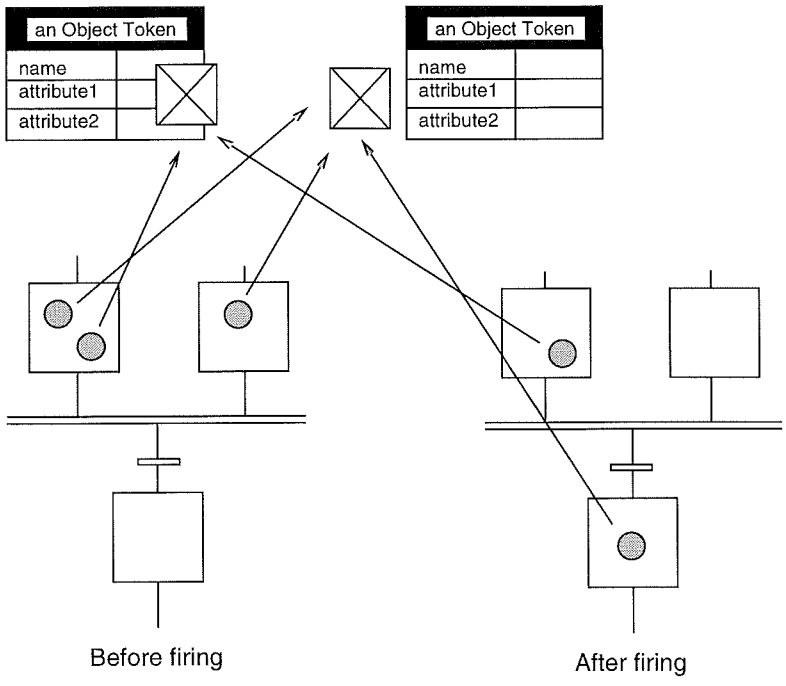


Figure 6.13 An and-convergence situation.

The reason for letting the grafchart marker point to an object token, containing the attributes, and not letting the grafchart marker itself contain the attributes, is the parallel-join structure. If the grafchart marker would contain the attributes and if the value of an attribute of one of the grafchart markers, in one of the parallel-branches, is modified, it would be unclear how the parallel-join (and-convergence) should be treated. Which value of the modified attribute should be kept and which should be ignored? By letting the grafchart markers be pointers to an object token containing the attributes, problems like this are avoided. This is also a natural way to model a system since a grafchart marker in a chart is a marker visualizing the state of only one object. For example, the grafchart markers in Figure 6.9 each represent the state of one tank. Even if the state is visualized by a parallel-branch (the tank is heated and filled at the same time) the markers still represent one tank.

Arc inscriptions

Most of the existing work on High-Level Petri Nets and Coloured Grafcet, [Agaoua, 1987], [Suau, 1989], is based on arc inscriptions. Arc inscriptions can be of two main types: the function representation and the expression representation, see Chapter 4.1.

In High-Level Grafchart explicit arc inscriptions are used very rarely. This does however not restrict the different ways of firing a transition. By instead allowing the receptivities of the transition to be written in several different ways, the same performance, as the one achieved using arc inscriptions in H-L Petri Nets, can be achieved. Color transformations, attribute changes, and deletion and creation of tokens are all possible to do. Using arc inscriptions one easily loses the clarity of the net. Since one of the main advantages of Grafcet is its clear and intuitively understandable way of representing sequences, it is undesirable to use arc inscriptions in HL-Grafchart.

PN-transition

A major reason why arc inscriptions are necessary in High Level Petri Nets is that transitions may have multiple input places and multiple output places, i.e., and-convergence and and-divergence structures, where different tokens are involved and where different tokens follow

different branches. A PN situation like this is shown in Figure 6.14. The arc inscriptions are needed to specify how the different input places should contribute to the enabling of the transition and how the different output places should be affected when the transition is fired.

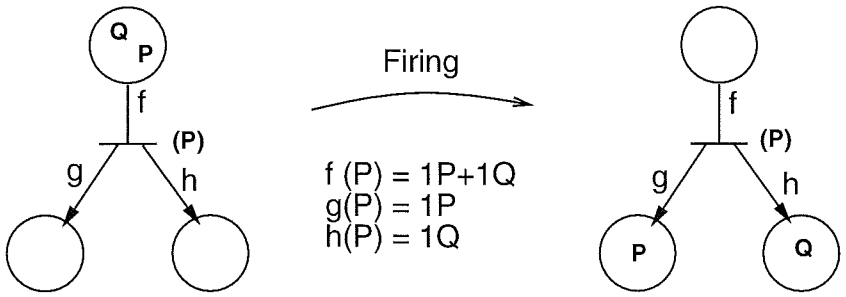


Figure 6.14 A Petri Net and-divergence structure.

The transition is enabled with respect to the colour `P`. When firing the transition one token of colour `P` and one token of colour `Q` should be removed from the preceding place, the `P` token should follow the left path and the `Q` token should follow the right path.

In HL-Grafchart situation like this can be handled in two different

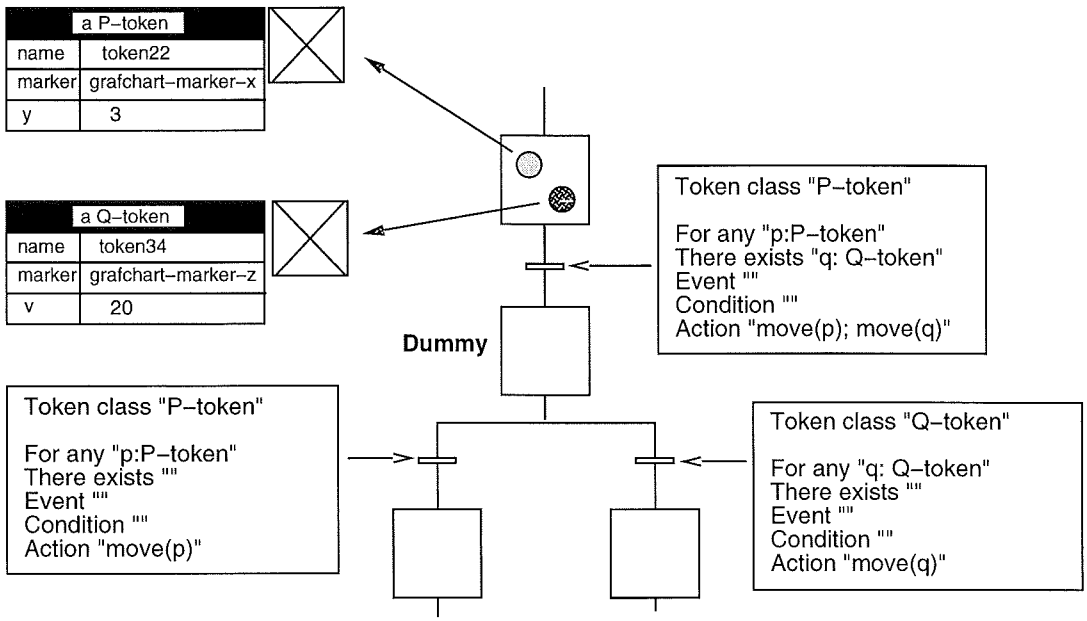


Figure 6.15 An and-divergence situation transformed into an or-divergence situation.

ways. The first solution involves a structure transformation. The and-divergence case is transformed into an or-divergence case and an extra empty dummy step is introduced, see Figure 6.15. The first transition contains a receptivity that checks if there is a token of each class in the input step, if this is the case the two tokens will be moved to the dummy step. Two transitions follow the dummy step, one that is immediately fireable with respect to one of the classes and one that is immediately fireable with respect to the other class.

In some applications, however, this structure transformation can be undesirable. Therefore a special kind of transition, named PN-transition, is introduced. A PN-transition can have more than one incoming arc and more than one outgoing arc. The arcs are numbered and the numbers can be used to specify from which step and to which step the token should be moved. The arc numbers of PN-transitions is the only case of arc inscriptions in HL-Grafchart.

In Figure 6.16 a PN-transition is shown.

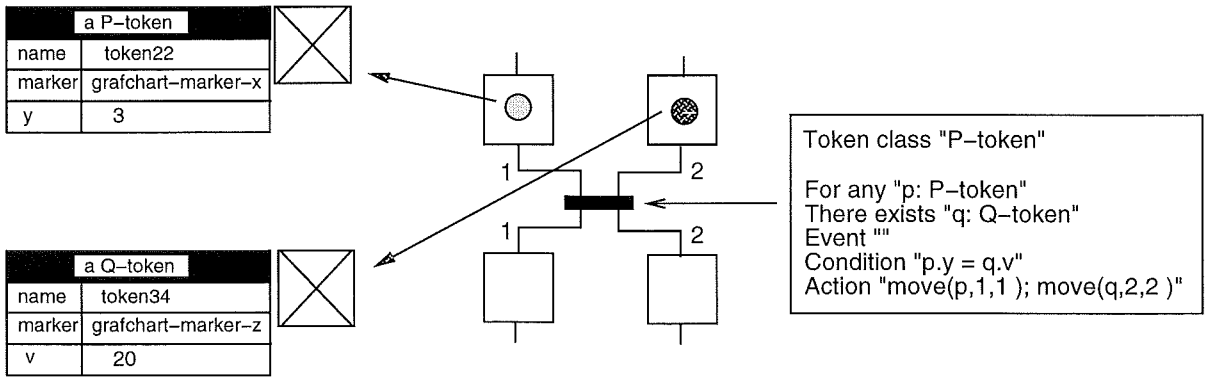


Figure 6.16 An and-divergence situation using a PN-transition.

Macro steps

As soon as a grafchart marker enters a macro step a copy of the grafchart marker is placed in the enter-step of the macro step, i.e., the grafchart marker placed in the macro step and the grafchart marker placed in the enter-step points at the same object token. Several token can be in a macro step at the same time.

Procedure- and Process steps

In ordinary Grafchart a procedure or process step has an associated attribute that specifies the name of the Grafchart procedure that should be called. In High-Level Grafchart the call to a Grafchart procedure is more flexible.

Associated to each procedure or process step is a procedure-call-template. A procedure-call-template is built up by three parts. The `token class` part specifies the class of the token to which the templates apply. The `parameter` part specify the attributes to the procedure, if any, and the `procedure` part determines the name of the Grafchart procedure to be called. The name of the Grafchart procedure can either be given explicitly or implicitly through a reference. It is possible to reference attributes visible in the procedure step or process step context through the `sup.` notation. In a similar way, attributes of the token itself can be referenced.

- `inv.attribute1`
  refers to the `attribute1` attribute of the token

In Figure 6.17 a procedure step is shown. Two tokens are placed within the procedure step, one token of class `P-token` and one token of class `Q-token`. Tokens of class `P-token` cause calls to a Grafchart procedure named `gp1` whereas tokens of class `Q-token` cause calls to a Grafchart procedure named by the `proc` attribute of the token itself. One of the attributes of the Grafchart procedure `gp1` is named `x`. This attribute is given the value of the `x` attribute of the `P-token` giving rise to the call.

Each call is executed in its own copy, i.e., when a call is done to a Grafchart procedure a copy of the Grafchart procedure body is created and this copy is executed, when the execution reaches its end the copy is deleted. This means that there can never be more than one token in each Grafchart procedure and therefore the Grafchart procedures of ordinary Grafchart can be used also in HL-Grafchart. The attributes of the token are, if necessary, transformed into attributes of the Grafchart procedure called by the token.

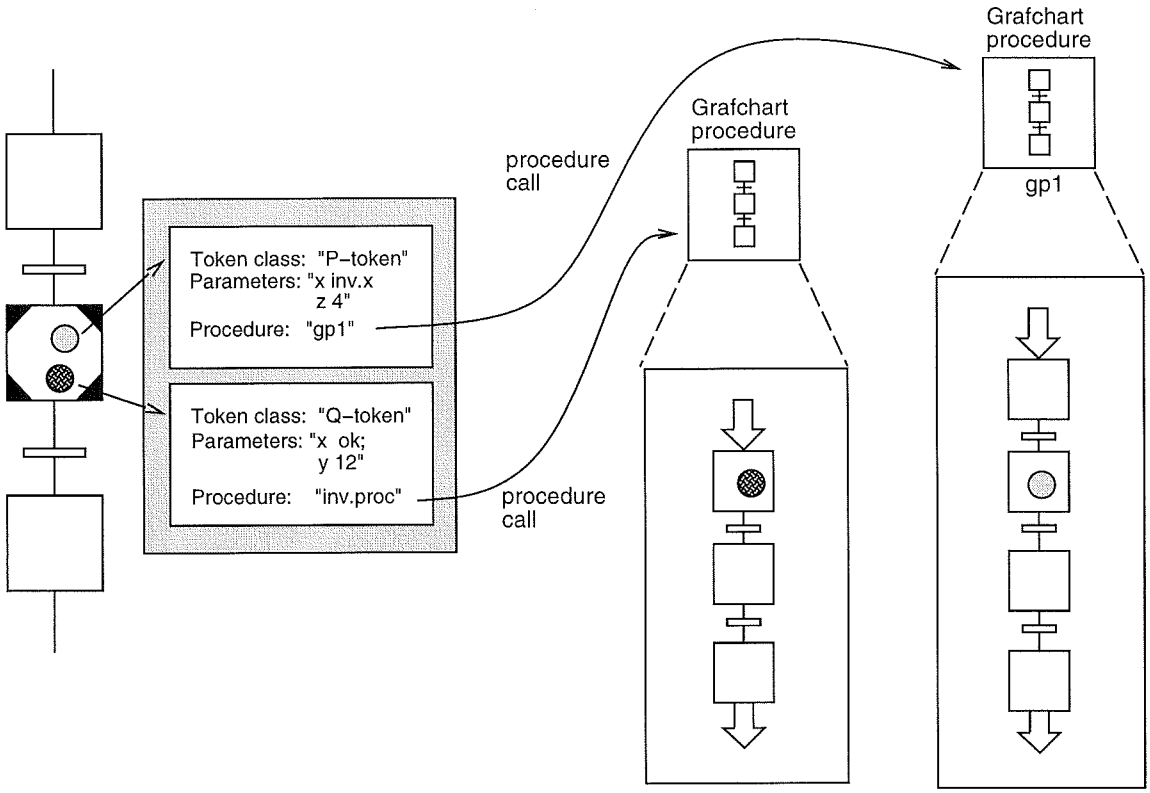


Figure 6.17 Two tokens placed in a procedure step.

6.4 Multi-dimensional Charts

Since a token is an object and since objects are allowed to have methods also tokens may have methods. This is the basis for the multi-dimensional chart feature, whereby a token moving around in a chart may itself contain one ore more charts. This gives several interesting structuring possibilities.

A part of a multi-dimensional function chart is shown in Figure 6.18. It consists of one step and one process step. A token of class `P-token` is placed in the step and a token of type `Q-token` is placed in the process step. The name of the Grafchart procedure to be called from the process step is given by the reference `inv proc`, i.e., the Grafchart procedure to be called is a method named `proc` of the token.

The `Q-token`, placed in the process step, in Figure 6.18 has caused a call to the Grafchart procedure gp2. The `Q-token` can however, continue its execution independently of the execution of gp2. When the `P-token`

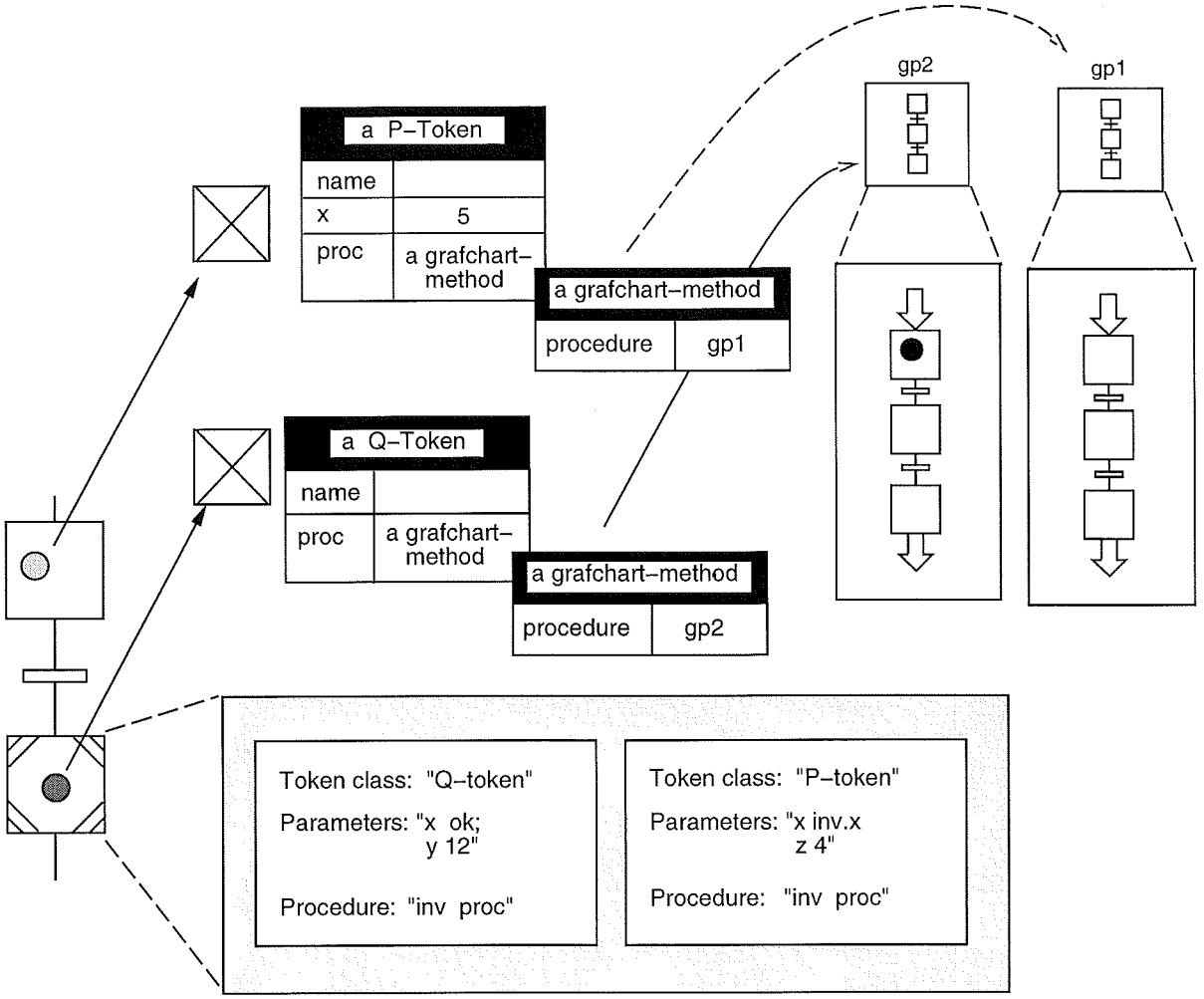


Figure 6.18 A multi-dimensional chart.

enters the process step a call to the Grafchart process gp1 will be effected.

The reference to the attributes of an object token is done in different ways depending from where the reference is done, see Figure 6.19. If the attribute of an object token should be referenced from within the chart where the corresponding grafchart marker is placed, the `temporary-token-name.attribute` notation is used, and, if the attribute should be referenced from within a method that belongs to the object token, the `self.attribute` notation is used.

The different levels in a multi-dimensional function chart can commu-

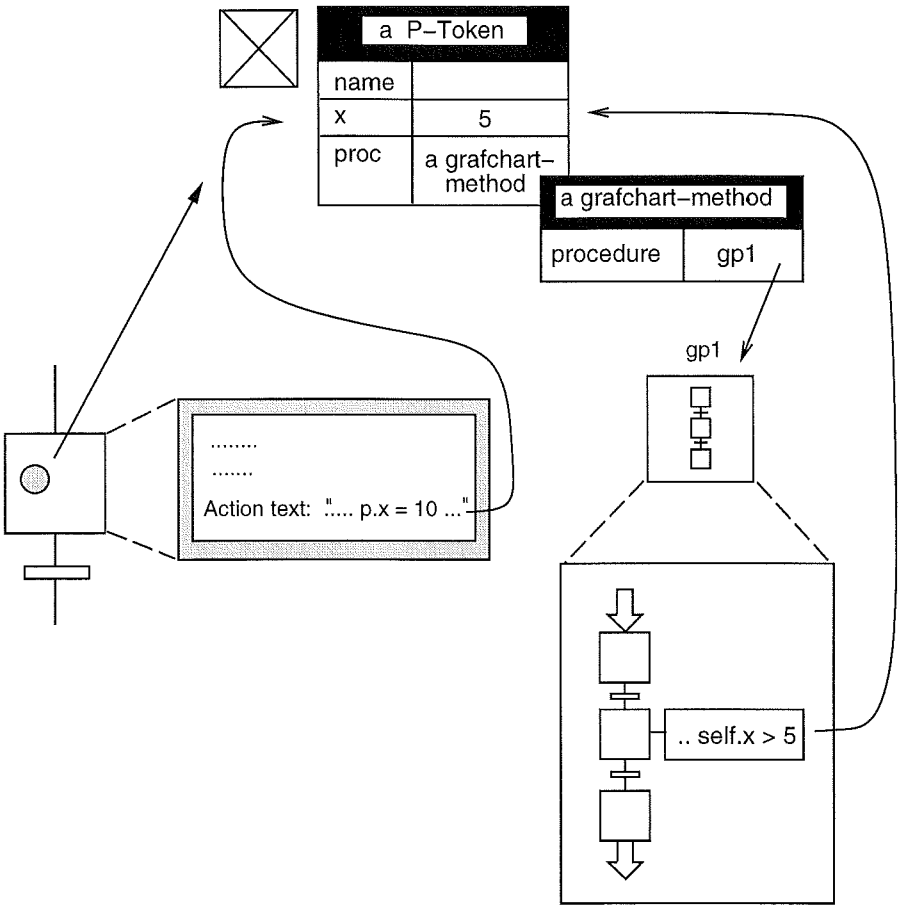


Figure 6.19 Different references to an attribute of an object token.

nicate with each other through the different dot notations that exist, see Figure 6.20.

Four different dot notations exist:

1. The `sup` notation, described in Chapter 6.1
2. The `self` notation, described in 6.2
3. The `temporary-token-name` notation, described in 6.3 (Steps and Actions)
4. The `inv` notation, described in 6.3 (Procedure- and Process steps).

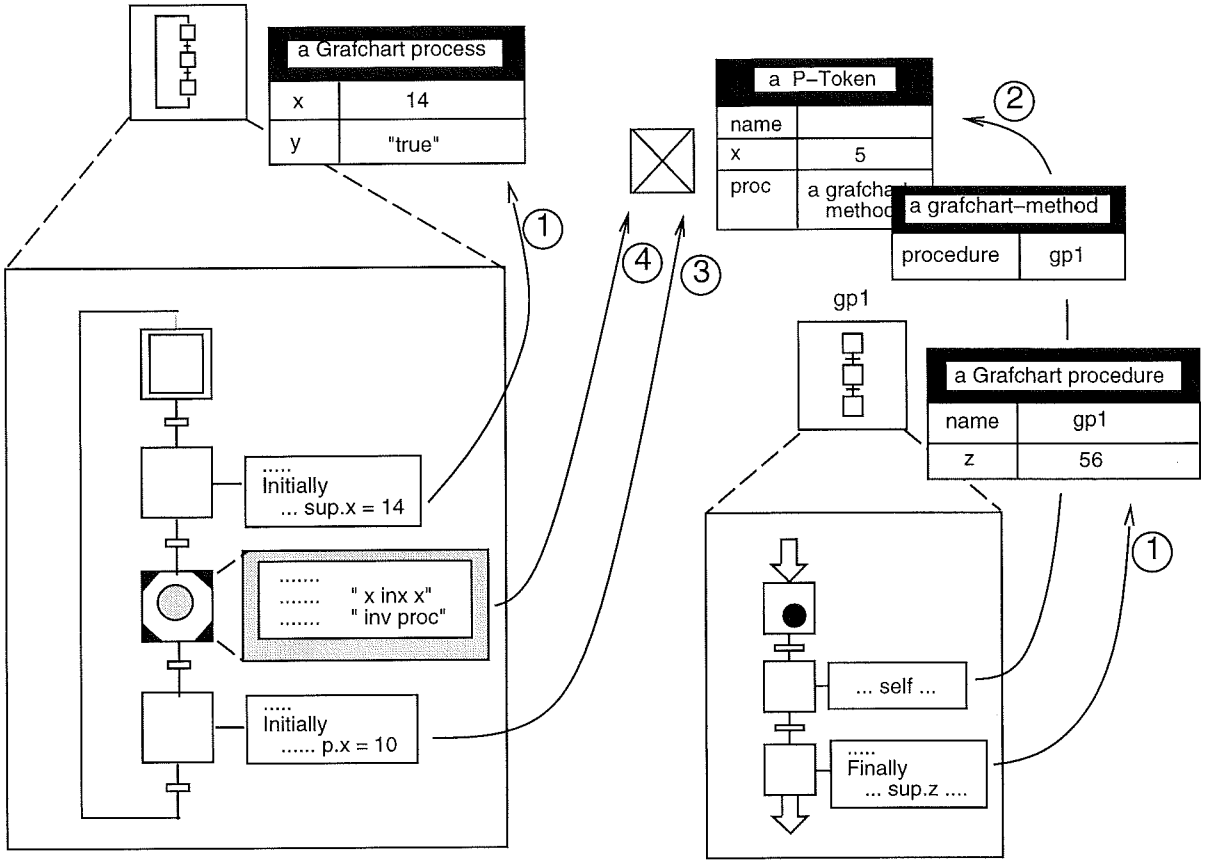


Figure 6.20 The communication between the different levels in a multidimensional function chart.

6.5 Implementation

HL-Grafchart is implemented in G2, [Moore *et al.*, 1990], see Appendix A.

Class Hierarchy

The graphical language elements in HL-Grafchart are defined in a class hierarchy, see Figure 6.21. The class names are extended with '-o' to indicate that these classes are used in HL-Grafchart.

To specify the actions associated with a step the user uses an object called `action-o`, to specify the receptivities associated with a transition the uses uses an object called `receptivity-o` and to specify the name of the Grafchart procedure that is to be called from a procedure or process step an object called `procedure-call-o` is used. The class hierarchy of

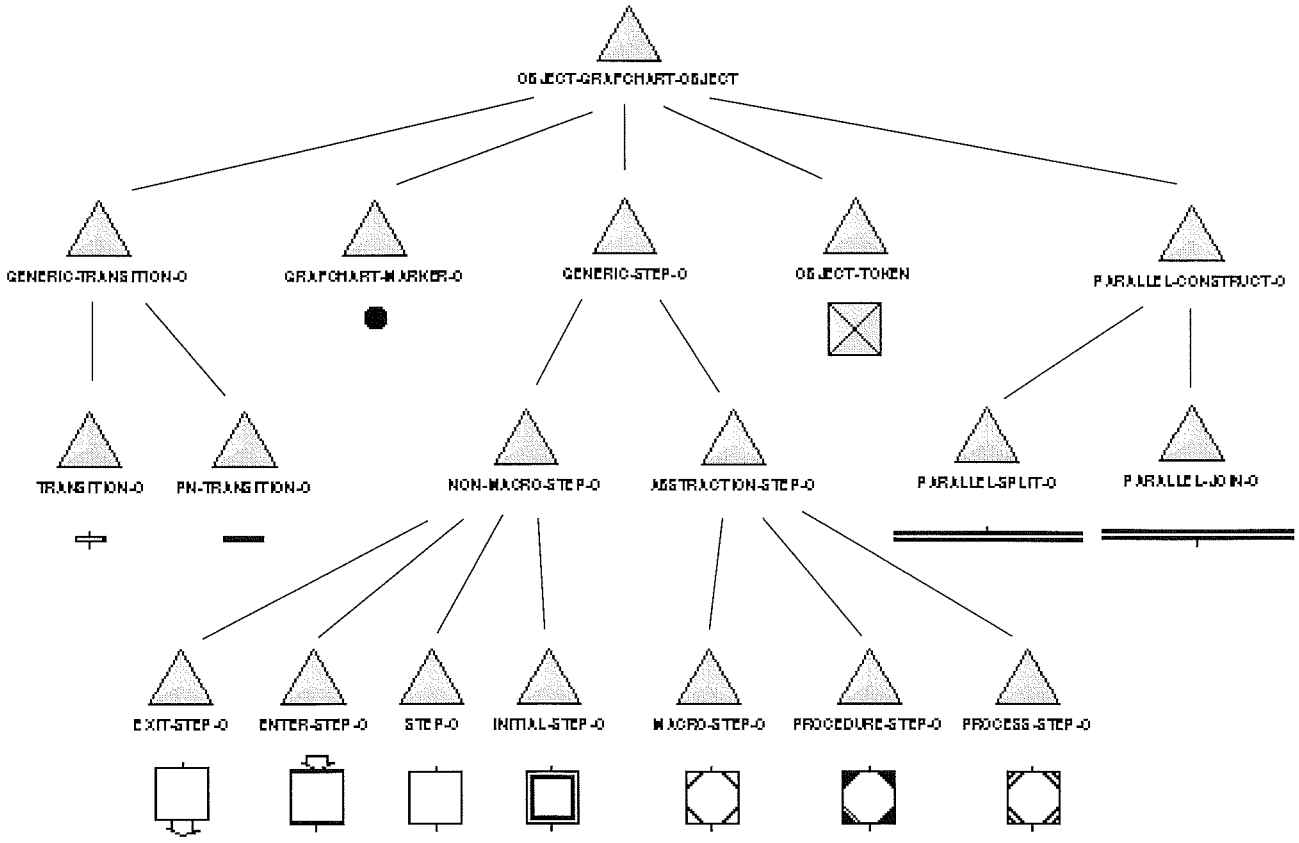


Figure 6.21 The class hierarchy in HL-Grafchart.

these three "help-elements" and their icons are shown in Figure 6.22 and 6.23.

Relations

The implementation relies on the use of G2 relations. The following relations are used.

The relation between a grafchart marker and an object token is called `animating`. A grafchart marker may be animating at most one object token. The inverse relation is called `animated-by`. An object token may be animated-by more than one grafchart marker.

The relation between a grafchart marker and a step is called `placed-at` and the inverse relation is called `holding`. A grafchart marker may be placed-at at most one generic-step but a generic-step may be holding more than one grafchart marker.

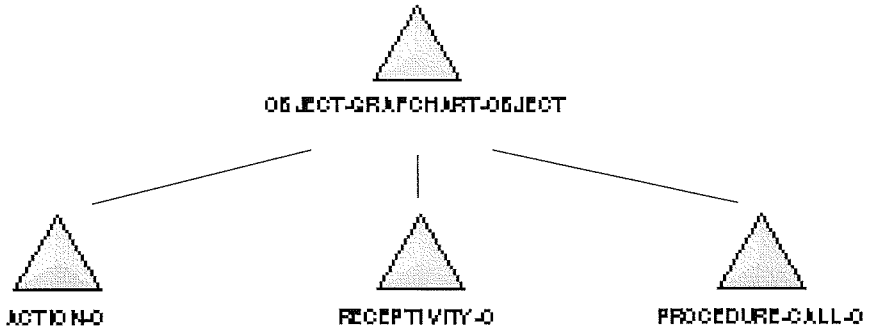


Figure 6.22 The class hierarchy in HL-Grafchart.

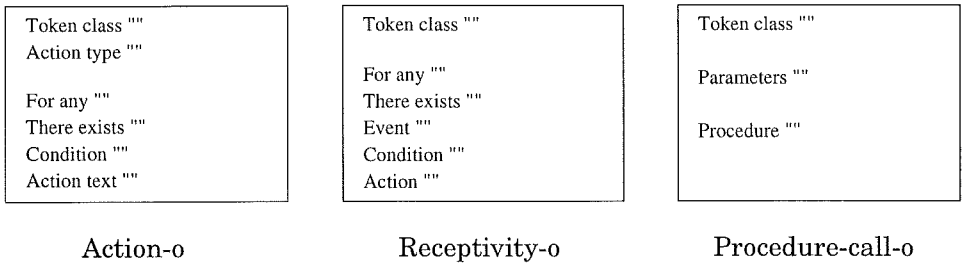


Figure 6.23 The class hierarchy of the "help-elements" in HL-Grafchart.

A relation exists between grafchart markers and receptivities. This relation is called `activating`, the inverse relation is called `activated-by`. A grafchart marker may be activating more than one receptivity. A receptivity may be activated-by more than one grafchart marker.

A relation exists between object tokens and receptivities. This relation is called `enabling`, the inverse relation is called `enabled-by`. An object token may be enabling more than one receptivity. A receptivity may be enabled-by more than one object token.

The relation between an object token and an action is called `action-invoking`. The inverse relation is called `action-invoked-by`. An object token may be action-invoking more than one action. An action may be action-invoked-by more than one object token.

A relation exists between a grafchart marker and a Grafchart-procedure. This relation is called `marker-invoking` and the inverse relation is called `marker-invoked-by`. A grafchart marker may be marker-invoking more than one Grafchart procedure. A Grafchart procedure may be marker-invoked-by at most one grafchart marker.

Compilation

Before a HL-Grafchart can be executed it has to be compiled. During compilation all actions and all receptivities are translated into the corresponding G2 rules. This means that all `sup` notations and `self` notations are replaced by a G2 expression. Most often the G2 expressions become very complex and hard to read, this explains why the dot notation was introduced.

The finally action in Figure 6.10 and its corresponding G2 rule is shown in figure 6.24.

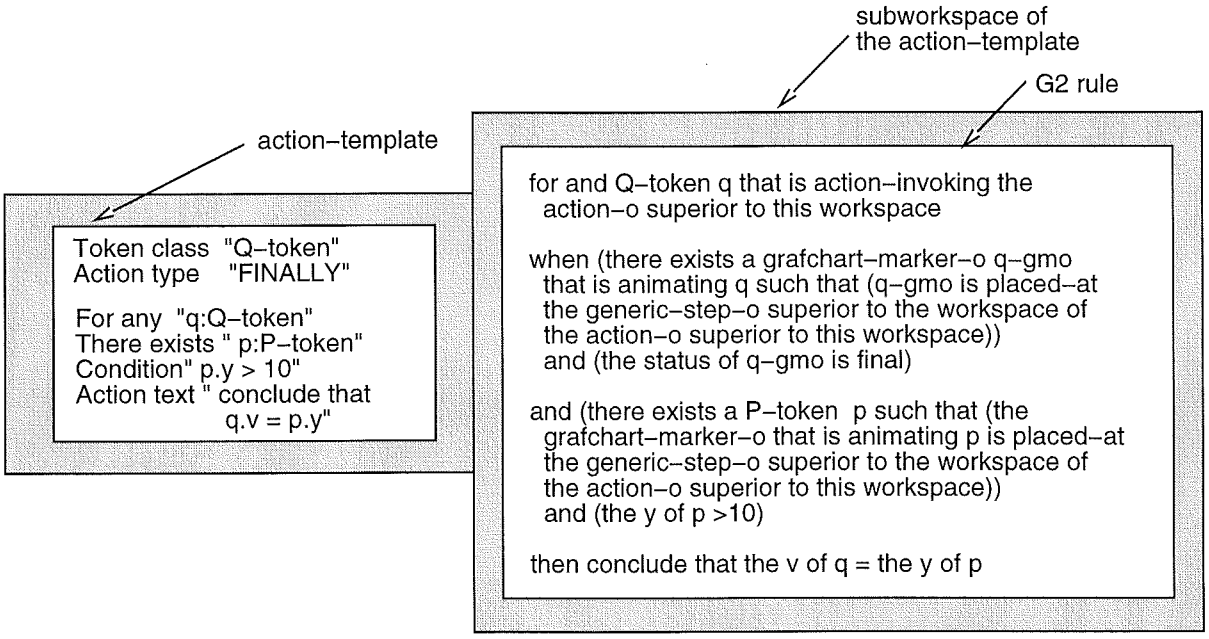


Figure 6.24 Step action and G2 rule.

6.6 Application

High-Level Grafchart has been used in some applications.

Lego car factory

A High-Level Grafchart model of the control system for a LEGO car factory has been implemented. The LEGO car factory is a small car factory model built in LEGO that assembles toy LEGO cars. The factory

itself is built in LEGO and consists of four conveyor belts, three storages for chassis, car frames, and car bodies respectively; two pressing machines, three machines that take parts from storage, and one machine that turns the car on the conveyer belt. The factory is supplied with several position sensors.

The model of the control system consists of two function charts. The car-controller is a straight sequence where each token represents a car. Each macro-step represents a certain operation that is performed on the car. The machine-controller has one token for each machine. The actions and conditions in the car-controller depend on the presence of a car in a certain step. The control system is structured into one machine part and one media part. The media in this case is the cars. Similar structuring concepts can also be applied to general discrete manufacturing problems. The LEGO car factory is presented in [Årzén, 1994a].

Alarm filtering

High-Level Grafchart has been used to implement alarm filters. Information overload due to large numbers of warnings and alarms is a common problem in, e.g., the process industry. This typically occurs in fault situations when a primary alarm is accompanied by large numbers of secondary alarms. Another case is nuisance alarms caused by improperly tuned alarm limits. One way of handling excess alarms is to use alarm filters. The task of the alarm filter is to filter out alarms and to aggregate multiple alarms into high-level alarms. In many cases it is the combination of multiple alarms that have occurred within a certain time period that gives the best indication of what is happening in the process.

Grafchart can be used to represent alarm (event) patterns. The approach is based on the possibility to use a finite state machine as an acceptor for strings in a formal language. Since Grafchart is an extended state machine with temporal facilities it is possible to accept more general expressions than the regular expressions that are possible with ordinary state machines.

Due to the sampled nature of the control system and to measurement noise, in some cases the relative ordering between a pair of low-level

alarms may be unimportant. Situations also exist when one wants to specify that n out of m alarms should have occurred, or that some alarms in a pattern could be omitted. By utilizing the possibilities in Grafchart for parallel and alternative paths, different sequence patterns of these types can be easily expressed. Situations with temporal constraints between alarms is important can also be expressed with Grafchart. Representation of different event patterns in Grafchart is presented in detail in [Årzén, 1996a].

Batch recipe structuring

Batch recipe structuring is the main application area of High-Level Grafchart. High-Level Grafchart is used at two levels, to represent the sequential structure in the recipes and to represent the sequence control logic contained in the equipment units. A number of different ways to structure the recipes have been investigated, see Chapter 9. Different recipe structures are also presented in [Johnsson and Årzén, 1996a], [Årzén and Johnsson, 1996], and [Johnsson and Årzén, 1994].

6.7 Summary

High-Level Grafchart is an extended version of Grafchart. It combines the graphical language of Grafcet/SFC with high-level programming language constructs and ideas from High-Level Petri Nets. This widely increases the expression power and structuring facilities of Grafchart. High-Level Grafchart adds four new features to Grafchart; parameterization, methods and message passing, object tokens and multidimensional charts. High-Level Grafchart is implemented in G2.

7

Batch Control Systems

Industrial manufacturing and production processes can generally be classified as continuous, discrete or batch, [Fisher, 1990]. How a process is classified depends mainly upon how the output from the process appears.

In continuous processes, products are made by passing materials through different pieces of specialized equipments. Each of these pieces of equipment ideally operates in a single steady state and performs one dedicated processing function. The output from a continuous process appears in a continuous flow, [SP88, 1995].

In discrete manufacturing processes, products are traditionally manufactured in production lots based on common raw materials and production histories. In a discrete manufacturing process,a specified quantity of products moves as a unit (group of parts) between workstations and each part maintains its unique identity, [SP88, 1995]. The output from a discrete manufacturing process appears one by one or in quantities of parts.

A continuous process might be, e.g., water purification plants, paper mills or the generation of electricity from a power plant, whereas a discrete manufacturing process could be, e.g., the production of cars.

In batch manufacturing the output appears in lots or in quantities of materials. The product produced by a batch process is called a batch. Batch processes are neither continuous nor discrete, yet they have characteristics from both.

An example of a batch process, [Rosenhof and Ghosh, 1987], taken from our daily life, is the preparation of a good cake. The work can be divided into three major tasks: preparation, cooking, and cooling and storing. The three tasks can be broken down into a sequence of substeps. The steps in each task should be done in a proper order to make a good cake. If not done in this way, the cake might not be very tasty and, as explained below, do not adhere to the formal definition of a batch process.

Batch processes define a subclass of sequential processes. The difference is that sequential processes not necessarily have to generate a product whereas batch processes do. In industry, batch and sequential process are used in many ways and in many areas: food, beverages, dairy processing, pharmaceutical, biotechnical manufacturing and chemical plants.

A formal definition of batch process is given by [Shaw, 1982]:

> *A process is considered to be batch in nature if, due to physical structuring of the process equipment or due to other factors, the process consists of a sequences of one or more steps that must be performed in a defined order. The completion of this sequence of steps creates a finite quantity of finished product. If more of the product is to be created, the sequence must be repeated.*

An other definition is given by [SP88, 1995]:

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

7.1 Batch Processing

Continuous processes have been used to produce many products that were originally produced by batch processes. One reason for this is that batch processes have typically been labor intensive and experienced operators have been necessary to produce batch products with

consistent quality. Since continuous plants were those that produced the largest volume of products this is where most of the research and development money were spent. However, increasing demands on flexibility and customer-driven production has led to an increased interest in batch processes. This is because batch processes are more economical for small scale production, as fewer pieces of process equipment are needed, and intermediate storage are not very expensive. Batch plants can also be made highly flexible, and thereby well suited for manufacturing of special products. For example, high quality malt whisky is produced in batch processes whereas grain liquor, the basis for blended whisky, is produced in continuous processes.

There are also processes that are not easily amenable to continuous operations. Some examples are given in [Rosenhof and Ghosh, 1987]:

1. Processes with feedstocks and/or products that can not be handled efficiently in a continuous fashion, such as solids and highly viscous materials;
2. Processes in which the reactions are slow, requiring the reactants to be held in process vessels for a long time (e.g. fermentation for beer and wine);
3. Processes in which only small quantities of products and/or different grades of the same product are required in limited quantities (e.g. dyestuff and specialty chemicals);
4. Processes that need precise control of raw materials and production along with detailed historical documentation (e.g. drug manufacturing).

Typically, batch plants are used to manufacture a large number of products. Within each product a number of different grades often exist.

Batch processes can be classified by: (1) the number of products they can make and (2) the structure of the plant, [Fisher, 1990].

1. A batch process can be single-product, multi-grade or multi-product. A single product batch plant produces the same product in each batch. The same operations are performed on each batch and

the same amount of raw materials is used. A multi-grade batch plant produces products that are similar but not identical. The same operations are performed on each batch but the quantities of raw materials and/or processing conditions such as, e.g., temperatures, may vary with each batch. The multi-product batch plant produces products utilizing different methods of production or control. The operations performed, the amount of raw materials and the processing conditions may vary with each batch.

2. The basic types of batch structures are series (single-stream), parallel (multistream) and a combination of the two. A series structure is a group of units through which the batch passes sequentially. If the plant has several serial groups of units placed in parallel but without interaction the plant has a parallel structure. If interactions exists between the parallel branches a series/parallel structure is achieved. Other names for the series, parallel and series/parallel structures are single-path,multi-path and network-structure, [SP88, 1995].

The batch plant classification by product and by structure can be combined in a matrix to show the degree of difficulty in automating the various combinations. The single-product, single-path batch plant is the simplest whereas the multi-product, network-structure combination is the most difficult.

7.2 Batch Control

Batch control projects have traditionally been among the most difficult and complex to implement [ARC, 1996]. Typically, batch control projects span over a wider scope of functionality than that required for either continuous or discrete manufacturing processes. With continuous and discrete processes, a reasonable level of automation can be attained merely by implementing basic regulatory or logic control. Batch operations typically require basic regulatory and logic control operating under sequential control; which in turn, is operating under basic recipe management in order to achieve process automation. The complexity of control within a process cell depends on the equipment available within the process cell, the interconnectivity among this

equipment, the degree of freedom of movements of batches through this equipment, and the arbitration of the use of this equipment so that the equipment can be used most effectively [SP88, 1995].

The discussion about batch control systems and the progress in batch process control has been hampered by the lack of a standard terminology. In the last years, there have been three major initiatives with the aim to provide a common language. The first major effort was an outgrowth of a Purdue University workshop on batch control in the mid 1980s, [Williams, 1988]. The second was made by NAMUR, [NAMUR, 1992]. The third major effort is sponsored by the Standards and Practices division of ISA (Instrument Society of America), the International Society for Measurement and Control, [SP88, 1995]. The standard is divided into two parts, Part 1, called S88.01 deals with models, terminology and functionality. This part of the standard was approved by the main committee of ISA and ANSI in 1995. Part 2 will deal with data structures and language guidelines. It is anticipated that an IEC (International Electrotechnical Commission) approval of the first part of S88 will appear soon without any substantial changes. This will make S88.01 an international standard for batch control. The ISA S88 standard is also known under the name SP88.

7.3 The Batch Standard ISA-S88.01

The first part of the standard describes Batch Control from two different viewpoints: the process view and the equipment view, see Figure 7.1.

The process view is represented by the process model and is normally the view of the chemists. The equipment view is represented by the physical model and is normally the view of the product engineer or the process operator.

Process Model

A batch process can be hierarchically subdivided as shown in Figure 7.1 (left).

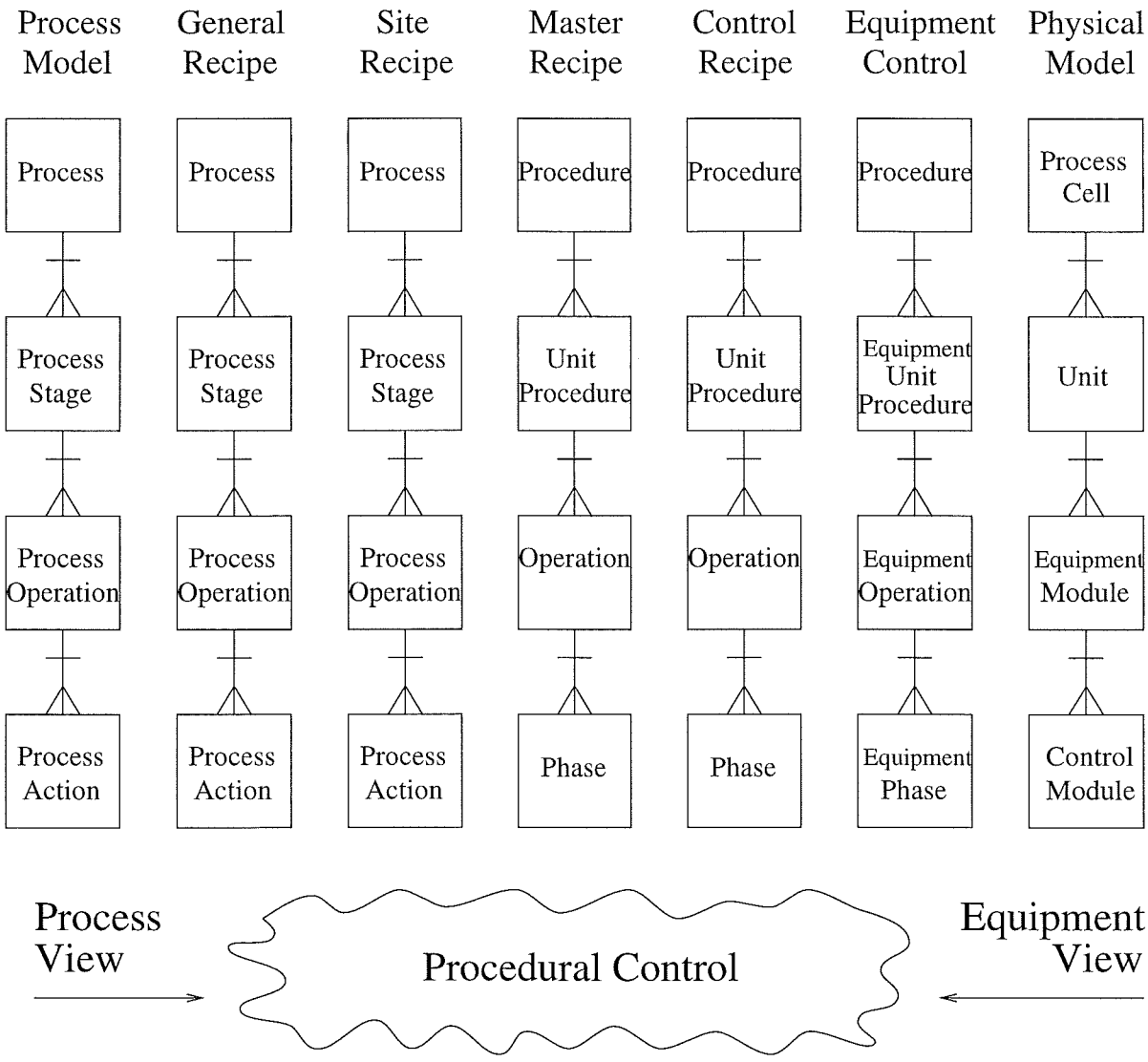


Figure 7.1 Relations between SP88 models and terminology.

- The process consists of an ordered set, serial and/or parallel, of process stages. A process stage is a part of a process that usually operates independently from other process stages. It usually results in a planned sequence of chemical or physical changes in the material being processed. Typical process stages can be, e.g., drying or polymerization.

- Each process stage consists of an ordered set of one or more process operations. Process operations describe major processing activities. It usually results in chemical or physical change in the

material being processed. A typical process operation is, e.g., react.

- Each process operation can be subdivided into an ordered set of one or more process actions that carry out the processing required by the process operation. Process actions describe minor processing activities that are combined to make up a process operation, Typical process actions are, e.g., add reactant, hold.

In the process model, the procedure for making a product does not consider the actual equipment for performing the different process steps.

Physical Model

The physical model of SP88 defines the hierarchical relationships between the physical assets involved in batch manufacturing. The model has seven levels, starting at the top with an enterprise, a site and an area. In Figure 7.1 (right) only the four lower levels are shown, with the following interpretation:

- A process cell contains one or more units.
- A unit can carry out one or more major processing activities such as react, crystallize or make a solution. Units operate relatively independently of each other. A unit is made up of equipment modules and control modules.
- An equipment module can carry out a finite number of minor processing activities like weighting and dosing. It combines all necessary physical processing and control equipment required to perform those activities. Physically, an equipment module may be made up of control modules and subordinate equipment modules. An equipment module may be part of a unit or may be a stand-alone equipment grouping within a process cell. It may be an exclusive use resource or a shared resource.
- A control module is typically a collection of sensors, actuators or controllers. Physically, a control module can be made up of other control modules.

Recipes

To actually manufacture a batch in a process cell the standard proposes a gradually refinement of the process model based on four recipe types; general recipe, site recipe, master recipe and control recipe. A recipe contains administrative information, formula information, requirements on the equipment needed, and the procedure that defines how the recipe should be produced. The procedure is organized according to the procedural control model, see Chapter 7.3 (Procedural control).

- General recipe
  The general recipe is an enterprise level recipe that serves as a basis for the other recipes. The general recipe is created without specific knowledge of the process cell equipment that will be used to manufacture the product.
- Site recipe
  The site recipe is specific to a particular site. The language in which it is written, the units of measurements, and the raw materials are adjusted to the site.
- Master recipe
  The master recipe is targeted to a specific process cell. A master recipe is either derived from a general recipe or created as a stand-alone entity by people that have all the information that otherwise would have been included in the general or the site recipe.
- Control recipe
  The control recipe is originally a copy of the master recipe which has been completed and/or modified with scheduling, operational and equipment information. A control recipe can be viewed as an instantiation of a master recipe.

The four recipes are gradually refined to the stage where all necessary aspects for the execution of the recipe on a certain type of equipment are taken into account. The general and site recipes are equipment independent whereas the master and control recipes are equipment dependent. In order to distinguish between equipment independent

and equipment dependent process steps different terminology is used. The terms Procedure, Unit Procedure, Operation, and Phases are introduced for the equipment dependent process steps, see Figure 7.1.

Procedural control

The four recipe levels together with the equipment control constitute the link between the process model and the physical model, denoted procedural control, see Figure 7.1. Procedural control is characteristic for batch processes. It directs equipment-oriented actions to take place in an ordered sequence in order to carry out a process-oriented task.

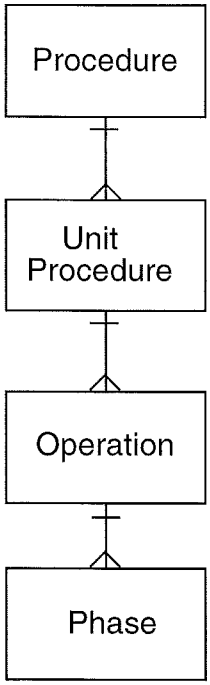


Figure 7.2 Procedure Control.

The procedural structure is hierarchical, see Figure 7.2. A procedure can gradually be broken down into smaller parts.

- The procedure is the highest level in the hierarchy and defines the strategy for accomplishing a major processing action such as making a batch. It is defined in terms of unit procedures, and/or operations, and/or phases. An example of a procedure is "Make a batch of product A".

- A unit procedure defines a set of related operations that causes a production sequence to take place within a unit. Examples of unit procedures are, e.g., polymerize, recover or dry. A unit procedure must be executed within a single unit process.
- An operation is a sequence of phases that defines a major processing sequence that takes the material being processed from one state to another. An operation usually involves a chemical or physical change. Examples of operations are, e.g., reaction and preparation.
- The smallest element that can accomplish a process oriented task is a phase. It defines a product independent processing sequence. A phase may be decomposed into steps and transitions according to Grafcet/SFC. Examples of phases are, e.g., add catalyst.

The IEC 1131-3 standard, which was published in 1993, specifies programming languages for controllers. This standard fills an important void since part 1 of ISA-S88 does not specify languages for configuring the sequential and batch control functions. Many suppliers have already incorporated the IEC 1131-3 standard in their products.

Sequential Function Charts (SFC) are gaining acceptance for configuration of the procedural part of recipes. The main reasons for this are that SFC are graphical, easy to configure and easy to understand. SFC is also the basis for Procedural Function Charts (PFC), a graphical language for recipe representation currently being defined in the SP88 working group.

Equipment control

The control recipe itself does not contain enough information to operate a process cell. On some level it must be linked to the process equipment control, i.e., the control actions that are associated with the different equipment objects. SP88 offers large flexibility with respect to at which level the control recipe should reference the equipment control. It is also allowed to omit one or more of the levels in the procedural model. The situation is shown in Figure 7.3. The dashed levels could either be contained in the control recipe or in the equipment control. Several examples of how this can be done will be given in Chapter 9.

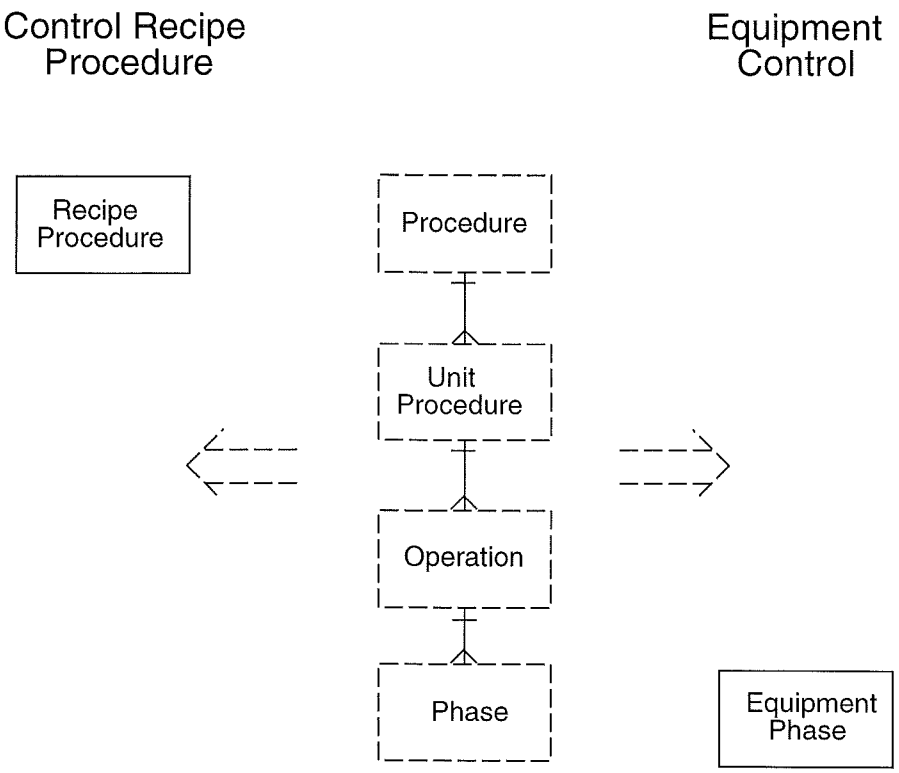


Figure 7.3 Control recipe/ Equipment control separation.

Control Activity Model

To successfully manage batch production many control functions must be implemented. The control activity model shown in Figure 7.4, identifies the major batch control activities and the relationships amongst them. This model was outlined in SP88 [SP88, 1995], and provides an overall perspective on batch control.

The control activities shown relate to real needs in a batch manufacturing environment.

- Recipe Management
  The need to have control functions that can manage general, site, and master recipes implies a need for the recipe management control activity.
- Production Planning and Scheduling
  Production of batches must occur within a time domain that is planned and subsequently carried out. Production planning and

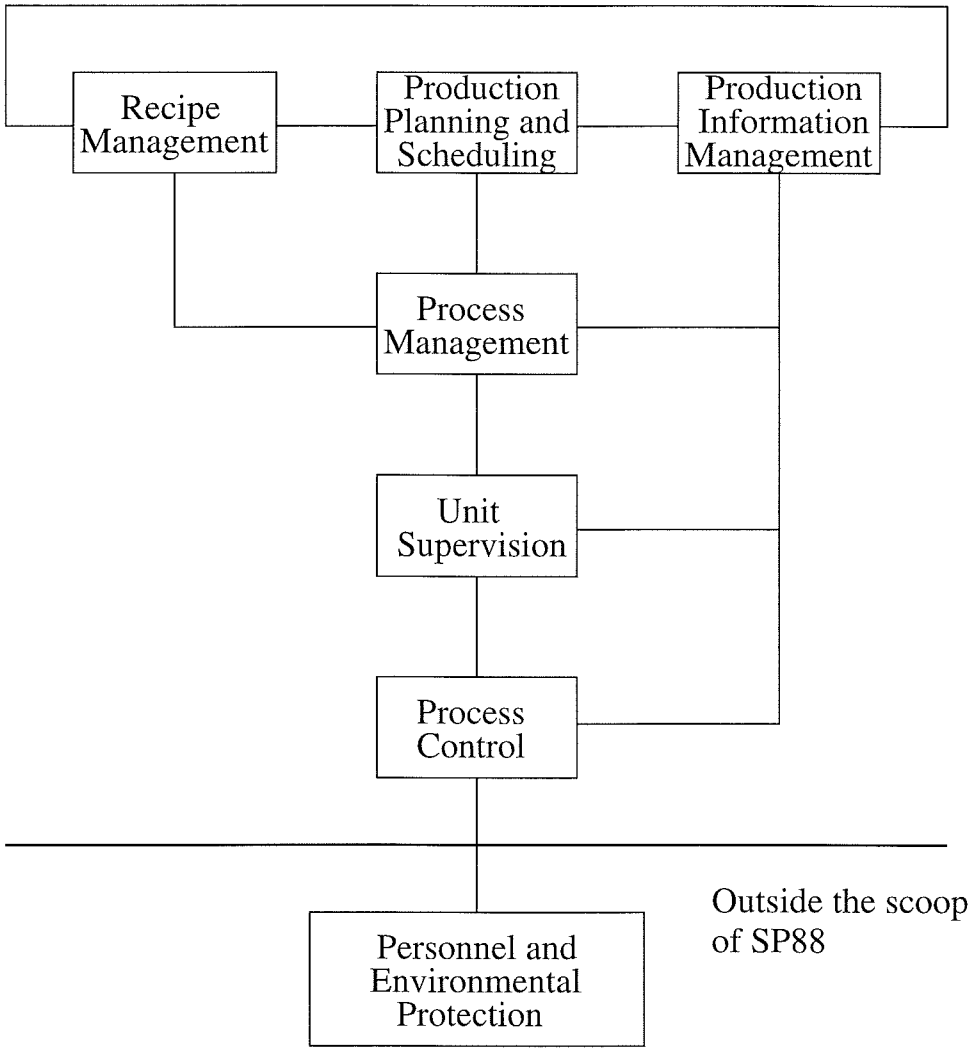


Figure 7.4 The control activity model of SP88.

scheduling is the control activity where these control functions are implemented.

- Production Information Management
  Various types of production information must be available and the collection and storage of batch histories is a necessity. The production information management control activity in the model covers these control functions.

- Process Management
  Control recipes must be generated, batches must be initiated and supervised, unit activities require coordination and logs and re-

ports must be generated. These control functions fall under the process management control activity.

- Unit Supervision
  The need to allocate resources, to supervise the execution of operation and phases, and to coordinate activities taking place at the process control level are examples of control functions needed at the unit supervision control activity level.
- Process Control
  The Process Control control activity discusses control functions that deal directly with equipment actions.

7.4 Summary

Industrial manufacturing and production processes can be classified as continuous, discrete or batch. Traditionally, continuous and discrete processes have been the areas where most research has been performed. However, increased demands on production flexibility, small scale production and customer-driven production has led to an increased interest for batch processes and batch control. A batch process is neither continuous nor discrete, yet it has characteristics from both. A batch cell can be made highly flexible, both with respect to the number of different products it can produce and with respect to the structure of the plant. The multi-product, network structured batch cell is the most flexible but also the most difficult type of plant to control. The recent batch control standard ISA-S88.01, formally defines the terminology, models and functionality of batch control systems. The standard mentions the possibility to use Grafcet but only at the very lowest level of control. However, there is also a need for a common representation format, complying with the standard, at the recipe level.

8

High-Level Grafchart and Batch Control

The SP88 standard is an important step towards a formal definition of the terminology, models and functionality of batch control systems for multi-product batch processes. The standard defines, e.g., the procedural structure of a recipe. The structure is hierarchical. Sequential Function Charts (SFC) is used for representing sequence control in the PLC standards IEC 848 and IEC 1131. Although the possibility to use SFC is mentioned in SP88, it is only presented as a formalism that may be used at the very lowest level, i.e, the PLC level, of the recipes. The standard mention nothing about the implementation of the higher procedural levels in the recipe.

The aim of this thesis is to shown various ways in which the concepts of High-Level Grafchart can be used in the context of recipe-based batch control. The aim is not to present a complete batch control system. As will be shown High-Level Grafchart can be used at all levels in the hierarchical procedure model, from the PLC level sequence control to the representation of entire recipes. High-Level Grafchart also facilitates the linking that has to exist between the control recipe procedural elements and the equipment control procedural elements.

A batch scenario, consisting of one batch cell, has been defined and implemented in G2. The batch scenario is used as a test platform to investigate how High-Level Grafchart can be used in a batch control system.

This chapter starts with a presentation of related activities. The common point of the activities presented, is that they are all based on Petri Nets or Grafcet. The chapter also contains a presentation of how High-Level Grafchart can be used in a batch control system and, it contains a description of the defined batch scenario.

8.1 Related Activities

Batch processes receive increasing interest in industry as well as the academic control community. This can be noticed by the increased number of researchers in the field. The control of batch processes can be attacked in various ways. In this section an overview of related work and activities, all with Petri Nets or Grafcet as a basis, is presented. The presentation does not try to cover the field completely, other activities, not presented in this section, but yet interesting, most likely exist.

By modeling batch plants and recipes mathematically and by applying design methods to these models control laws can be generated. The generated control laws should ensure safe and correct operations. The PhD-thesis by Tittus, [Tittus, 1995], proposes both models, frameworks and design methods for this task.

The plant is modeled by a Petri Net describing all physically possible connections between the process units. Each process unit is modeled by an individual PN presenting the state of this unit. The recipe for a batch, is modeled by a PN describing all possible execution paths through the plant. If more than one batch is to be produced at the same time, the transitions of the Petri Nets of all recipes are interleaved and one PN describing all possible intermixings of the recipes is achieved. This net is synchronized with that of the plant, forming the control recipe (Note: the terminology is not the same as that of S88.01). The control recipe is reduced to all physically possible intermixings of the recipes and the plant by considering the fact that each resource can only be used by one recipe at a time. Further reduction can be done by removing the places and transitions that can lead to an unsafe or a deadlock situation. From the reduced control recipe, a

discrete supervisor, that guarantees correct and parallel execution of the recipes, can be generated.

The work by Tittus differ from that of this thesis in that it is more focused upon formal verification and synthesis, see Chapter 1. The aim is to generate a safe control recipe and no effort is put into the representation or implementation of the recipes.

Wöllhaf and Engell have developed an object-oriented tool for modeling and simulation of the production process in a recipe-driven multipurpose batch plant, [Engell and Wöllhaf, 1994]. Models of the production plant, the recipes and the batches of material are developed. Both continuous and discrete aspects of the simulation are included in order to support the supervision of the plant and the scheduling production tasks. The work follows the German NAMUR standard, [NAMUR, 1992], see Chapter 7.2. The control recipe, represented with Sequential Function Charts, essentially contains the discrete model which describes the production steps and the transitions. The plant model together with the batches of material, constitute the continuous system. By substituting the basic functions of the recipe by the technical functions of the plant, a hybrid system, possible to use for simulations, is created.

The focus of the work by Wöllhaf and Engell is modeling and simulation. They focus upon the mathematical algorithms used to solve the differential and algebraic equations for simulation of the materials. A large database for physical and chemical properties of the most common substances is included in their tool. However, they do not focus upon the recipe management system and the issue of making the recipes reusable and flexible.

The aim of the work by Hanisch and Fleck is to join the theoretic work on resource allocation problems with the need of such strategies in recipe-control framework (recipe-based control systems), [Hanisch and Fleck, 1996]. They use high-level Petri Nets to model both the recipe-based operations and the resource allocation strategy. The resource allocation module is implemented separately within the process control system. The strategy of the resource allocation module is to optimize the productivity. Simulation of the system allows the control engineer

to answer questions about the utilization of resources, bottlenecks and resource dimensions.

Not many tools nor methods exist that allows to verify the recipe before it is used to control a chemical process. Hazardous processes exist and motivate the need for such methods. The idea of the approach of Brettschneider, Genrich and Hanisch, [Brettschmeider *et al.*, 1996], is to model the recipe, the plant and the device control (equipment control) by means of high-level Petri Nets. By merging the nets, verification and performance analysis can be performed using the analysis methods of Petri Nets.

8.2 High-Level Grafchart for Batch Control

The physical and procedural models of SP88, as well as the different recipe types, can all be nicely represented in High-Level Grafchart and the G2 environment.

Physical Model

The physical model of SP88 defines the hierarchical relationships between the physical units involved in batch control. Only the lower four levels will be considered as shown in Figure 8.1 (left).

The hierarchical structure of the physical model is straightforward to implement in an object-oriented environment such as G2. The attributes of a G2 object can either be values (numbers, strings, or symbols) or other objects. The latter case gives a hierarchical object structure that well matches the physical model. A G2 object representing an equipment unit may have attributes that contain other G2 objects representing the equipment modules in the unit. The equipment modules may have attributes containing the control modules and other equipment modules within the module. The equipment unit object may itself be a part of an object representing the batch cell. The situation is shown in Figure 8.1 (right).

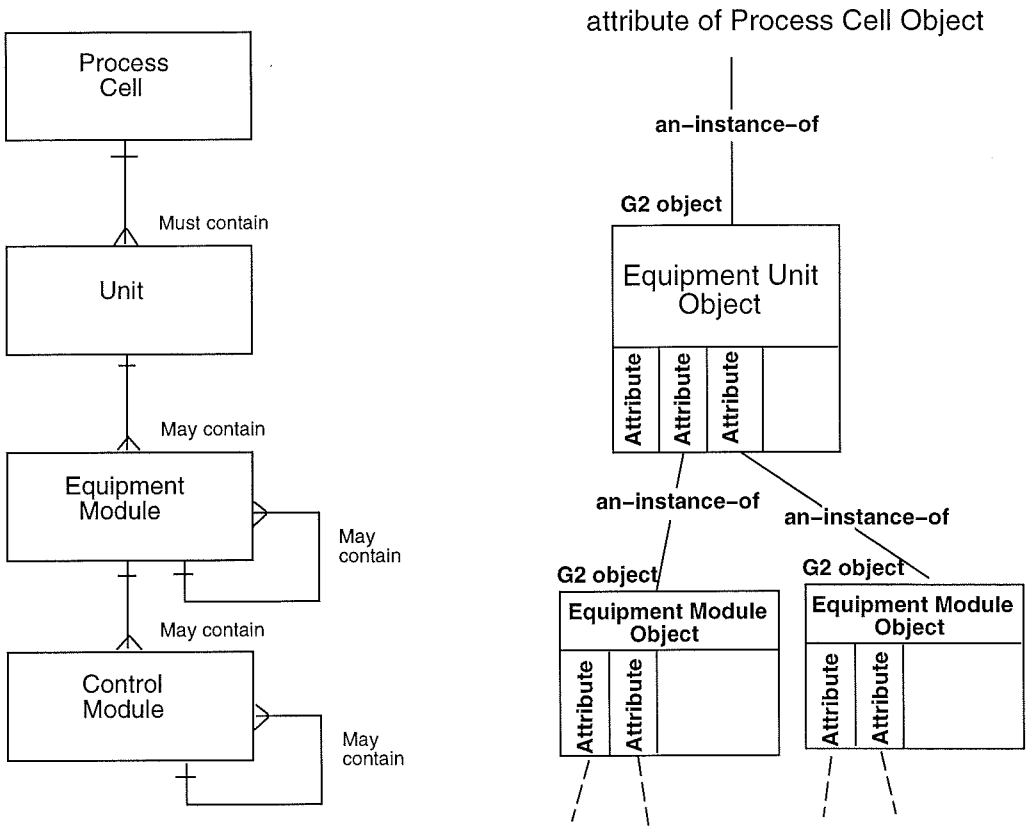


Figure 8.1 SP88 Physical Model (left) and its G2 representation (right).

Procedural model

The procedural model of SP88 is also hierarchical, see Chapter 7.3 (Procedural control). The structure of the procedural model is shown in Figure 8.2 (left).

The hierarchical levels of the procedural model can conveniently be described with the hierarchical abstraction facilities of Grafchart. A procedure can be represented as a function chart composed of macro steps representing the unit procedures. These macro steps may contain other macro steps or procedure steps representing the operations. Similarly, the macro steps representing the operations contain other macro steps representing the phases. Finally, the phase macro steps contain ordinary steps with associated step actions. The situation in shown in Figure 8.2 (right).

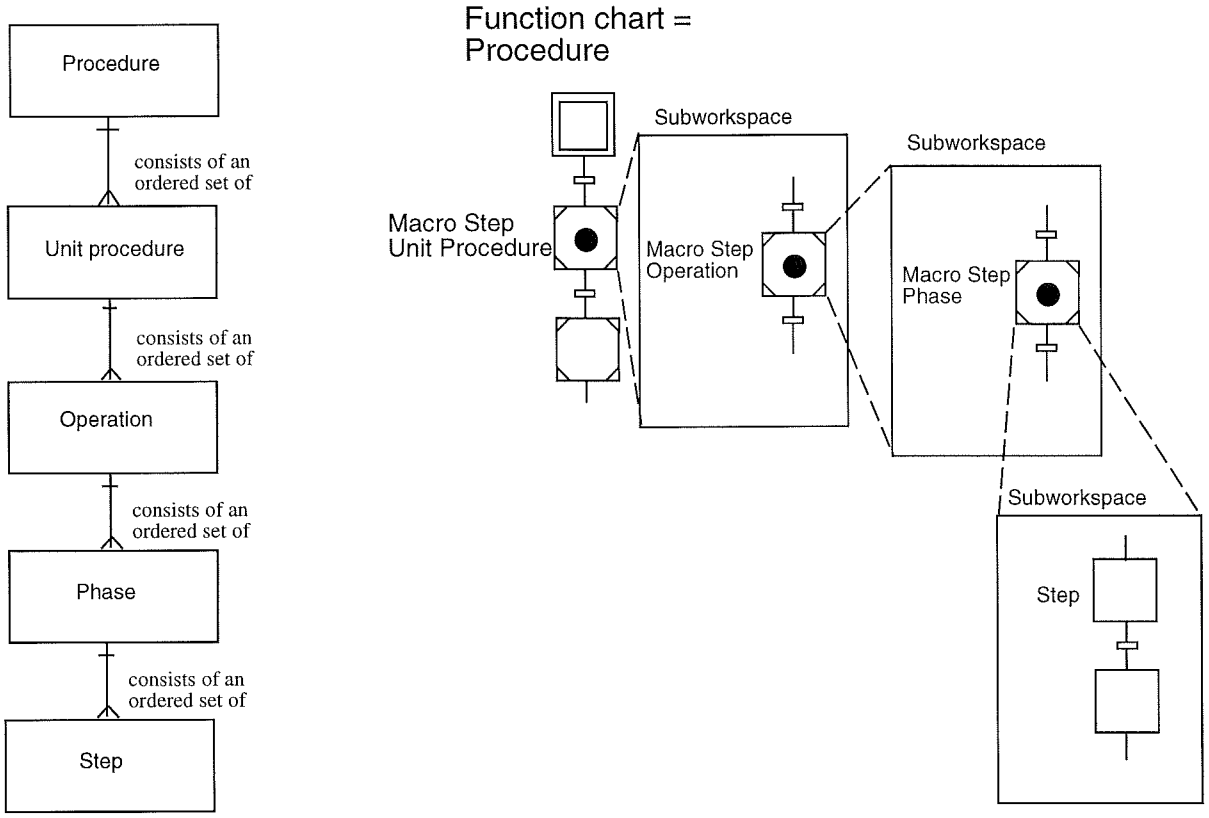


Figure 8.2 SP88 Procedural Model (left) and its representation in Grafchart (right).

Recipes

SP88 defines four recipe types: general recipes, site recipes, master recipes and control recipes. Here, only the master and control recipes are considered. A recipe contains administrative information, formula information, requirements on the equipment needed to produce the recipe and the procedure that defines how the recipe should be produced. The procedure is organized according to the procedural control model. The master recipe is targeted to a process cell. Each individual batch is represented by its control recipe. The control recipe is originally a copy of the master recipe which has been completed and/or modified with scheduling, operational and equipment information. A control recipe can be viewed as an instantiation of a master recipe.

With HL-Grafchart, recipes can be represented in two main ways: as function chart objects or as object tokens. In the first case a recipe is represented as a function chart with parameters (attributes). The

procedure part of the recipe is represented by the function chart and the formula information and equipment requirements are represented by parameters (attributes) of the Grafchart process encapsulating the function chart. This information can be accessed from the recipe procedure using the `sup.attribute` notation. This way of representing a recipe is shown in Figure 8.3 (left). In the second case a recipe is represented as an object token. This object token contains the formula information and equipment requirements as attributes and the recipe procedure as a Grafchart method, see Figure 8.3 (right). The procedure method can access the formula information and equipment requirements using the `self.attribute` notation.

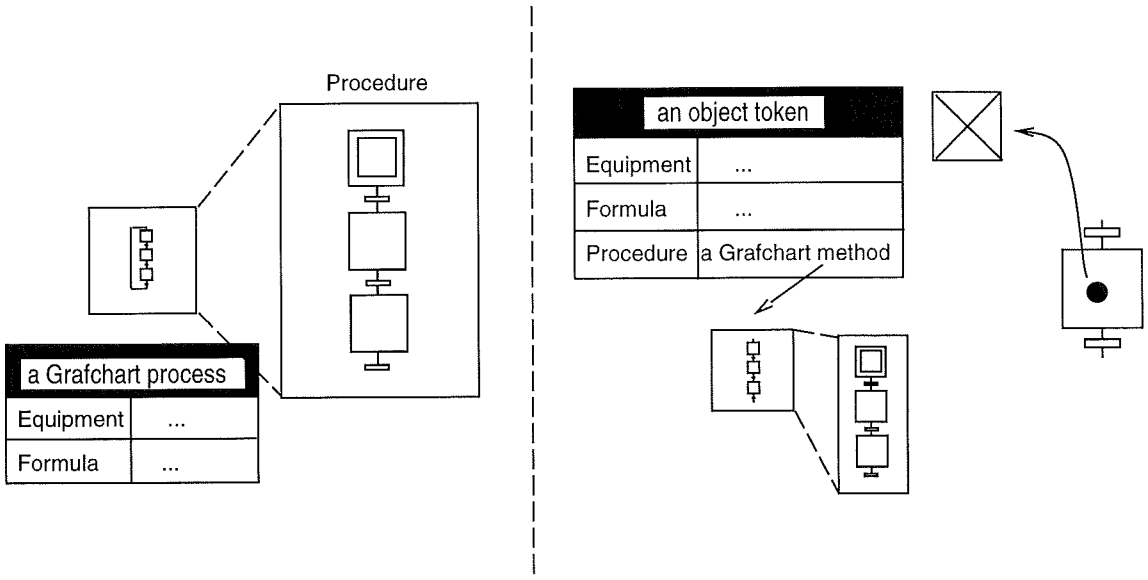


Figure 8.3 Control recipe as a function chart (left) and as an object token (right).

Recipe and Equipment control Separation

The control recipe does not contain enough information to operate the process cell. At some level it must be linked to the equipment control that is responsible for the actual operation of the process equipment. The separation between the control recipe procedure and the equipment control is illustrated in Figure 7.3. The dashed levels could either be contained in the control recipe or in the equipment levels. The linking could either be done on the procedure level, unit procedure level, operation level, or phase level.

Using High-Level Grafchart the linking is implemented using methods and message passing according to Figure 8.4. The element in the control recipe where the linking should take place is represented by a procedure step. Depending on at which level the linking take place, this procedure step could represent a recipe procedure, recipe unit procedure, recipe operation or recipe phase. This procedure step calls the corresponding equipment control element which is stored as a Grafchart method in the corresponding equipment object. If the linking takes place at the unit procedure level then the equipment control element corresponds to a equipment unit procedure. If the linking is done at the operation level the equipment control element is an equipment operation.

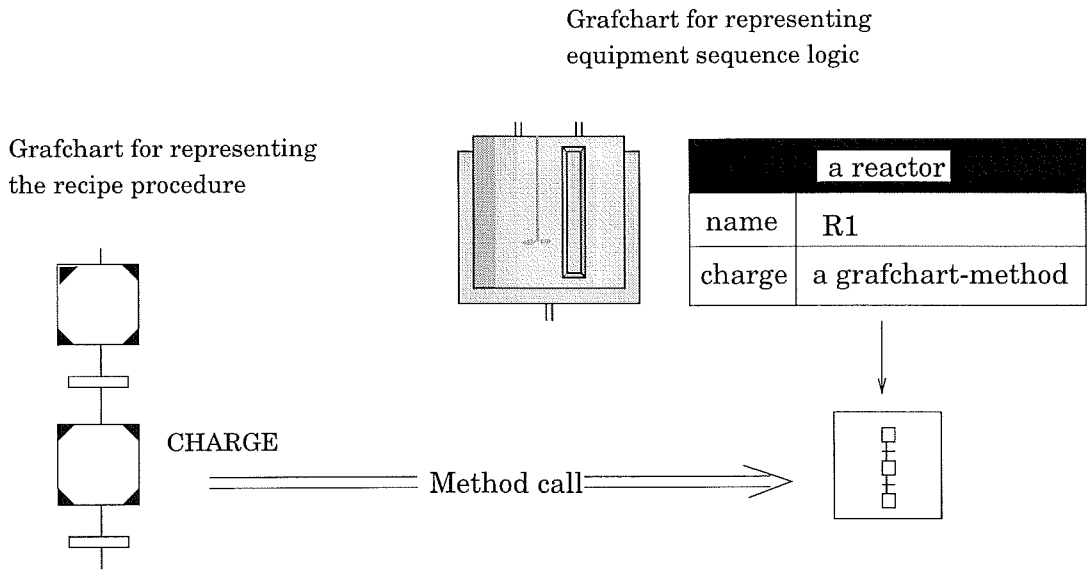


Figure 8.4 Control recipe/Equipment Control linking.

8.3 A Batch Scenario Implemented in G2

A batch scenario consisting of one batch cell has been defined and implemented in G2. The batch scenario is used as a test platform to investigate how High-Level Grafchart can be used for recipe handling.

In the batch cell, different products can be made. The batch cell is structured as a network, which means that for each batch that will

be produced there are several possible ways through the plant. It is possible to have several batches in the plant at the same time, the batches may be of the same or of different types. The batch cell is thus of the multi-product, network-structure type, see Chapter 7.

The cell consists of three raw material storage tanks, two mixers, three buffers, two batch reactors, and three product storage tanks. The units are interconnected through valve batteries. The cell can produce two products named D and E using three reactants A, B and C. A schematic of the batch cell is shown in Figure 8.5.

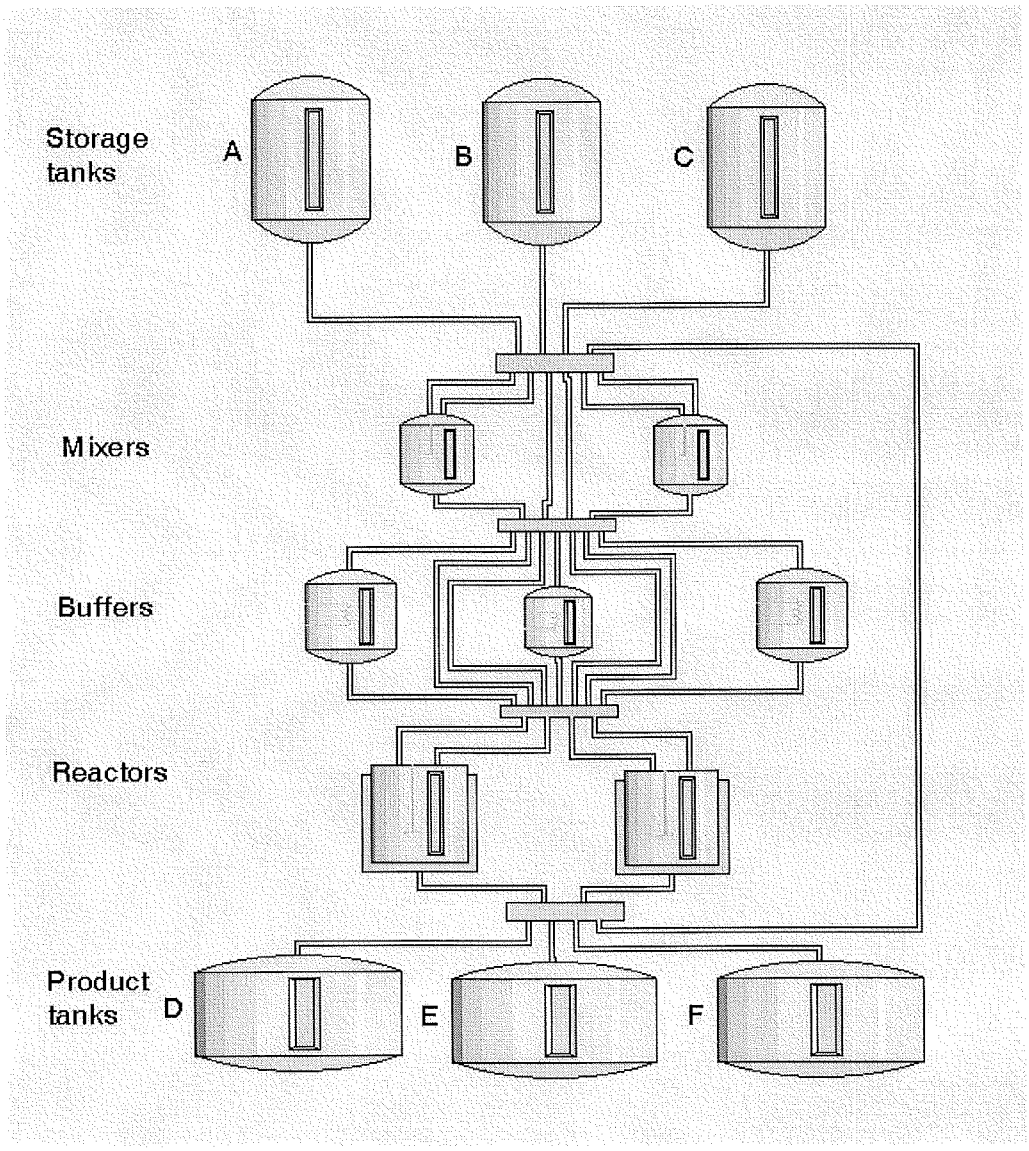


Figure 8.5 The batch cell.

The scenario is divided into three major parts: a dynamic simulator that simulates the dynamics of the scenario, a process control system and an operator interface. Figure 8.6 shows the three parts of the batch cell.

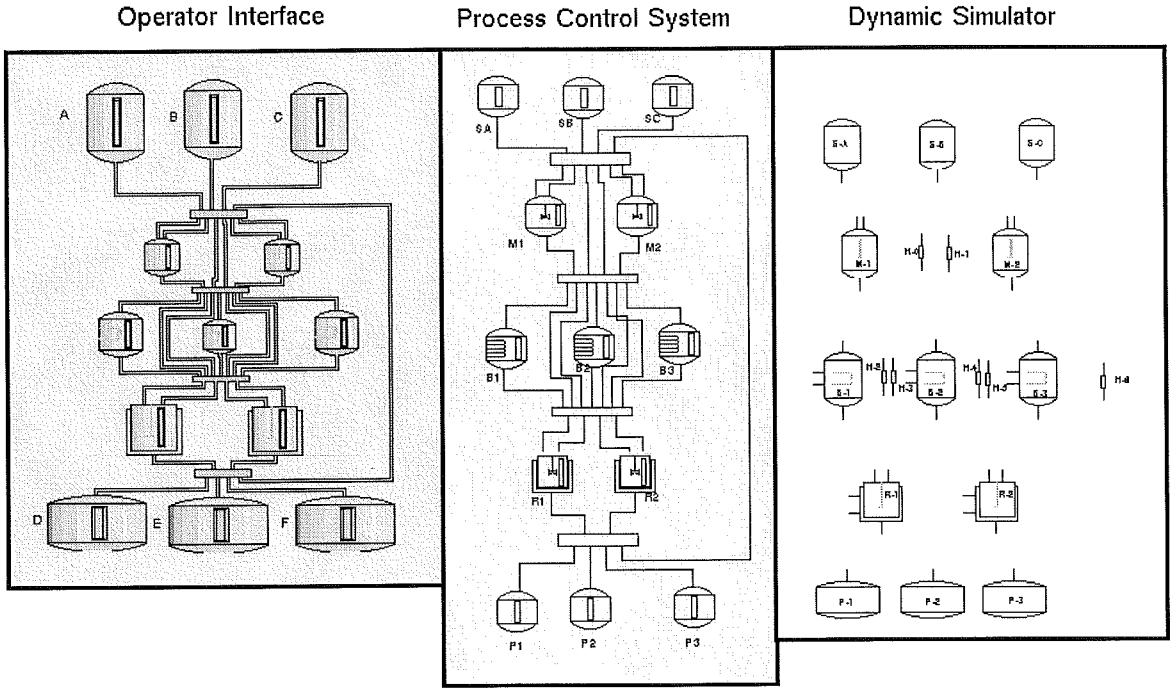


Figure 8.6 The three parts of the batch scenario.

Unit processes

Storage tanks The storage tanks contain the raw materials, i.e the reactants. Since there are three different reactants there are three storage tanks in the cell, one for each reactant. Each storage tank has a level-transmitter and a temperature-transmitter. Each storage tank also has an output on-off valve and pump connected to the outlet of the tank. The storage tanks are assumed never to be empty. The schematic of a storage tank is shown in Figure 8.7 (left).

Mixers The main task of a mixer is to mix different substances, but a mixer can also be used to store substances while waiting for some other unit. The mixer is equipped with an agitator. The mixer has two inlets, each controlled by an on-off valve. In this way it is possible

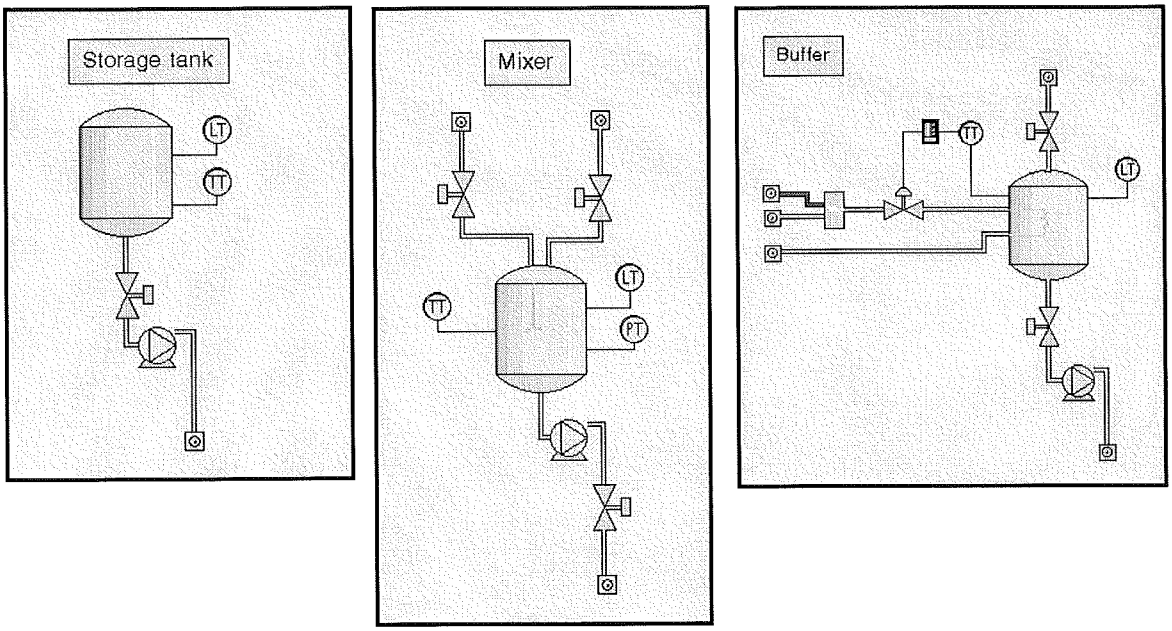


Figure 8.7 A storage tank (left). A mixer (middle). A buffer (right).

to fill the mixer in parallel. The outflow of a mixer is controlled by a pump and an on-off valve. The mixer has three transmitters: a level-transmitter, a temperature-transmitter and a pressure-transmitter. In Figure 8.7 (middle) a mixer is shown.

Buffers The task of the buffer is to store substances and if necessary maintain the temperature of the substance. A buffer can also be used to heat or cool a substance. The inflow of the buffer is controlled by an on-off valve and the outflow is controlled by a pump and an on-off valve. As shown in Figure 8.7 (right) a buffer has a level and a temperature transmitter. The temperature transmitter is connected to a PID-controller. The PID-controller adjust the flow of hot or cold liquid to the heater of the buffer and, thus, controls the temperature of the content of the buffer.

Reactors A reactor has two inlets and one outlet. It has an agitator and four transmitters: level, temperature, pressure and flow. The outflow is, as before, controlled by a pump and an on-off-valve. The inflow, and thereby the level in the reactor, is controlled by control valves using a PID-controller. A schematic of the reactor is given in Figure 8.8

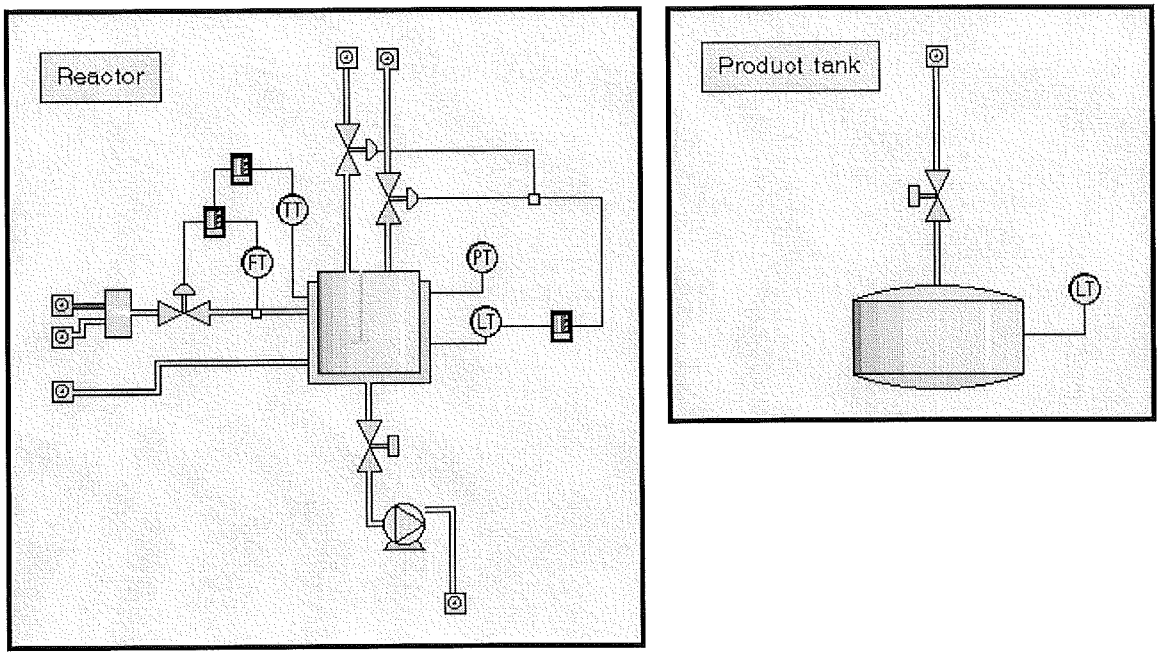


Figure 8.8 A reactor (left). A product tank (right).

(left).

The main task of the reactor is to perform chemical reactions. This is done by heating or cooling the substance inside the reactor. The reactor has a jacket which can be filled with a hot or cold liquid. The temperature in the reactor is controlled by two cascade coupled PID-controllers that adjust the flow through the jacket.

Product Tanks The product tanks are used for storing the finished products. They are assumed never to be full. Each product tank has an inlet valve and a level transmitter. A schematic of a product tank is shown in Figure 8.8 (right).

Valve-batteries Valve-batteries, see Figure 8.9, are placed in between the different units. A valve battery is organized in a matrix structure with `n` rows and `m` columns where `n` is the number of inlets and `m` is the number of outlets. By opening the valve placed in position `(i,j)`, inlet `i` will be connected to outlet `j`.

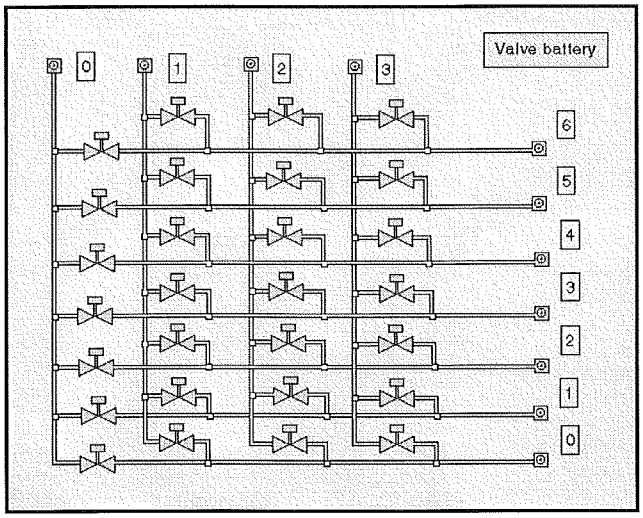


Figure 8.9 A valve-battery.

The Operator Interface

The operator interface, see Figure 8.5, provides the operator with a userfriendly interface, through which he or she can get information about the status of the plant. Through the interface the operator can also interact with the plant.

By clicking on any of the units the internal structure of this unit is shown. The operator can open and close the valves manually by clicking on them. The pumps can be turned on and off manually. Some of the units also have PID-controllers, these can be operated manually or automatically. When clicking on any of the sensors a trend-curve of the measured signal is shown, see Figure 8.10.

As the batch moves through the plant, its actual position is highlighted. A unit currently used by a batch is marked with a black dot placed at the upper right corner. The level indicator of this unit shows the actual level in the unit. If the batch is using a pipe, i.e. is on its way from one unit to an other, this pipe is marked with a colour.

The Dynamic Simulator

The dynamic simulator is used as a substitute for a real plant. It simulates the mass balances, energy balances and the chemical reactions. All simulations are done in real-time.

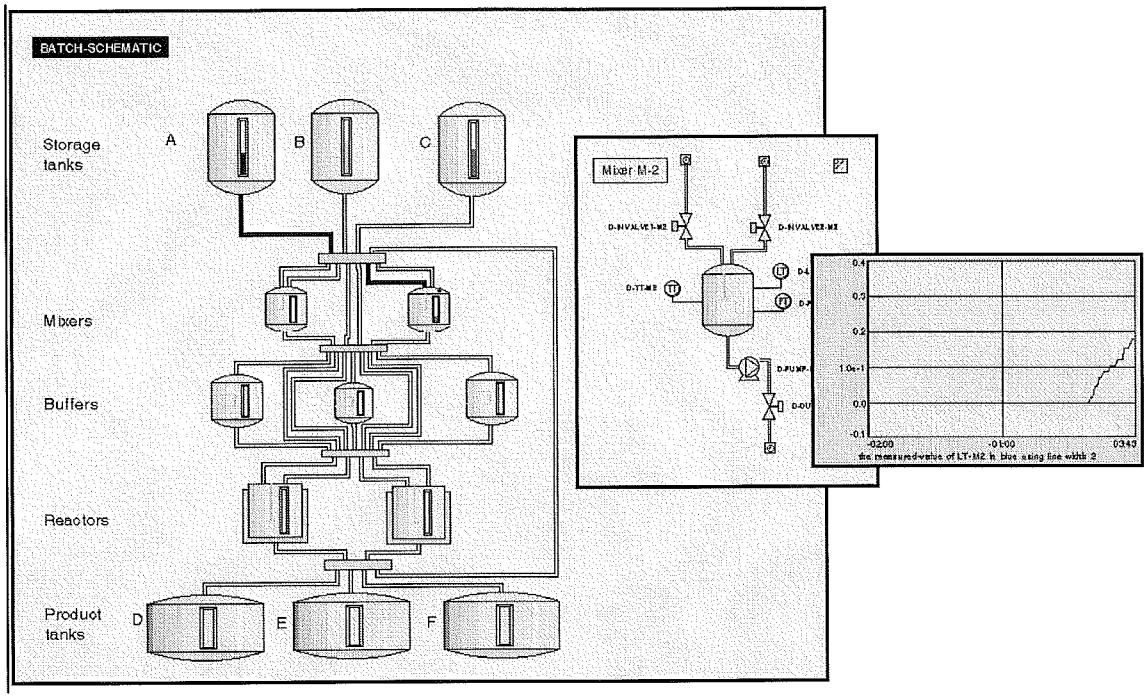


Figure 8.10 The operator interface showing the internal structure of a mixer and a trend curve.

Two reactions can be simulated in the scenario:

$$A + B \longrightarrow 2D$$
$$C + D \longrightarrow E$$

The total mass balance, the component mass balances and the energy balance for the reactions are calculated and simulated. A detailed description of the dynamic simulator is found in Appendix B.

The Process Control System

The process control system implements the basic control functions. It both receives information from and sends information to the simulator. For example, if the operator starts a pump this information must be propagated to the simulator and the pump should be turned on. Similarly, information about, e.g., the actual level in a unit should be sent from the simulator to the process control system.

The process control system contains equipment objects (unit processes) representing the units in the plant, i.e. the tanks, mixers, buffers and

reactors. The equipment objects have an internal structure of sensors, actuators and PID-controllers. These correspond to the equipment and control modules of SP88.

These objects are stored as attributes of the unit process objects, see Figure 8.11.

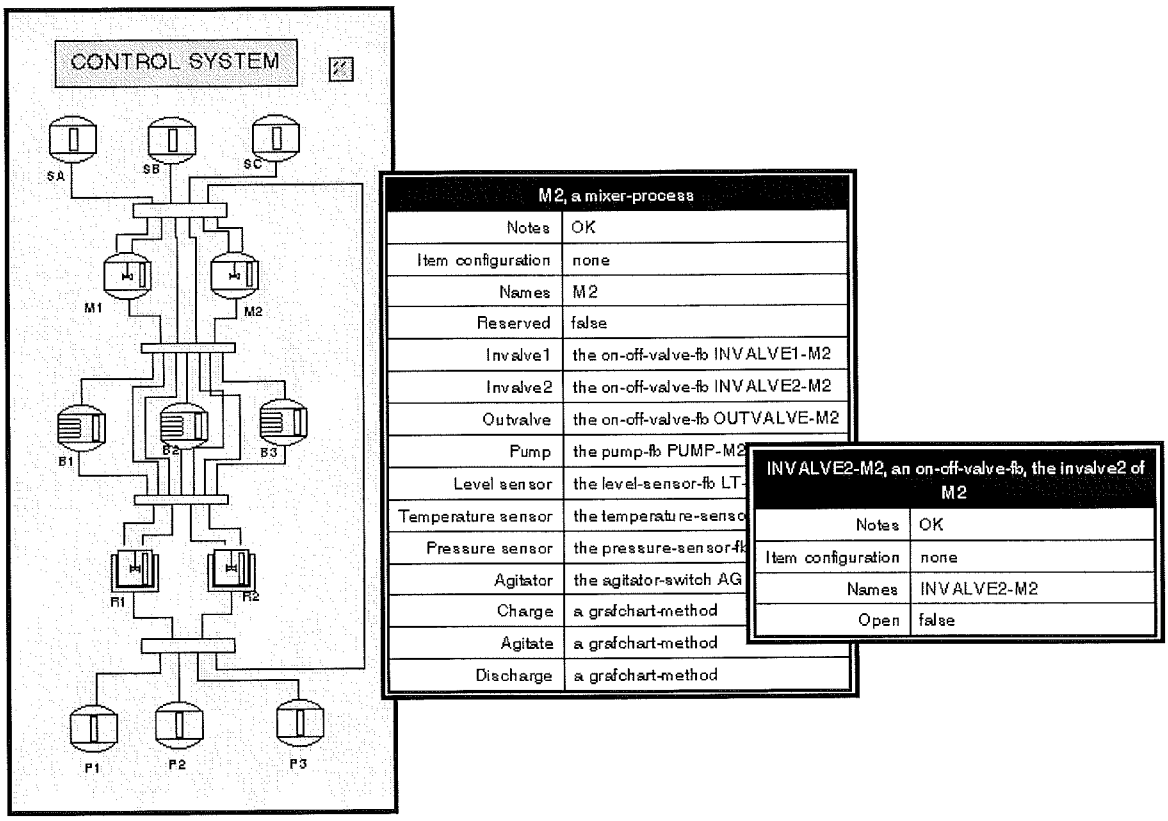


Figure 8.11 The process control system with its equipment objects and their internal structure.

To each unit process a number of methods are associated. These methods are descriptions of the operations or phases that the unit can perform, i.e., the methods are detailed descriptions of how to perform a certain task on one particular unit. The names of the methods are found in the attribute table of each object whereas the implementation of the different methods are found on the subworkspace of the class definition it belongs to. A mixer can, e.g., perform the following tasks: charge, agitate and discharge as shown in Figure 8.12. The methods are implemented as Grafchart methods.

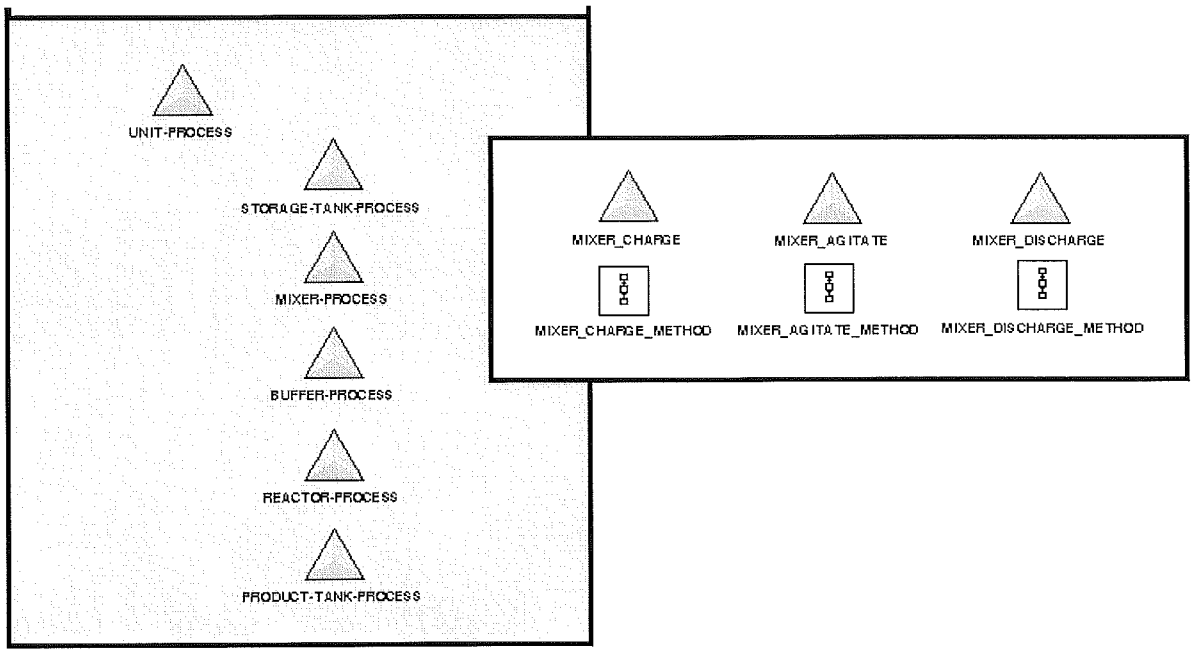


Figure 8.12 The methods of the class 'mixer-process'.

Recipes

Two reactions can be simulated in the scenario:

$$A + B \longrightarrow 2D$$
$$C + D \longrightarrow E$$

Recipe $A + B \longrightarrow 2D$: First the same amount of the two reactants are filled into a mixer. The filling can be done either in parallel or in sequence. When the filling is completed the agitation is started. When the two reactants are well mixed the content of the mixer is transfered into a reactor. In the reactor the reaction takes place. This reaction needs to be heated for the reaction to start. Finally the reactor is emptied and the substance is stored in a product-tank.

Recipe $C + D \longrightarrow E$: The production of substance E can be done in either of two ways. Both start with producing substance D as described above.

- Substance D is transported from the reactor to a buffer and then reactant C is added. The content of the buffer is emptied into a

reactor and the reaction is started by heating the content. When the reaction is completed the reactor is emptied and the product is stored in a product-tank.

- Reactant C is filled into a buffer and heated to the same temperature as substance `D` at the same time as product D is being produced. Then the content of the buffer is transfered to the reactor and the two substances are agitated, and, if needed, further heated. When the reaction is completed, the product, i.e., substance E, is stored in a product-tank.

8.4 Summary

An object oriented language is well suited to represent the hierarchical structure of the physical model. High-Level Grafchart can be used to represent all the levels in the hierarchical procedural control model and, by using the support for methods and message passing, the linking between the control recipe procedural elements and the equipment procedural elements can be nicely represented.

9

Recipe Structuring using High-Level Grafchart

The main application of High-Level Grafchart is recipe structuring and resource management in multi-product, network-structured batch production cells. A number of different ways of how to structure the recipes and how to incorporate resource allocation have been investigated.

High-Level Grafchart is used at two levels, to represent the sequential structure in the recipes, i.e., the step by step instructions of how to go from raw material to product, and to represent the sequential control logic contained in the equipment units. To link the two parts together, method calls are used, as discussed in Chapter 8.2 (Recipe and Equipment control Separation).

In this chapter, four alternative representations of recipe structures are given. The structuring alternatives have large differences and they therefore have different advantages and disadvantages. Each major structuring alternative is presented in a separate section. Drawbacks and advantages are presented for each alternative. The first structuring alternatives, Section 9.1, use High-Level Grafchart version I whereas the reminding alternatives, Section 9.2, 9.3, and 9.4, use High-Level Grafchart version II. The two versions are described in Chapter 6.

In all structuring alternatives, a batch can be followed through the plant and it is possible to record its history in logs. This has, however, not yet been fully implemented and is therefore not described.

As a common example, the simple recipe for producing product D, see Chapter 8.3, will be used. The production is assumed to be performed in the process cell described in Chapter 8.3.

9.1 Control Recipes as Grafchart Function Charts

The alternatives given in this section use High-Level Grafchart version I, i.e., the Grafchart model is extended with parameterization and methods and message passing.

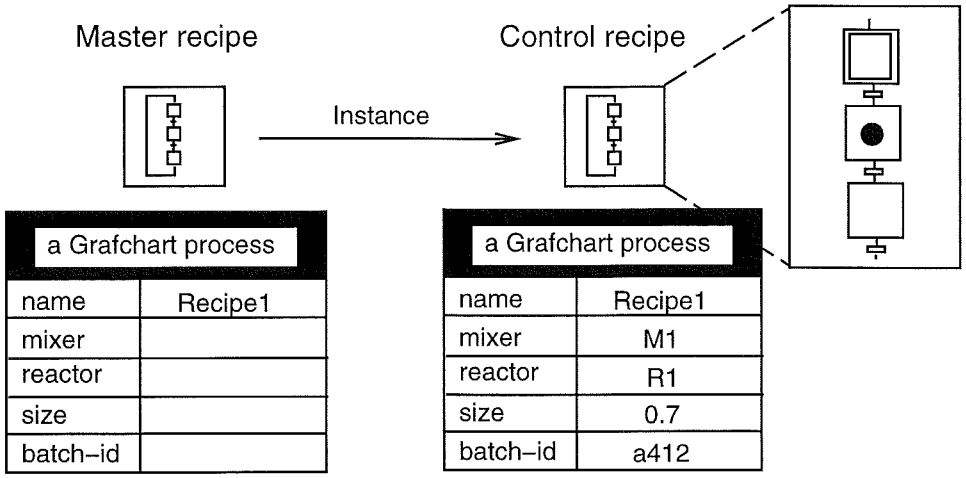


Figure 9.1 Master and control recipes.

A master recipe is represented by a Grafchart process. The Grafchart process has attributes specifying the type of equipment needed when producing the batch. The control recipe is a copy of the master recipe where the attributes are given their actual values, i.e., the equipment that will be used is specified. The situation is shown in Figure 9.1 .

From the steps and transitions within the control recipe the attributes can be referenced using the sup notation, see Chapter 6.1.

Recipes based on recipe phases

In the first example the control recipe procedure consists solely of phases. Each phase references the associated equipment phase through a method call according to Figure 9.2.

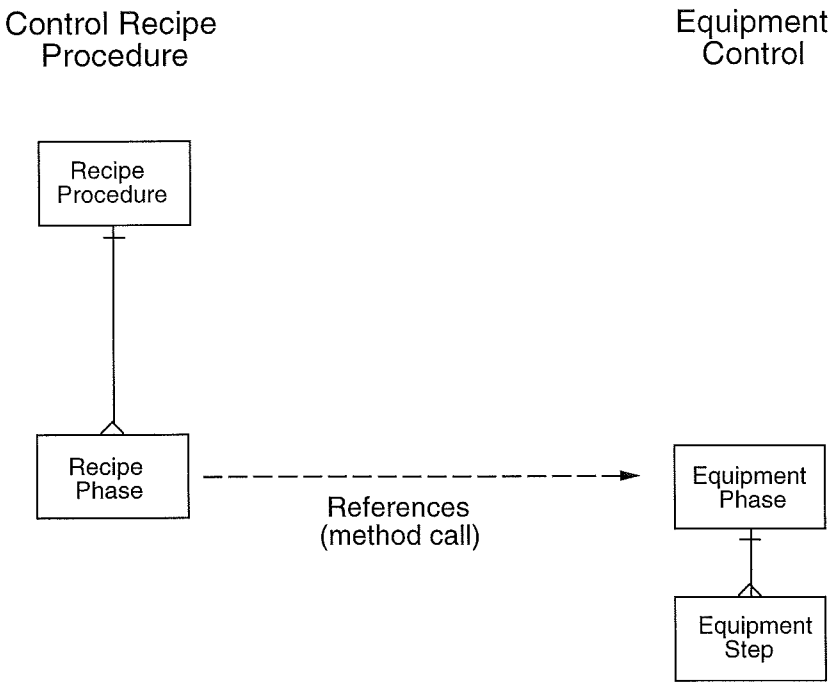


Figure 9.2 Recipes based on recipe phases.

The resulting control recipe is shown in Figure 9.3. The different unit processes that the batch passes through are indicated by the column-based organization of the recipe. The recipe starts by an assignment step in which the resources necessary to produce the recipe are allocated. Two version of the recipe have been implemented. One where the operator manually assigns all the equipment before starting the procedure and one where the equipment is assigned automatically through a simple search algorithm immediately before the equipment is needed. After the assignment step the two reactants `A` and `B` are filled into a mixer. This is done in parallel, since the mixers have two inlets. Then the agitation is started. When the agitation is finished the mixer is emptied and at the same time the reactor is charged. This is expressed with the Grafchart parallel construct. When the charging is finished, the reaction is started by heating the batch. Finally, the content of the reactor is transfered to the product tank and the production of the batch is completed.

Each recipe phase is represented by a procedure step that calls the corresponding equipment phase. In Figure 9.3 the attribute table for the agitate step is shown. The procedure step calls the agitate meth-

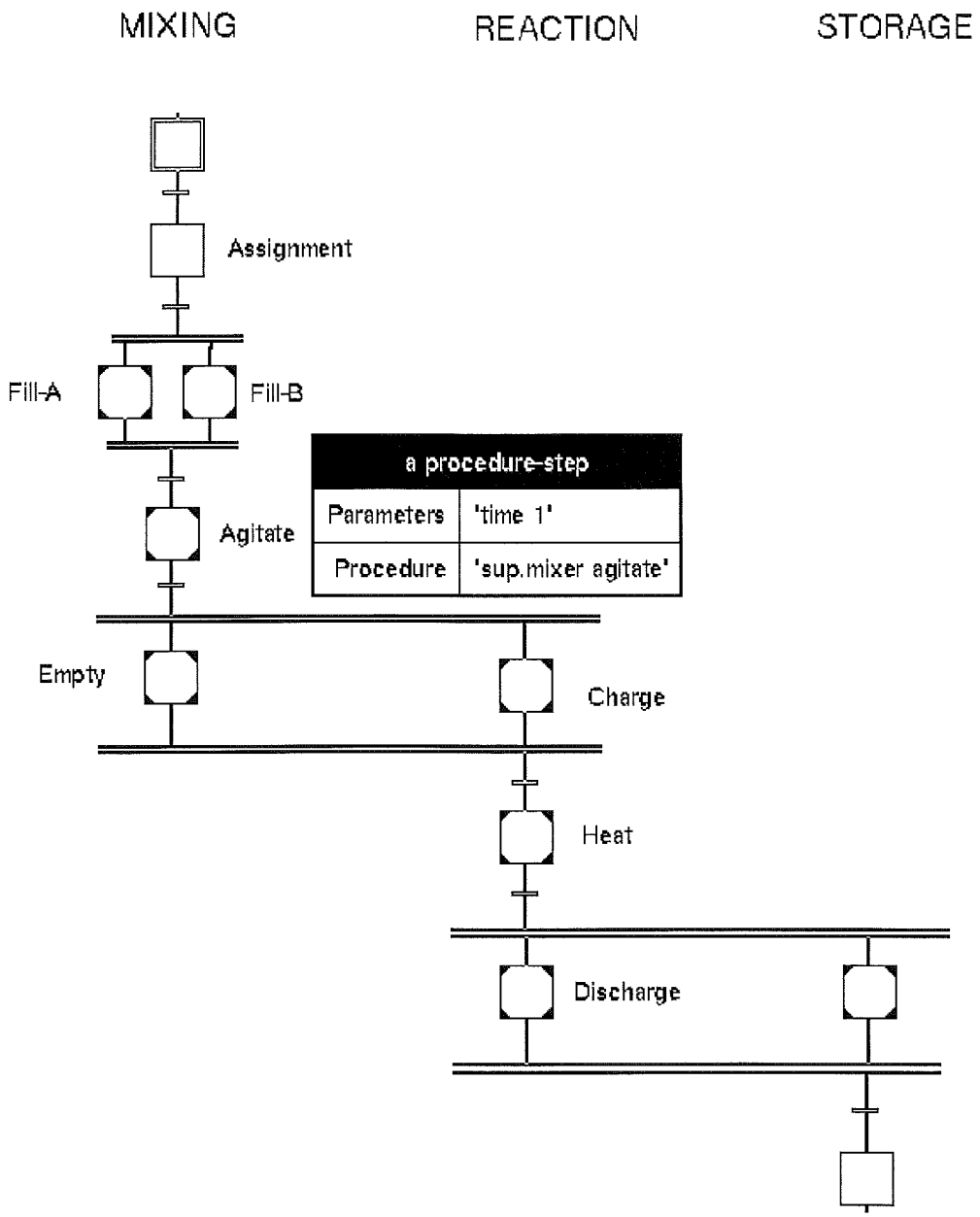


Figure 9.3 Control recipe based on recipe phases.

ods in the mixer that has been assigned to the batch. This mixer is contained in the `mixer` attribute of the recipe. The parameter `time` is given the value `1`. This denotes how long the agitation will take. The agitate equipment phase consists of two steps where the agitator motor is active until the defined time has elapsed. The situation is shown in Figure 9.4.

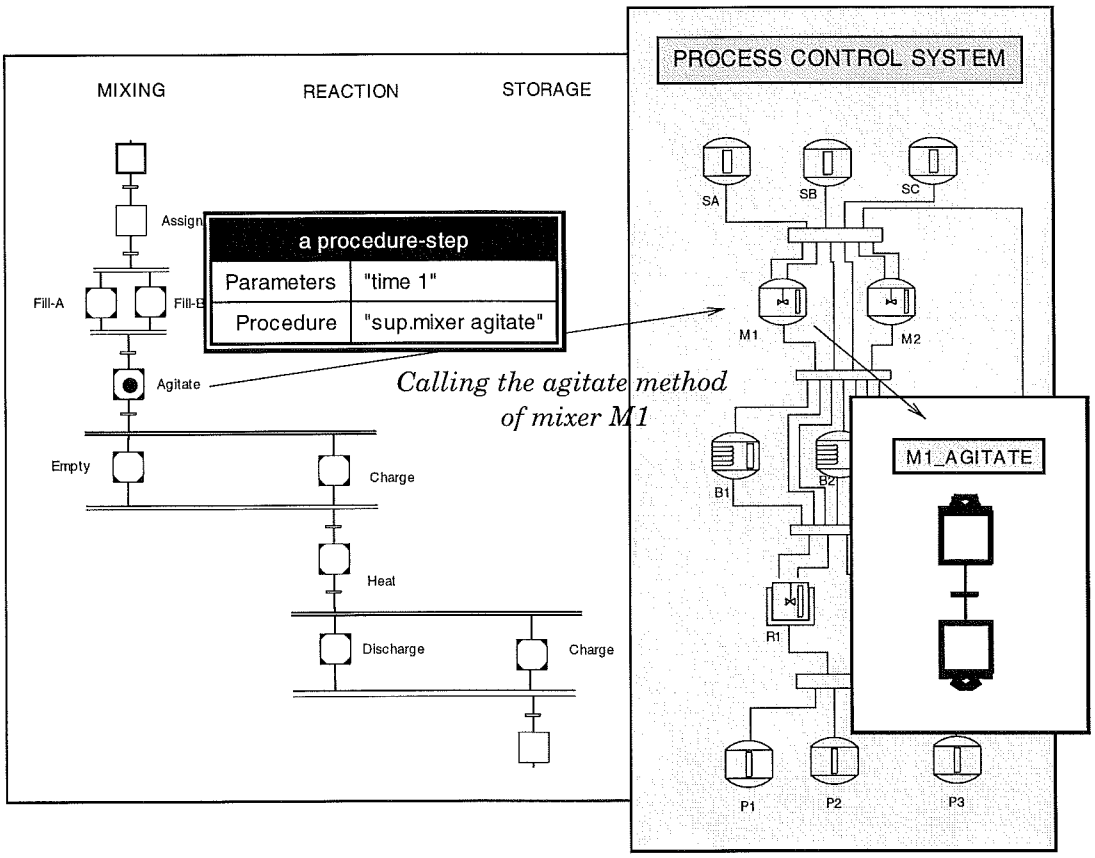


Figure 9.4 Recipe - Equipment linking.

Recipes based on recipe operations

In the second recipe structuring alternative the control recipe is structured into operations that internally are decomposed into recipe phases. The linking between the control recipe and the equipment control still takes place at the phase level. The structure is shown in Figure 9.5.

The resulting control recipe is shown in Figure 9.6. The control recipe is a straight sequence of recipe operations. Each recipe operation is represented as a macro step that contains the recipe phases. The phases are represented as procedure steps that call the corresponding equipment phases.

A problem with this approach is how to handle the parallelism between the operations. The emptying of the mixer and the charging of the reactor should be performed in parallel. Here this is solved using process

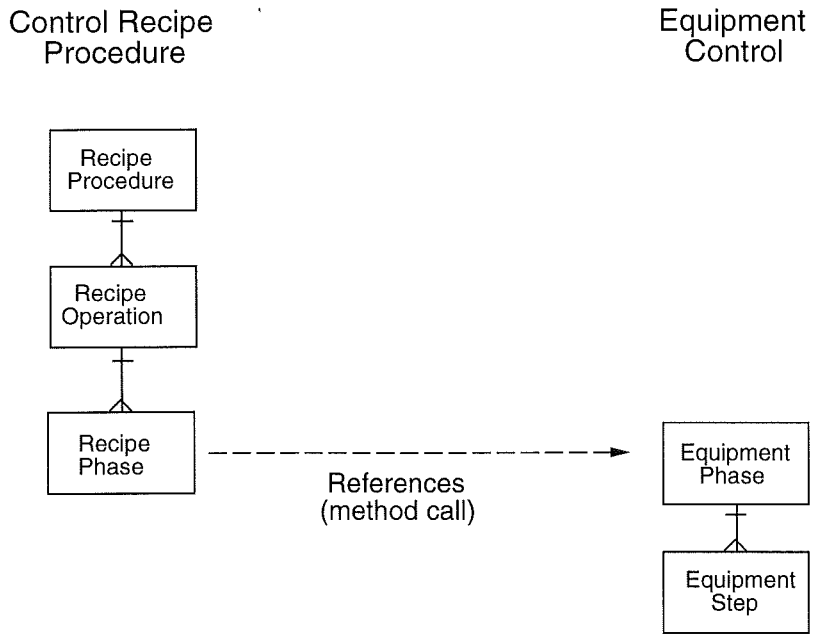


Figure 9.5 Recipes based on recipe operations.

steps. The `Empty` phase of the mixer operation is started through a process step, i.e., as a separate execution thread. The result will be that the `Empty` phase will execute at the same time as the `Charge` phase of the reaction operation. A drawback with this approach is that the parallelism is not expressed explicitly with the Grafchart constructs and therefore not directly visible to the operator as in the previous example. The recipe structure in Figure 9.6 could also be used if the linking was done at the operation level. In that case the equipment operations are internally structured as equipment phases.

Recipes based on equipment operations

In the third example the control recipe and the equipment control are linked at the operation level according to Figure 9.7.

In the previous examples, the control recipe has explicit information about which type of unit processes that the batch should be processed in, e.g., a mixer, buffer or reactor. Another possibility is to let the control recipe be completely independent from which unit type it should use. This can be achieved by defining a number of generic equipment independent operations. Examples of generic operations could be store, store-with-agitation, mix, mix-with-agitation, store-under-heating, ex-

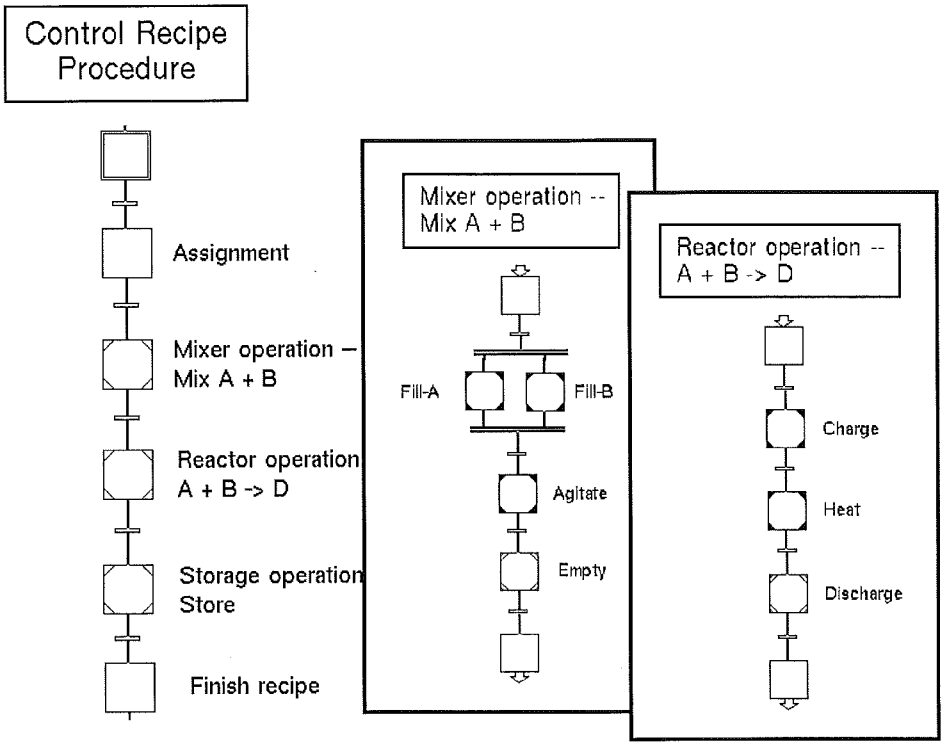


Figure 9.6 Control recipe based on recipe operations.

othermal-reaction etc. It is possible to mix two reactants in both mixers, buffers and reactors. However, it is preferred to mix reactants in a mixer. The mix-with-agitation operation cannot be performed in a buffer. The store-under-heating operation can be performed in both buffers and reactors but it is preferred to use the buffers for this. Hence, for each generic operation a preference ordering must be given. It is the task of the resource allocation scheme to find the best suited unit processes for each recipe operation and to assign this to the recipe when it is needed. The equipment operations are represented as equipment object methods. These methods calls the equipment phases which also are methods of the equipment objects. This structure has not been implemented and tested in the batch scenario.

Advantages and Disadvantages

Structuring a recipe management system with control recipes represented by Grafchart function charts, has advantages and drawbacks. The main advantage is that the system is easy to comprehend, one

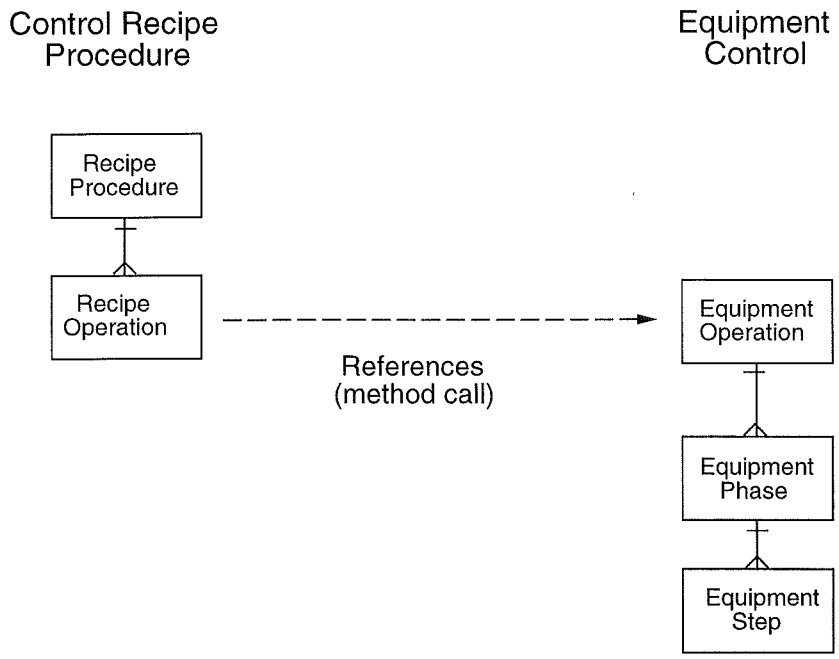


Figure 9.7 Recipes based on equipment operations.

batch equals one Grafchart process. Each control recipe, i.e., each Grafchart process can be modified independently of the other control recipes simply by changing the parameters of the procedure steps or by changing the function chart. However, the drawback is that it is not easy to get simultaneous information about all batches being produced in the cell.

9.2 Control Recipes as Object Tokens

The recipe structuring alternatives presented in this section use High-Level Grafchart version II, see Chapter 6. They utilize parameterization, method and message passing, and the object token extensions of HL-Grafchart.

In the previous section, one copy of the master recipe was needed for each batch that was produced. The reason for this is not that the batches are produced in different ways or that they use different types of equipment, but that they do not necessarily use the same equipment or have the same batch-identity number. Since the attributes, in the

last section, were global in the context of the function chart, the same control recipe cannot be used simultaneously by more than one batch. If, however, the batch specific information is stored locally, e.g., in the tokens, the same function chart can be used by more than one batch. Information that the batches share is represented globally, whereas batch specific information is represented locally. Since the token contains the batch specific information, the token represents the control recipe and the function chart that the token is reside in is the common master recipe. This way of structuring a recipe is shown in Figure 9.8.

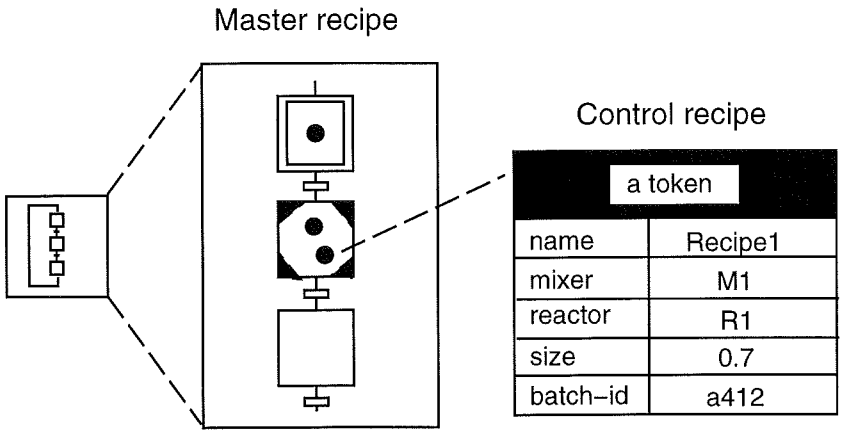


Figure 9.8 Control recipes as tokens.

The recipe alternatives presented in this section are all based on recipe phases and each phase calls the associated equipment phase through a method call. It is also possible to build up the recipe using operations instead of phases, in the same way as described in Chapter 9.1 (Recipes based on recipe operations).

The recipe shown in Figure 9.3 can be structured with object tokens. The attributes previously contained in the control recipe function chart will now be contained in the tokens. It is possible to have several tokens in the same master recipe. Each token corresponds to one control recipe, i.e., one batch.

Recipes extended with resource allocation

Using the object token structuring possibility it is possible to combine resource allocation with recipe execution in a Petri Net fashion, see Figure 9.9.

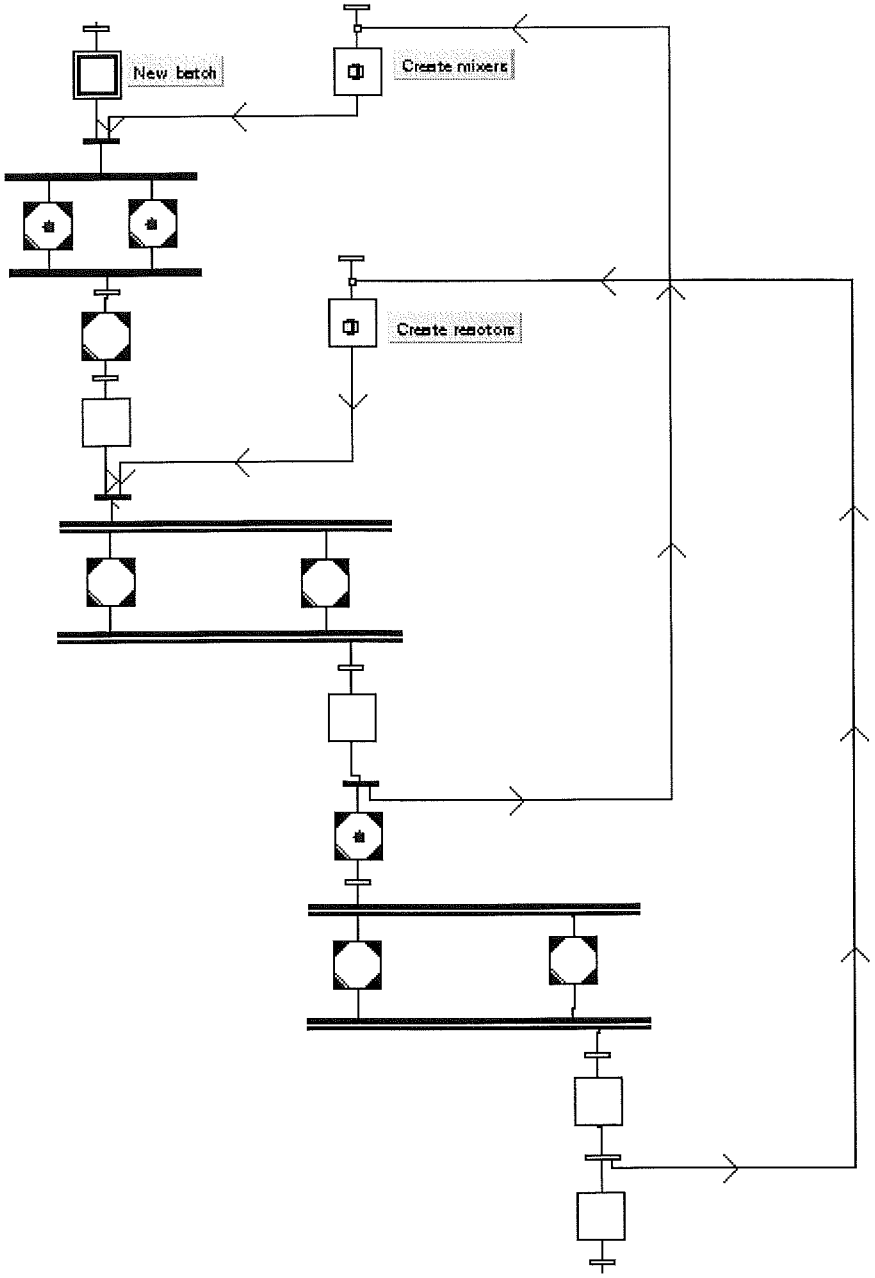


Figure 9.9 Master recipe with resource allocation.

In the recipe two batches are under production. One batch is currently filling reactant `A` and reactant `B` in one of the mixers and the other is heating the content in one of the reactors. At the moment there is thus only one mixer (out of two) and one reactor (out of two) reserved by a batch.

In this approach an available equipment unit is represented as a token residing in a step. The token correspond to a semaphore that guarantees a batch exclusive access to the resource (equipment unit). PN-transitions with multiple input and output arcs are used to express the resource synchronization. If a batch has specific resource requirements, these can be expressed and checked against the equipment capacity in the transition receptivity.

The chart can also be extended with more equipment oriented operations such as cleaning (CIP - clean in place). In Figure 9.10 an extremely simple master recipe is shown. Here, a CIP-operation must be performed on each unit after it has been used by a batch.

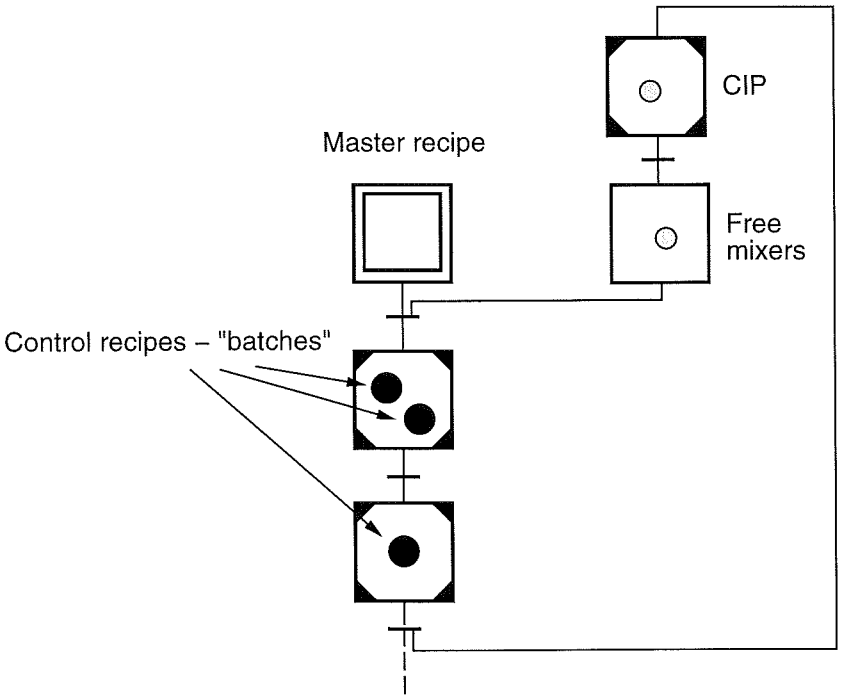


Figure 9.10 A master recipe with combined equipment oriented operations.

Combined network

Using the token object structuring possibility combined with resource allocation it is possible to merge all different master recipes into one single High-Level Grafchart. This chart visualizes the resources sharing that exists between the batches. The chart can be analyzed with Petri Nets analysis methods to detect if there is a risk for deadlock. The case for two simple master recipes is shown in Figure 9.11.

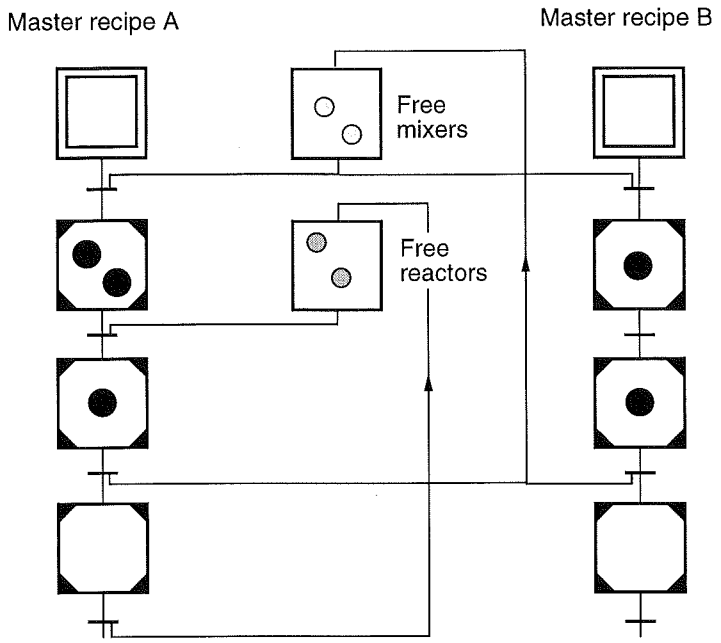


Figure 9.11 Combined net with all master recipes.

The combined network of the two recipes $A+B \rightarrow 2D$ and $C+D \rightarrow E$, see Chapter 8.3, is shown in Figure 9.12. The recipe $A + B \rightarrow 2D$ is represented to the left in the figure and the more complex recipe, $C + D \rightarrow E$ is represented to the left. The resources needed are shown in the center of the figure. The tokens representing mixers reside in the upper step, the tokens representing the buffers in the mid-step and the tokens representing the reactors in the bottom-step. The buffers are only used by the recipe $C + D \rightarrow E$. This net is large and complex and therefore not suitable for representation.

A combined net can however be represented in a more attractive way by using connection posts. Each master recipe function chart is represented separately. The different function charts are connected through connection posts and analysis can be performed in the same way as previously.

Advantages and Disadvantages

The structuring alternative presented in this section gives a more compact representation compared with the structuring alternative given in Chapter 9.1. This structuring alternative makes it easy to see how many batches of a certain type that are under production. The recipes

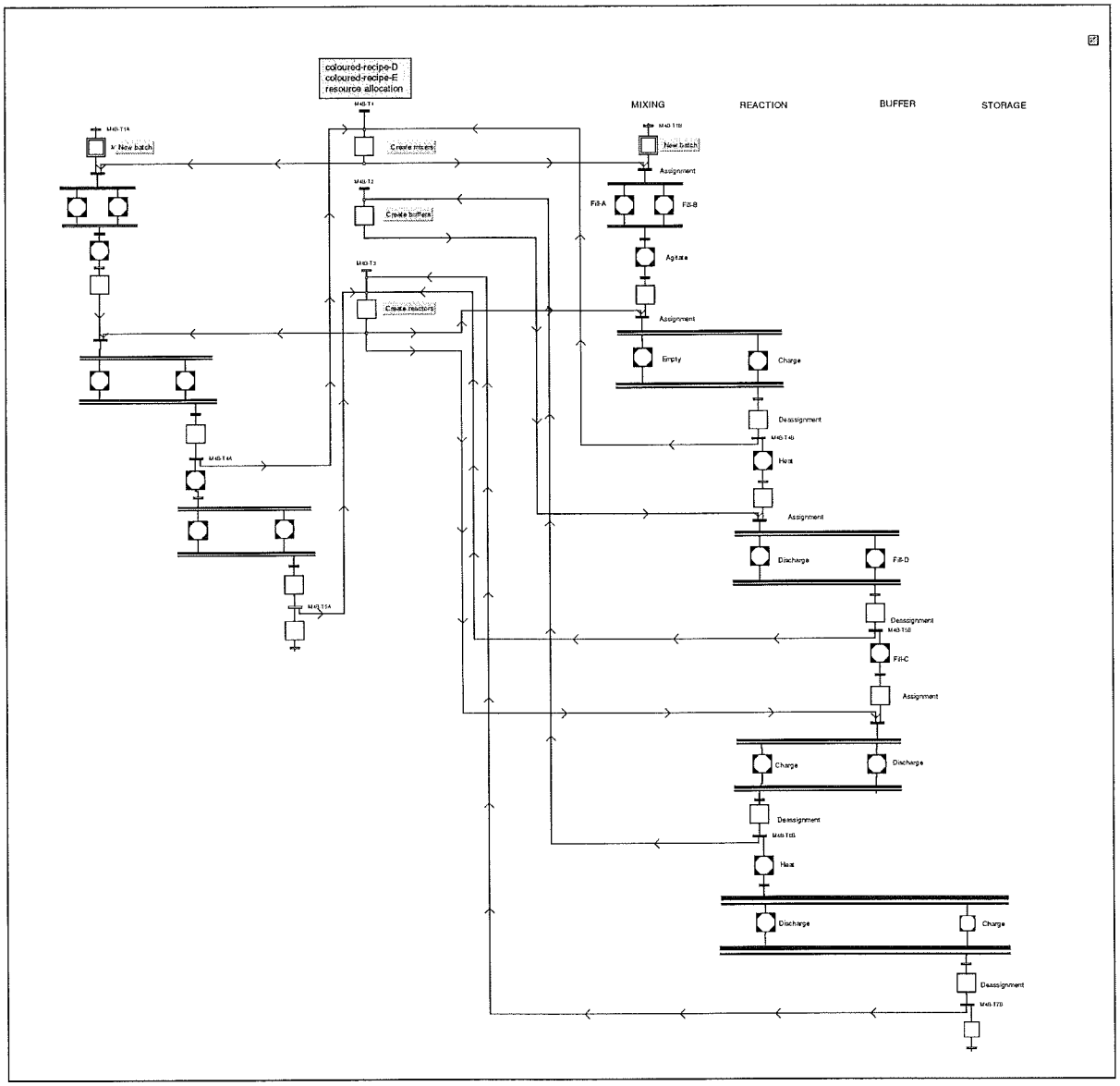


Figure 9.12 Combined net with all master recipes.

have been structured using phases. It can however also be structured using operations, in the same way as in Chapter 9.1. If the master recipe function chart is changed this will affect all control recipes. This can be both an advantage and a disadvantage. A drawback is that it is not possible to modify the procedure of one control recipe alone. This can however be resolved by using one HL-Grafchart function chart for each batch, according to Section 9.1, together with connection posts. This situation is shown in Figure 9.13.

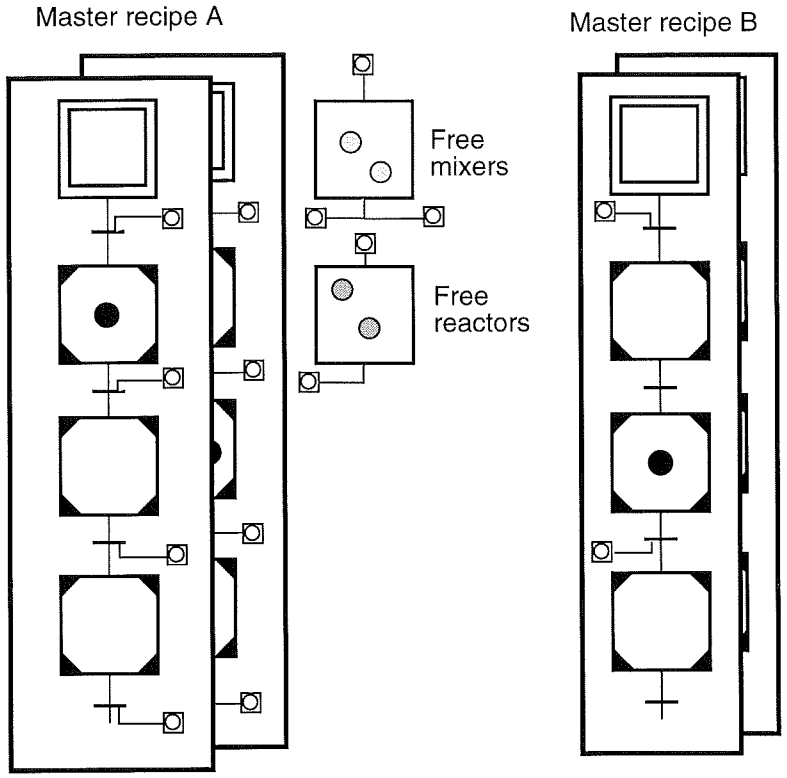


Figure 9.13 Combined net with all master recipes using connection posts.

9.3 Multi-dimensional Recipes

The recipe structuring alternatives presented in this section use all four features of HL-Grafchart, i.e., parameterization, methods and message passing, object tokens, and multi-dimensional charts.

The multi-dimensional feature of Grafchart fits nicely into the batch control problem. In the first recipe structuring alternative, Section 9.1, only one token, representing one batch, could reside in one chart. In the second recipe structuring alternative, Section 9.2, several tokens, each representing one batch, could reside in a chart, provided they were all of the same type. The tokens shared the structure of the master recipe but contained specific information about the equipment to use when being produced. Another, more sophisticated structuring alternative is to let the token contain, not only information about the equipment but also a method describing how it is produced. The chart in which the tokens reside now becomes more general and allows for

tokens of different types to reside in the same chart. The idea is shown in Figure 9.14

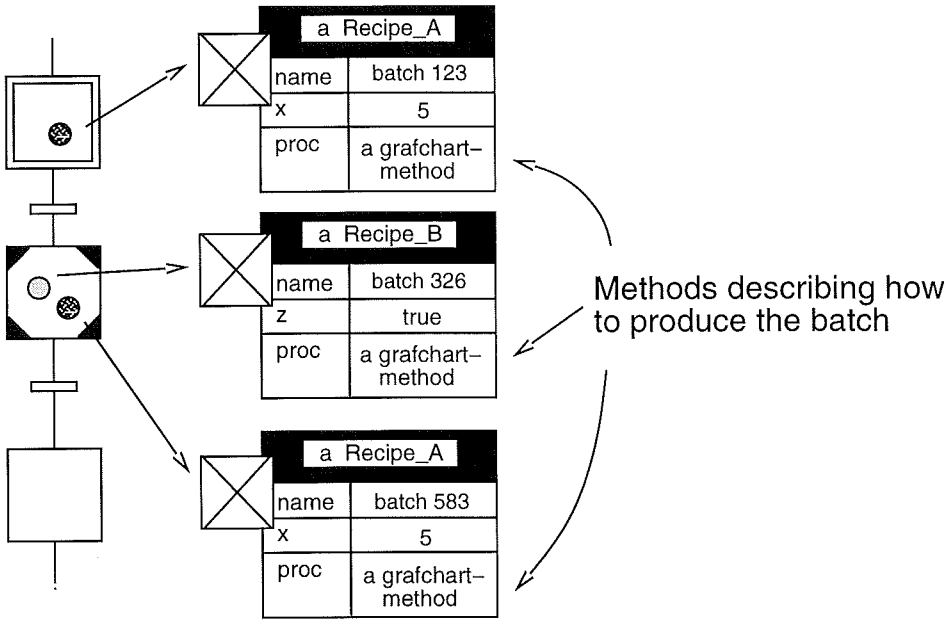


Figure 9.14 A multi-dimensional chart.

Multi-dimensional recipes with implicit resource allocation

The first multi-dimensional recipe is shown in Figure 9.15. The function chart of the base-level of the multi-dimensional recipe is very general and applies to all batches, independently of their type. In Figure 9.16, this chart consists of one initial step, two steps and one procedure step. The tokens in the initial step can represent the orders effected to the plant. The step following the initial step contains initializational tasks that have to be performed before the production can start. The batches currently under production reside in the procedure step and the finished batches in the last step.

When a token enters the procedure step a call is performed and a method of the token is started. This method describes how this specific batch should be produced and constitutes a part of the second level in the multi-dimensional recipe. The method is implemented as a Grafchart procedure, see Chapter 6.2. The attributes of the Grafchart procedure contain the equipment specification of the batch. The different levels of the multi-dimensional chart communicate through the different dot notations, see Chapter 6.4.

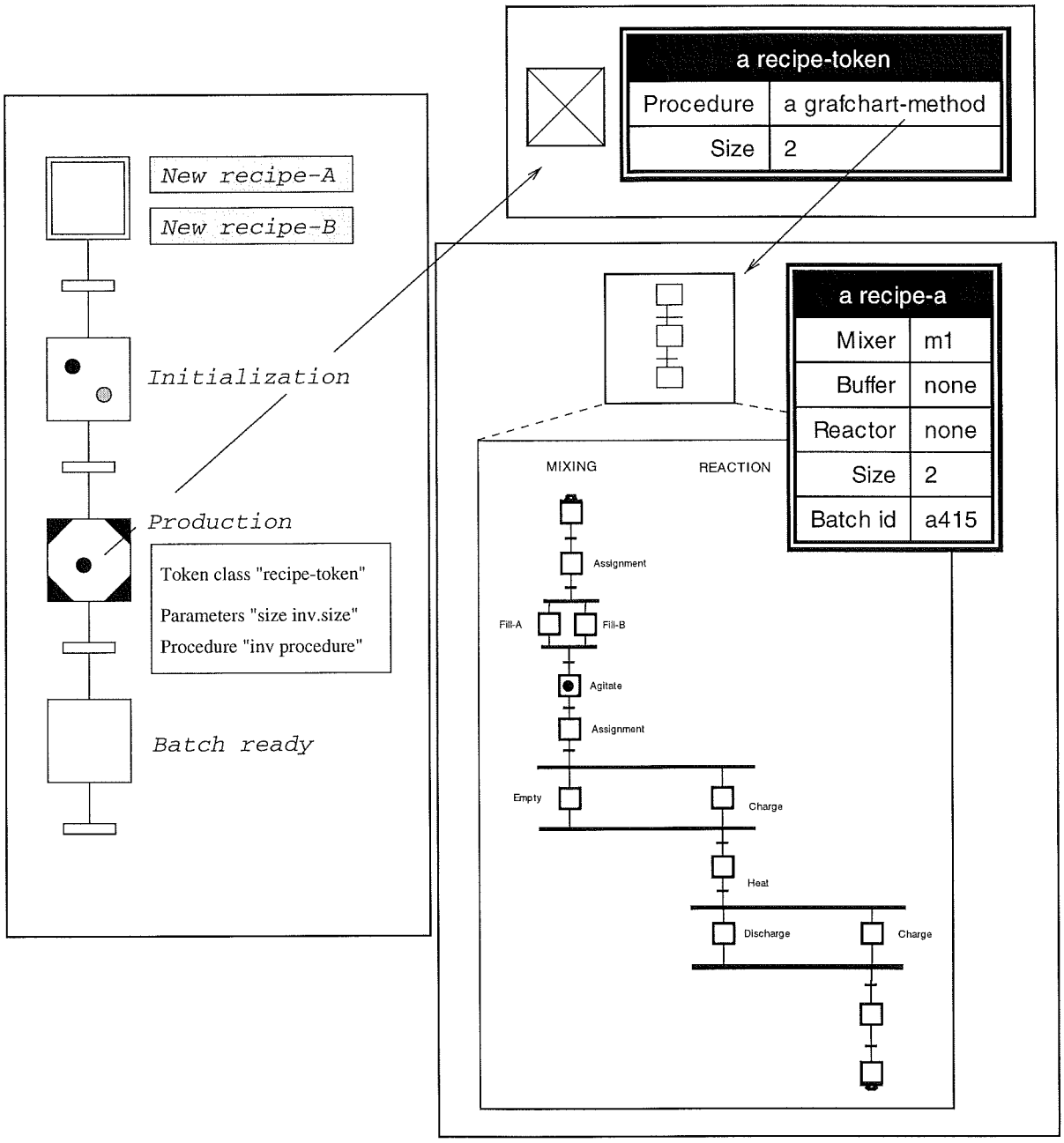


Figure 9.15 Multi-dimensional chart with resource allocation.

Each control recipe, i.e., each token, is represented by an object token and visualized by a grafchart marker. When a new batch is to be produced a new control recipe object token is created and a grafchart marker of the correct type is placed in the initial-step of the function chart. A master recipe is represented as an object token with attributes

and a method. A control recipe object token is created by copying the master recipe object token.

The resource allocation of this recipe structuring alternative is done implicitly. Immediately before an equipment is needed a simple search algorithm is executed and the correct equipment is automatically assigned to the batch.

Multi-dimensional recipes with explicit resource allocation

In this structuring alternative the resource allocation is explicitly represented. The function chart of the base-level contains an initial-step, a process step, two circular path (one for the batch and one for the equipments) and a final step, see Figure 9.16.

After initialization of a new batch, which is done in the initial step, the production of the batch is started. This is done in the process step, from which the method of the token is called. The method represents the recipe procedure. A multi-dimensional structure of the chart is now achieved. When the call has been performed the token directly moves to the next step. It enters a circular path where it sequentially requests resources and continues its recipe execution in that resource. The token does not leave the circular path until the execution of the batch has reached its end, i.e., until the batch is ready.

The resources (equipment units) in the plant are also represented by tokens. These tokens are included in another circular path, describing the state of the resources. A resource can either be available, occupied by a batch or in cleaning mode (cip). A batch requiring a resource can only reserve a resource if the resource is available. If the resource required by the batch is not available the batch has to wait until it becomes available. When a batch releases a resource, the resource has to be cleaned before it can be utilized again by another batch.

Advantages and Disadvantages

The recipe structures presented in this section are more advanced, and maybe harder to understand at a first glance, than those previously presented. However, the structures have advantages that the previously ones do not have. All types of control recipe tokens are allowed to circulate in the base-level function chart, and as a consequence of

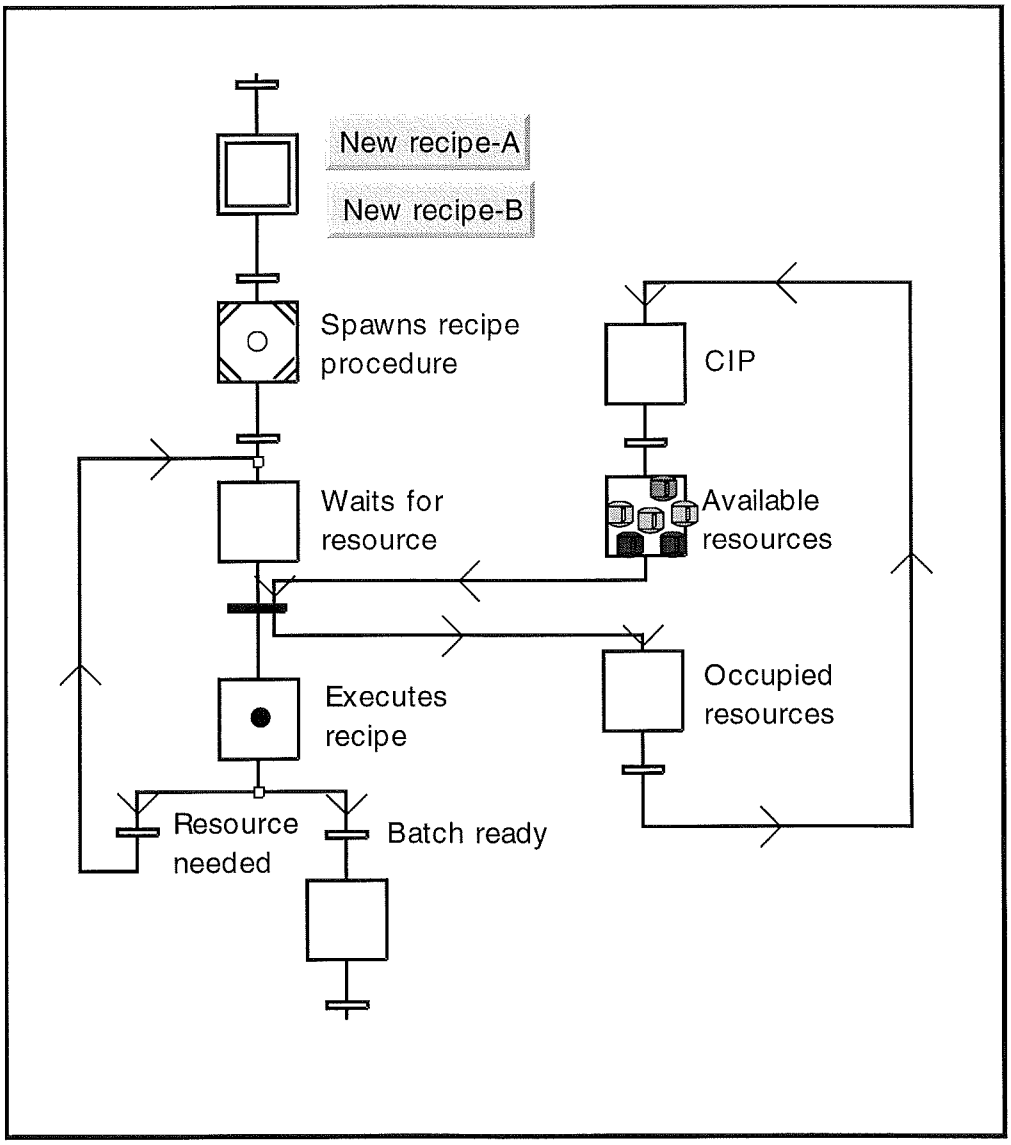


Figure 9.16 Multidimensional chart with resource allocation.

this only one chart is needed for a production cell. The base-level function chart gives complete information about: (1) the number of ordered batches, (2) the number of batches in production, (3) the ready produced batches and, (4) the status of the equipment units. If more specific information about a particular batch is required, this can easily be obtained since the information is well structured and placed within the token.

9.4 Process Cell Structured Recipes

An alternative to the previous solutions is to represent the process units as steps instead of tokens. This gives a resource allocation chart that resembles the physical structure of the process cell according to Figure 9.17 where a simplified process cell is shown.

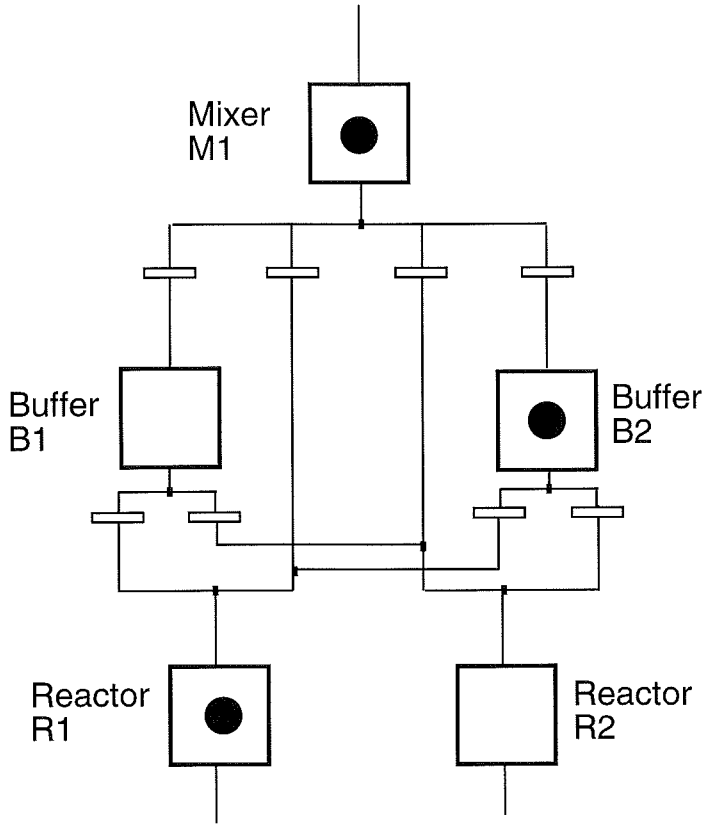


Figure 9.17 Process cell structured recipe.

Each batch is represented as a control recipe token that moves around in the resource allocation net. The presence of a control recipe token in an equipment step indicates that the recipe is executing an operation in that unit process. The resource allocation mechanism is hidden in the transitions between the equipment steps. The net structure obtained is very similar to the Petri net structures used for analyzing batch control cells in [Genrich *et al.*, 1994], [Yamalidou and Kantor, 1991]. The difference is that here the resource allocation is integrated with recipe execution through the multi-dimensional chart nature.

A problem with this approach is how to handle the parallelism between

operations performed in different units, e.g., the emptying of a mixer which is done in parallel with the charging of a reactor. One solution could be the following: When the emptying/charging is started, the token placed in the mixer step is transformed into a real number, e.g., the number 1.0 indicating that all of the content is contained in the mixer. At the same time the number 0.0 is placed in the reactor indicating that nothing of the content is in the reactor. As the mixer is emptied the token number of the mixer is decreased and the token number in the reactor is increased. When the mixer is empty, i.e., the number equals 0.0, and all of the content is in the reactor the number of the mixer is deleted and the number in the reactor, which is equal to 1.0, is transformed in to a black dot again.

In the Petri Net field, nets with tokens as real numbers are known as continuous Petri Nets, see Chapter 2.6.

Advantages and disadvantages

The recipe structure presented in this Section focuses more upon the process cell then on the recipe itself. An advantage can be the similarities to Petri Nets.

9.5 Resource Allocation

Resource allocation is an important part in a recipe management system, as in many other areas. If the resource allocation is not done in a correct way things can get mixed up, e.g., if a mixer is allocated to more than one batch their contents will be mixed and might have to be thrown away, which could be very costly for the company. Improper handle of exclusive-use common resources, such as the process units, might also lead to a deadlock situation, i.e., a situation were two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.

The resource allocation problem is treated in two main ways: implicitly and explicitly. In the implicit way, the resources are not represented by any token, instead the allocation is hidden in the actions of a step. This is the case of the structures presented in Section 9.1 (Recipes

based on recipe phases) (Recipes based on recipe operations), Section 9.2 (Recipes with several tokens), and Section 9.3 (Multi-dimensional recipes with implicit resource allocation). There are two versions of the implicit resource allocation. The first and most basic version is to let the operator prespecify, before starting the recipe execution, all the equipment that the batch will use. If the specified resource is not available when needed, the recipe execution will pause and wait until it becomes available. In the second version of implicit resource allocation, the resources are automatically allocated. A simple search algorithm looks for an available resource of the correct type, and, if found, allocates it to the recipe.

In explicit resource allocation, resource tokens are used to represent the resources, i.e., the state of the resources in the plant are graphically presented for the user. This is the case of the structures in Section 9.2 (Recipes extended with resource allocation) (Combined network), and Section 9.3 (Multi-dimensional recipes with explicit resource allocation).

Resource allocation of exclusive-use resources and deadlock analysis have been looked at in other areas and their results might be useful also for a recipe management system in the batch control field.

Semaphores and Monitors

In real-time programming the concept of semaphores are used to obtain mutual exclusion, [Dijkstra, 1968]. A semaphore is a flag indicating if a resource is available or not. If a process requests a nonavailable resource its request can be placed in a queue. When the requested resource is released the first process in the queue can allocate it. The queue is maintained by a monitor. Different processes can be given different priorities. This means that a process requesting a non-available resources might be placed in the first position of the queue even if there are already other processes, assumed its priority is higher than that of the other processes. The task of the monitor is to adjust the order in the queue and to indicate for the first process when the resource in question becomes available. For further reading about semaphores and monitors see [Burns and Wellings, 1996].

The resource token in the recipe structures with explicit resource allocation can be compared with a semaphore. The fact that different

recipes could be given different priorities and that they could be put in a waiting queue is however relevant also for batch processes. One way of approaching this would be to introduce special FIFO-steps that maintain a queue of tokens.

In Figure 9.18 a small example of a system with mutually exclusive resources is shown.

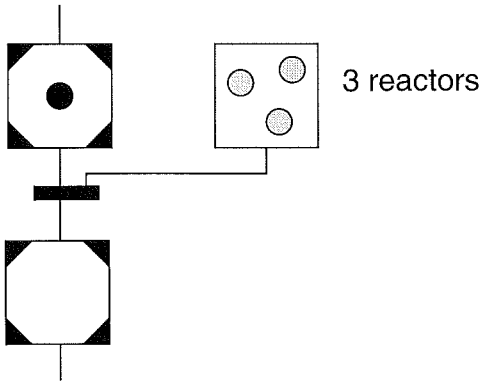


Figure 9.18 A Grafchart structure representing three mutually exclusive resources.

Shared resources

The semaphore construct can also be used for shared resources, i.e., resources that can be used by maximum n simultaneous users. A situation like this can be implemented in Grafchart. Each sharable resource is represented by as many tokens as its maximum usage limit, see Figure 9.19.

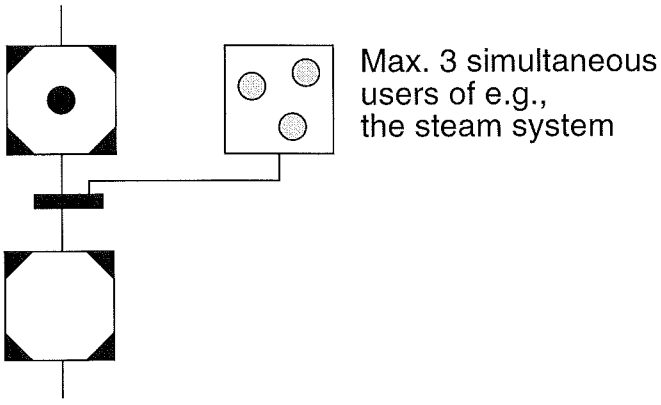


Figure 9.19 A Grafchart structure representing a limited sharable resource.

Sharable resources might also have real-value capacity constraints. This can be implemented in HL-Grafchart, according to Figure 9.20.

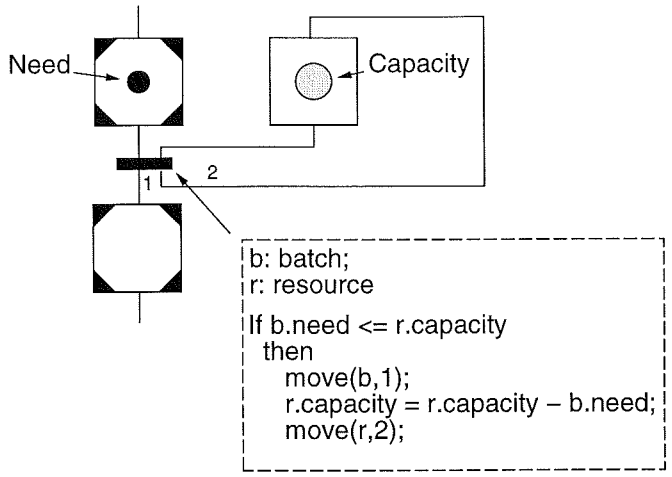


Figure 9.20 A HL-Grafchart structure representing a sharable resource with real-valued capacity constraints.

Situations were the resource has to fulfill certain constraints, e.g., the resource has to have a volume not less than a certain value, can be implemented in HL-Grafchart using the same structure as in Figure 9.20 extended with a receptivity checking the constraint.

Banker's algorithm

An algorithm commonly used for resource allocation in real-time systems is `Banker's algorithm`. The algorithm is used for deadlock-avoidance, [Silberschatz and Galvin, 1995]. When a new process enters the system it must declare the maximum number of instances of each resource type that it might need. This number may not exceed the total number of resources in the system. When a process requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If so, the resources are allocated, otherwise, the process must wait until some other process releases enough resources.

Ideas from this algorithm might be useful if implementing a deadlock-detecting system within the recipe management system.

PN-analysis

A Petri Net can formally be analyzed with respect to deadlocks, using the analysis methods described in Chapter 2.4. However, the analysis can only be done for autonomous PN, see Chapter 2.1 (Dynamic Behavior). If a PN is non-autonomous the sub-adjacent PN, i.e., the corresponding autonomous PN, will be investigated.

Since the structure of a Grafchart or a HL-Grafchart can be transformed into that of a Grafcet and thereby also into that of a Petri Net, these analysis methods can be applied to the Grafchart and HL-Grafchart structured recipes, if explicit resource allocation is used.

For example, if batches of the two types $A + B \rightarrow 2D$ and $C + D \rightarrow E$ should be produced in the same plant at the same it could be good to know how many batches that can concurrently be in production without the risk of entering a deadlock situation. By transforming the combined network of the two recipes, i.e., the network shown in Figure 9.12, into a Petri Net, analysis methods can be applied. However, the analysis methods do not take into account the receptivities of the transitions in the net. This means that specific demands that a batch can have on a resource, e.g., that the resource has to have a certain volume, and that are represented within a receptivity, see Figure 9.20, are not considered. To solve this problem, resources with different capacity, must reside in different steps.

9.6 Distributed Execution

The recipe alternatives proposed implicitly assumes a centralized execution environment where the recipe execution and the equipment controllers all reside within the same computer. In an industrial batch control system this is very seldom the case. Instead, a distributed environment is used where, e.g., each equipment controller is implemented as a separate node. The different nodes communicate with each other over a network. The question is then how well the presented recipe alternatives are suited for distributed execution.

The simplest way to obtain distributed execution is to assume that all recipe management, including recipe execution, is performed in a

single supervisory node. The linking between the recipe level and the execution control level, that previously was realized by a Grafchart method call, will now be implemented as a remote procedure (method) call (RPC). A drawback with this approach is that it is still, by large, centralized in nature. If the supervisory recipe execution node fails all recipe execution in the entire batch cell will stop.

True distributed recipe execution can only be obtained by sending recipe information between the different equipment controller. Assume that all the equipment units that a control recipe will use are prespecified by the operators before the recipe is started. This is done in one of the nodes of the system. When the recipe is started the recipe is sent to the first equipment node. When the execution in the first equipment node is finished, the recipe (or only the remaining part of the recipe) is sent to the next equipment node, etc.

Depending on which structuring alternative that is used different amount of information has to be passed between the equipment nodes. If each control recipe is represented by a separate function chart, then the function charts have to be passed between the nodes. If, however, function charts are used only to represent the master recipes and object tokens are used for representing the individual batches, then things become a bit simpler. It is now possible to store a copy of each master recipe function chart in each node, and to pass only the object token information together with information about the current step of an object token, between the nodes.

If the resource allocation should be integrated with the recipe execution as, e.g., in Figure 9.9 or 9.11, things become more complicated. One possibility is to let one equipment nodes for each equipment type, be responsible for the allocation of resources of that type. When a recipe needs to allocate a resource of a certain type, a request is sent to the resource allocation node for the equipment type. The batch will wait in the preceding equipment unit until a suitable equipment unit becomes available. Information about the identity of the new equipment unit is sent back to the equipment node where the batch is currently waiting, and the transfer to the new equipment unit can start. A rather elaborate handshaking procedure is needed to implement this synchronization. Distributed recipe execution of this nature is available in modern

commercial batch control system such as, e.g., the batch control system in SattLine from Alfa Laval Automation.

9.7 Other Batch Related Applications of High-Level Grafchart

Grafchart and HL-Grafchart has been used in other batch related applications.

Automating Operating Procedure Synthesis

Plant personnel often spend considerable amount of time and effort, preparing and verifying the recipes, i.e., the operating procedures. It would therefore be valuable to automate the synthesis. A framework for this task is presented in [Viswanathan *et al.*, 1997]. The framework is based on Grafchart. Grafchart is used to represent the procedural knowledge. The declarative knowledge, i.e., the plant and process specific knowledge, is represented in an object-oriented way. Grafchart is also used to model the information that is incrementally generated during operating procedure synthesis, the so called inferred knowledge. An hierarchical planning strategy is presented that processes the declarative knowledge, utilizing the procedural knowledge, to generate the inferred knowledge incrementally, which leads to the synthesis of the operating procedures.

The implementation of this framework, named *i*TOPS (Intelligent Tool for Operating Procedure Synthesis), is described in [Johnsson *et al.*, 1997]. *i*TOPS is implemented in G2. The application of *i*TOPS to an industrial case study is also presented.

An Integrated Batch Plant Operation System

The activities involved in batch plant operation are of a diverse nature and call for a range of different views to be solved efficiently. However, the activities interact strongly with each other and it is therefore not possible to treat the different activities as different problems.

In [Simensen *et al.*, 1996] an integrated batch plant operation system is presented. The information model underlying this system is based on

a knowledge base containing different models of the plant. The models depicts the problem area in terms of functions, domain and dynamics. To structure the system both user-views and representation-views are used. The system has the ISA-S88.01 standard as a basis.

A prototype of the integrated batch plant operation system has been implemented and it has been applied to a simulated batch scenario. Depending on what kind of activity the user would like to perform, different views are desirable. The user uses an extended version of the Control Activity Model of ISA-S88.01 as a navigator metaphor. The integrated batch plant operation system is implemented in G2.

9.8 Summary

The main application of High-Level Grafchart is recipe structuring and recourse management in multi-product, network-structured batch production cells. High-Level Grafchart is well suited for these tasks. By using the features of High-Level Grafchart in different ways, large variations can be made when structuring the recipes. They still however, comply with the ISA S88.01 standard. The simplest way is to represent each control recipe with a Grafchart function chart with batch specific information represented as attributes of the function chart. The recipes can also be structured using object tokens. In this case, control recipes of the same type, move around in the same chart. Batch specific information is stored within the token. By using the multi-dimensional structure of High-Level Grafchart, the chart in which the tokens reside can be given a general structure. Tokens of different types, i.e., of different master recipes, can now reside in the same chart. The token contains its own recipe-procedure as a method.

Resource allocation can be made in two ways: implicitly or explicitly. If the resource allocation is done explicitly, i.e., if each resource is represented by a token, formal analysis methods can be applied to the net, and properties like deadlocks can be detected. Resource allocation of exclusive-use resources and deadlock-analysis have been looked at in other areas, e.g., Petri Nets, and their results can be useful also for a recipe management system in the batch control field.

10

Conclusions

This thesis has presented High-Level Grafchart and its application to recipe based batch control. High-Level Grafchart is a toolbox aimed at supervisory control. It has its roots in several areas:

- Grafcet: The syntax of Grafcet is clear and has been very well accepted in industry for sequence control.
- Petri Nets: Formal analysis methods for Petri Nets exist for verification of properties such as deadlocks.
- High-Level Petri Nets: High-Level Petri Nets allow a compact description of systems with several similar parts.
- Object-oriented programming languages: Object-oriented programming languages have powerful abstraction facilities.

High-Level Grafchart is based on Grafchart. Grafchart is both a mathematical model and a toolbox. It is based on the well accepted syntax of Grafcet, i.e., steps and transitions. In addition to this Grafchart has extended abstraction and error handling facilities. High-Level Grafchart adds four new features: parameterization, methods and message passing, object tokens, and multi-dimensional charts. This features increase the abstraction and structuring possibilities and thereby make the implementation of a control system easier. High-Level Grafchart can be used both on the local PLC control level and on the supervisory control level. High-Level Grafchart is implemented in G2, an object-oriented programming environment.

The main application area of High-Level Grafchart is batch control in general and recipe structuring and resource management in multi-product, network-structured batch production cells in particular. In the thesis it is presented how High-Level Grafchart can be used for recipe structuring. By using the features of High-Level Grafchart in various ways, recipes can be given different structures with different advantages and disadvantages. All structures comply with the recent batch standard ISA S88.01.

An object oriented language, such as High-level Grafchart, is well suited for representing the hierarchical structure of the physical batch plant. High-Level Grafchart can be used to represent all the levels in the hierarchical procedural control model, i.e., the entire recipe. By using the support for methods and message passing, the linking between the control recipe procedural elements and the equipment procedural elements can be nicely represented.

Resource allocation can be made in two ways: implicitly or explicitly. If the resource allocation is done explicitly, i.e., if each resource is represented by a token, formal analysis methods can be applied to the net, and properties like deadlocks can be detected.

10.1 Future Research Directions

There are several directions in which the current work can be continued. They can be divided into topics that concern High-Level Grafchart and topics that concern recipe-based batch control.

Formal Definition of High-Level Grafchart

A formal definition of High-Level Grafchart should be developed. This should include both syntax and semantics.

Formal Analysis Power

The relationship between High-Level Grafchart and the existing Petri Net based formal analysis tools needs to be clarified. Subsets of High-Level Grafchart, where the correspondence is exact, can be found. It would also be possible to develop an interface between High-Level

Grafchart and some existing analysis framework. Using this it should be possible to automatically generate, e.g., reachability or coverability graphs for a function chart.

Improvements in the Implementation of High-Level Grafchart

The version of High-level Grafchart that is currently implemented can still be viewed as a prototype. Several improvements are possible concerning, e.g., execution speed, action and transition syntax, etc. It would also be possible to change the local interpretation algorithm of Grafchart into a global interpretation algorithm.

Alternative Implementation Platforms

G2 is currently used as implementation platform. However, there is nothing that prevents High-Level Grafchart from being migrated to other platforms.

Batch Scheduling and Resource Allocation

The current recipe management system performs dynamic resource allocation. By applying Petri Net based methods one can ensure that no deadlocks may occur. Conventional scheduling tools are based on off-line resource allocation. An interesting question that could be investigated is the relationships between dynamic and static resource allocation and how they should be combined. For example, a pre-defined schedule could be used as long as no unexpected events occur. If some event occurs the system switches to dynamic resource allocation. Interesting analogies can be found in the concurrent programming area where, also, both dynamic (priority-based) and static (off-line) scheduling can be used. It is likely that results derived in real-time programming can be applied also for batch scheduling.

Formal analysis of recipe-based batch control

It would be interesting to apply formal methods to the batch control application. This can be done in a number of different ways. Independently of which way that is used it is necessary to be able to model the batch cell. The current recipe execution system contains no explicit model of the batch cell and the equipment units. One possibility is to model the equipment as finite automata or a hierarchical automata. By

translating the Grafchart controllers into Grafcet and then further into finite automata it may be possible to apply Supervisory Control Theory combined with Grafcet according to [Charbonnier, 1996] to ensure that the recipes are correct. Another possibility is to use continuous Petri Nets to model the batch plant and to translate Grafchart and High-level Grafchart into ordinary Petri Nets. The combined controller and batch cell model would now constitute a hybrid Petri net which could be formally analyzed.

Monitoring of Batch and Discrete Systems

Monitoring of batch control and discrete manufacturing systems is an interesting area. In batch control this task consists both of the monitoring of the ordinary components, compare monitoring of continuous processes, and monitoring of the recipe execution. The approach will be based on a model of each equipment component. This model will be expressed as a hierarchical automaton represented by Grafchart. When a recipe wants to perform an operation on an equipment object, it is first checked that the equipment is in a state that allows the operation. When the operation is performed, one, or several, state transition are performed in the equipment model. The equipment model will also contain the basic, non-recipe oriented, safety interlocks.

11

Bibliography

AGAOUA, S. (1987): *Spécification et commande des systèmes à événements discrets, le grafcet coloré.* PhD thesis, Institut National polytechnique de Grenoble.

ARC (1996): "Batch process automation strategies." Industrial Automation Strategies for Executives. Automation Research Corporation, Memorandum.

ÅRZÉN, K.-E. (1991): "Sequential function charts for knowledge-based, real-time applications." In *Proc. Third IFAC Workshop on AI in Real-Time Control.* Rohnert Park, California.

ÅRZÉN, K.-E. (1993): "Grafcet for intelligent real-time systems." In *Preprints IFAC 12th World Congress.* Sydney, Australia.

ÅRZÉN, K.-E. (1994a): "Grafcet for intelligent supervisory control applications." *Automatica*, **30:10**.

ÅRZÉN, K.-E. (1994b): "Parameterized high-level Grafcet for structuring real-time KBS applications." In *Preprints of the 2nd IFAC Workshop on Computer Software Structures Integrating AI/KBS in Process Control Systems, Lund, Sweden.*

ÅRZÉN, K.-E. (1996a): "A Grafcet based approach to alarm filtering." In *Proc. of the IFAC World Congress 1996.*

ÅRZÉN, K.-E. (1996b): "Grafchart: A graphical language for sequential supervisory control applications." In *Proc. of the IFAC World Congress 1996.*

ÅRZÉN, K.-E. and C. JOHNSSON (1996): "Object-oriented SFC and ISA-S88.01 recipes." In *WBF'96—World Batch Forum.* Toronto, Canada.

BASTIDE, R., C. SIBERTIN-BLANC, and P. PALANQUE (1993): "Cooperative Objects: A concurrent, Petri-net based, object-oriented language." In *Proceeding of IEEE Conference on Systems, Man and Cybernetics.* Le Touquet, France.

BATTISTON, E., F. DE CINDIO, and G. MAURI (1988): "OBJSA: A class of high level nets having objects as domains." In ROZENBERG, Ed., *Advances in Petri Nets 1988*, vol. 340 of *Lecture notes in computer science*, pp. 20–43. Springer Verlag, Berlin Heidelberg, New York.

BRETTSCHMEIDER, H., H. GENRICH, and H. HANISCH (1996): "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.

BURNS, A. and A. WELLINGS (1996): *Real-time systems and programming languages.* Addison-Wesley.

CHARBONNIER, F. (1996): *Commande supervisée des systèmes à événements discrets.* PhD thesis, Institut National Polytechnique de Grenoble.

CHARBONNIER, F., H. ALLA, and R. DAVID (1995): "The supervised control of discrete event dynamic systems: A new approach." In *34th Conference on Decision and Control.* New Orleans.

DAVID, R. (1995): "Grafcet: A powerful tool for specification of logic controllers." *IEEE Transactions on Control Systems Technology*, **3:3**.

DAVID, R. and H. ALLA (1992): *Petri Nets and Grafcet: Tools for modelling discrete events systems.* Prentice-Hall International (UK) Ltd.

DESROCHERS, A. A. and R. AL'JAAR (1995): "Applications of Petri nets in manufacturing systems: Modelling, control and performance analysis." *IEEE Press.*

DIJKSTRA, E. (1968): "Cooperating sequential processes." In GENUYS, Ed., *Programming languages*. Academic Press N.Y.

ENGELL, S. and K. WÖLLHAF (1994): "Dynamic simulation for improved operation of flexible batch plants." In *Proc. CIMPRO 94*, pp. 441–455. Rutgers University, New Jersey, USA.

FISHER, T. G. (1990): *Batch Control System; Design, Application, and Implementation*. Instrument Society of America.

FLEISCHHACK, H. and U. LICHTBLAU (1993): "MOBY - A tool for high level Petri nets with objects." In *Proceeding of IEEE Conference on Systems, Man and Cybernetics*. Le Touquet, France.

GAFFE, D. (1996): *Le modèle Grafcet: réflextion et intégration dans une plate-forme multiformalisme synchrone*. PhD thesis, Université de Nice-Sophia Antipolis.

GENRICH, H. J., H.-M. HANISCH, and K. WÖLLHAF (1994): "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*.

HANISCH, H.-M. and S. FLECK (1996): "A resource allocation scheme for flexible batch plants based on high-level Petri nets." In *IEEE SMC, CESA96*. Lille, France.

HARPER, R. (1986): "Introduction to standard ML." Technical Report. Dept. of Computer Science, University of Edinburgh. ECS-LFCS-86-14.

HOLLOWAY, L. and B. KROGH (1990): "Synthesis of feedback control logic for a class of controlled Petri nets." *IEEE Transactions on Automatic Control*, **35:5**, pp. 514–523.

HOLLOWAY, L. and B. KROGH (1994): "Controlled Petri nets: A tutorial survey." In *11th International Conference on Analysis and Optimization of Systems - Discrete Event Systems*. Springer-Verlag. number 199 in Lecture Notes in Control and Information Science, pages 158-168, DES94, Ecole des Mines de Paris, INRIA.

IEC (1988): *IEC 848: Preparation of function charts for control systems*. International Electrotechnical Commission.

IEC (1995): "IEC 1131-3." Technical Report. International Electrotechnical Commission.

JENSEN, K. (1981): "Coloured Petri Nets and the invariant method." *Theoretical Computer Science, North-Holland*, **14**, pp. 317–336.

JENSEN, K. (1990): "Coloured petri nets: A high level language for system design and analysis." In ROZENBERG, Ed., *Advances in Petri Nets 1990*, vol. 483 of *Lecture notes in Computer Science*, pp. 342–416. Springer Verlag, Berlin Heidelberg, New York.

JENSEN, K. (1992): *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.*, vol. 1, Basic Concepts. Springer-Verlag.

JENSEN, K. (1995): *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.*, vol. 2, Analysis Methods. Springer-Verlag.

JENSEN, K. (1997): *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.*, vol. 3, Practical Use. Springer-Verlag.

JENSEN, K. and G. ROZENBERG (1991): *High-level Petri Nets.* Springer Verlag.

JOHNSSON, C. and K.-E. ÅRZÉN (1994): "High-level Grafcet and batch control." In *Symposium ADPM'94—Automation of Mixed Processes: Dynamical Hybrid Systems.* Brussels, Belgium.

JOHNSSON, C. and K.-E. ÅRZÉN (1996a): "Batch recipe structuring using high-level Grafchart." In *IFAC'96, Preprints 13th World Congress of IFAC.* San Francisco, California.

JOHNSSON, C. and K.-E. ÅRZÉN (1996b): "Object tokens in high-level Grafchart." In *CIMAT'96—Computer Integrated Manufacturing and Automation Technology.* Grenoble, France.

JOHNSSON, C., S. VISWANATHAN, R. SRINIVASAN, V. VENKATASUBRAMANIAN, and K.-E. ÅRZÉN (1997): "Automating operating procedure synthesis for batch processes: Part 2. implementation and application." *Computers and Chemical Engineering.* Submitted to.

KLUGE, W. and K. LAUTENBACH (1982): "The orderly resolution of memory access conflicts among competing channel processes." *IEEE Trans. Comp.*, **C-31:2**, pp. 194–207.

LAKOS, C. (1994): "Object petri nets - definitions and realtionship to coloured nets." Technical Report R94-3. Dept. of Computer Science, University of Tasmania.

LAKOS, C. and C. KEEN (1994): "Loopn++: A new language for object-oriented petri nets." Technical Report R94-4. Dept. of Computer Science, University of Tasmania.

LÓPEZ GONZÁLEZ, J. M., J. I. LLORENTE GONZÁLEZ, J. M. SANTAMARÍA YUGUEROS, O. PEREDA MARTÍNEZ, and E. ALVAREZ DE LOS MOZOS (1994): "Graphical methods for flexible machining cell control using G2." In *Proc. of the Gensym European User Society Meeting, Edinburgh, October.*

MEALY, G. (1955): "A method for synthesizing sequential circuits." *Bell System Technical Journal*, **5:34**, pp. 1045–1079.

META SOFTWARE CORPORATION, Cambridge, MA,USA (1993): *Design/CPN Tutorial for X-Windows.* version 2.0.

MOLLOY, M. (1982): "Performance analysis using Stochastic Petri nets." *IEEE Trans. Computers*, **C-31:9**, pp. 913–917.

MOORE, E. (1956): "Gedanken experiments on sequential machines." *Automata Studies*, pp. 129–153. Princeton University Press.

MOORE, R., H. ROSENOF, and G. STANLEY (1990): "Process control using a real time expert system." In *Preprints 11th IFAC World Congress.* Tallinn, Estonia.

MURATA (1989): "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE*, **77:4**.

MURATA, T., N. KOMODA, K. MATSUMOTO, and K. HARUNA (1986): "A Petri net-based controller for flexible and maintainable sequence control and its application in factory automation." *IEEE Transaction Ind. Electron.*, **33:1**, pp. 1–8.

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

NILSSON, B. (1991): "En on-linesimulator för operatörsstöd," (An on-line simulator for operator support). Report TFRT-3209. Department of Automatic Control, Lund Institute of Technology.

OZSU, M. (1985): "Modelling and analysis of distributed database concurrency control algorithms using an extended Petri net formalism." *IEEE Transaction Software Engineering*, **SE-11:10**, pp. 1225–1240.

PETERSON, J. (1981): *Petri net theory and the modeling of systems.* Prentice-hall.

PETRI, C. A. (1962): "Kommunikation mit automaten." Technical Report. Institut fur Instrumentelle Mathematik, Universitat Bonn. Schriften des IIM Nr. 3. Also in English translation, Communication with Automata, New York: Griffiss Air Force Base. Tech. Rep. RADC-TR-65-377, vol. 1, Suppl. 1, 1966.

RAMADGE, P. and W. WONHAM (1989): "The control of discrete event systems." In *Proceedings of the IEEE*, vol. 77, pp. 81–98.

ROSENHOF, H. P. and A. GHOSH (1987): *Batch Process Automation, Theory and Practice.* Van Nostrand Reinhold.

SANCHEZ, A., G. ROTSTEIN, and S. MACCHIETTO (1995): "Synthesis of procedural controllers for batch chemical processes." In *Proc. of 4th IFAC Symposium on Dynamics and Control of Chemical Reactors, Distillation Columns and Batch Processes (DYCORD+'95), Denmark.*

SHAW, W. (1982): *Computer Control of Batch Processes.* EMC Controls Inc, Cockeysville, MD.

SIFAKIS, J. (1978): "Use of Petri nets for performance evaluation." *Acta cybernet.*, **4:2**, pp. 185–202.

SILBERSCHATZ, A. and P. GALVIN (1995): *Operating System concepts.* Addison-Wesly.

SILVA, M. and E. TERUEL (1996): "Petri nets for design and operation of manufacturing systems: An overview." In *First International Workshop on Manufacturing and Petri Nets, Osaka Japan, International Conferences on Application and Theory of Petri Nets (ICATPN96).* Universidad de Zaragoza, Spain.

SILVA, M. and S. VELILLA (1982): "Programmable logic controllers and Petri nets: A comparative study." In *IFAC Software for Computer Control, Madrid, Spain*, pp. 83–88. Ferrate, G. and Puente, E.A., Pergamont Press, Oxford England.

SIMENSEN, J., C. JOHNSSON, and K.-E. ÅRZÉN (1996): "A framework for batch plant information models." Report ISRN LUTFD2/TFRT--7553--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

SONNENSCHEIN, M. (1993): "An introduction to Gina." In *Proceding of IEEE Conference on Systems, Man and Cybernetics*. Le Touquet, France.

SP88 (1995): "Batch control." Instrument Society of America.

SUAU, D. (1989): *Grafcet coloré: Conception et realisation d'un outil de generation de simulation et de commande temps réel*. PhD thesis, Universite de Monpellier.

TITTUS, M. (1995): *Control synthesis for Batch processes*. PhD thesis, Chalmers University of Technology.

VALETTE, R., M. COURVOISIER, J. BEGOU, and J. ALBUKERQUE (1983): "Petri net based programmable logic controllers." In *1st Int. IFIP conference: Comp. appl. in production and engineering*, pp. 103–116.

VISWANATHAN, S., C. JOHNSSON, R. SRINIVASAN, V. VENKATASUBRAMANIAN, and K.-E. ÅRZÉN (1997): "Automating operating procedure synthesis for batch processes: Part 1. knowledge representation and planning framework." *Computers and Chemical Engineering*. Submitted to.

WILLIAMS, T. (1988): "A reference model for Computer Integrated Manufacturing (CIM)." In *International Purdue Workshop on Industrial Computer Systems, ISA*.

YAMALIDOU, E. C. and J. C. KANTOR (1991): "Modeling and optimal control of discrete-event chemical processes using Petri nets." *Computers Chem. Engng*, **15**, pp. 503–519.

YOELI, M. (1987): "Specification and verification of asynchronous circuits using marked graphs." In *Concurrency and Nets*, pp. 605–622.

A

An introduction to G2

G2, developed by Gensym Corporation in the USA, was originally developed as a real-time expert system. It has however evolved into a very powerful object oriented programming environment with strong graphical features. G2 is written in Common LISP which is automatically translated into C. However, all user programming is done in G2's built in programming language using either rules, functions or procedures. G2 runs on a variety of UNIX and Windows platforms.

Classes and objects The programming language of G2 is strictly object oriented. This means that objects are implemented in classes defined in a class hierarchy. Each class has an icon and a number of specific attributes. The icon can be defined either textually with the text editor or graphically using the icon editor in G2. The attributes contain the data associated with a class. Subclasses inherit properties from its superclasses, multiple inheritance is allowed.

An application is built up by placing objects (instances of a class) on a workspace. By connecting the objects their relationship is shown. Associated with each object is its attribute table. The table is automatically created from the definition of the object's class.

Composite objects are objects that have an internal structure. As other objects, composite objects are represented by one icon and have an attribute table. The values of these attributes may be other objects. It is, however, not possible to have a graphical representation of the composed object and its internal objects at the same time. If such a representation is desired this can be implemented using the subworkspace

concept. In G2 all objects may have an associated subworkspace and on this subworkspace other objects may be positioned, i.e., the internal structure of an object can be represented on its subworkspace.

Rules and procedures G2-rules can be used to indicate how to respond to and what to conclude from changing conditions within the application. There are five different types of rules; if-, when- initially- whenever- and unconditionally rules. Rules can be scanned or invoked in a number of ways.

Procedures are written in a Pascal like language. The roles of procedures are dual, either they can be used as ordinary procedures or they can function as processes. Procedures are called from rules, from other procedures or from user actions, e.g., from buttons. Procedures can have input parameters and they can return one or several values. When a procedure operates as a process it is reentrant and each invocation of the procedure execute as a separate task. Since the language of G2 is object oriented it also contains methods. A method is a procedure that implements an operation for an object of a particular class.

Simulator G2 has a built in simulator which can provide simulated values for variables. The simulator allows the simulated expressions to be algebraic equations, difference equations , or first order differential equations.

G2 provides a powerful graphical interface and easy ways to manipulate and reason about objects. It is particularly well fitted for applications which need graphical representations like the batch scenario. For more information about G2 see [Moore *et al.*, 1990].

B

A Dynamic Simulator

The dynamic simulator is used as a substitute for a real plant. It simulates the mass balances, energy balances and the chemical reactions. All simulations are done in real-time.

A table over the notation and the constants used in the dynamic simulator is given in the end of the appendix, Chapter B.7.

Two reactions can be simulated in the scenario:

$$\begin{aligned} A + B &\longrightarrow 2D \\ C + D &\longrightarrow E \end{aligned}$$

The total mass balance, $m(t)$, the component mass balances, $x(t)$, and the energy balance, $T(t)$, see Figure B.1, for the reactions are calculated.

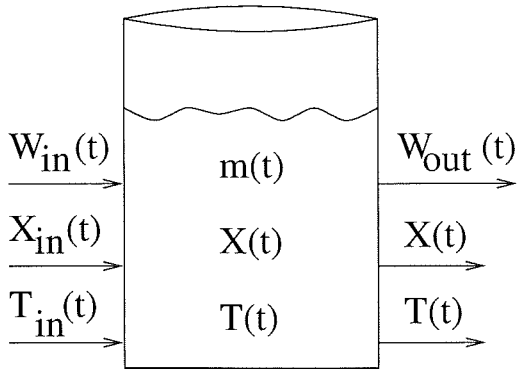


Figure B.1 Mass balance, component mass balance and energy balance.

B.1 Total Mass Balance

The total mass balance of a unit is given by a differential equation together with an initial condition.

$$\begin{aligned} \frac{dm(t)}{dt} &= \underbrace{W_{in}}_{incoming\ massflow} - \underbrace{W_{out}}_{outgoing\ massflow} \\ m(0) &= m_o \end{aligned}$$

The equation comes from the principle of the conservation of mass. $m(t)$ is the weight of the content in the unit, i.e. the mass. The mass is increased if there is an inflow to the unit and it is decreased if there is an outflow of the unit. W_{in} is the mass flowing in to the unit per time unit and W_{out} is the mass flowing of the unit per time unit.

B.2 Component Mass Balance

The component mass balance tells how much there is of a specific substance within a unit, it is given in percentage of the total mass. Unlike mass, chemical components are not conserved. If a reaction occurs inside a system, the weight of an individual component will increase if it is a product of the reaction or decrease if it is a reactant. The mass fraction, $x(t)$, changes continuously and is therefor given by a differential equation together with an initial condition.

$$\begin{aligned} \frac{dx(t)}{dt} &= \underbrace{\frac{W_{in}}{m}(x_{in} - x)}_{in-out} + \underbrace{\frac{M \cdot V}{m} r}_{produced} \\ x(0) &= x_o \end{aligned}$$

where M is the molweight of the substance, V is the volume and r is the reaction rate.

B.3 Energy Balance

The energy balance gives a differential equation for the temperature in the unit, $T(t)$. The equation is based on the first law of thermodynamics, the conservation of energy.

$$\begin{aligned} \frac{dT(t)}{dt} &= \frac{W_{in}}{m}(T_{in} - T) + \frac{1}{c_p \cdot m}Q_p - \frac{1}{c_p \cdot m}Q_c \\ T(0) &= t_o \end{aligned}$$

Q_p is the energy produced during the reaction, Q_c is the energy that is removed by, e.g., a cooling jacket, and c_p is the heat capacity.

Energy Balance for the reactor jacket

The equations given above, apply to all units in the batch scenario. The reactor, however, needs one more equation that describes its jacket.

Produced reaction rate, Q_p:

$$Q_p = \Delta H_{reac} \cdot V \cdot r$$

If $Q_p \leq 0$ the reaction consumes energy, i.e., it is an endotherm reaction. If $Q_p \geq 0$ the reaction generates energy, i.e. it is an exotherm reaction.

Cooling heat, Q_c:

$$\begin{aligned} Q_c &= \mathcal{H} \cdot A_c \cdot (T - T_c) \\ A_c &= k_w \frac{m}{\rho \cdot A} \end{aligned}$$

The cooling heat is the energy that is removed due to, e.g., a cooling jacket.

Energy balance for the jacket of the reactor:

$$\frac{dT_c(t)}{dt} = \frac{W_{ic}}{m_c}(T_{ic} - T_c) + \frac{1}{c_p \cdot m_c}Q_c$$

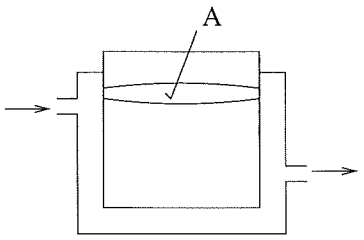


Figure B.2 Reactor with a jacket.

B.4 Level and Volume

Equations for calculating the level and the volume for all the units are also part of the dynamic simulator.

B.5 Reaction Rates

The reaction rates, r_1 and r_2, of the reactions can be calculated by using the Arrenius equation:

$$A + B \longrightarrow 2D \tag{B.1}$$
$$C + D \longrightarrow E \tag{B.2}$$

Eq. B.1 gives:

$$\begin{aligned} r_A &= r_1 \\ r_B &= r_1 \\ r_D &= -2r_1 \end{aligned}$$

Eq. B.2 gives:

$$\begin{aligned} r_C &= r_2 \\ r_D &= r_2 \\ r_E &= -r_2 \end{aligned}$$

$$r_D = -2r_1 + r_2$$

The constants r_1 and r_2 are given by:

$$\begin{aligned} r_1 &= -K_1 \cdot e^{-\frac{E_1}{R(T+273)}} \cdot c_A \cdot c_B \\ &= -K_1 \cdot e^{-\frac{E}{R(T+273)}} \cdot \frac{\rho \cdot x_A}{M_A} \cdot \frac{\rho \cdot x_B}{M_B} \end{aligned}$$

$$\begin{aligned} r_2 &= -K_2 \cdot e^{-\frac{E_2}{R(T+273)}} \cdot c_C \cdot c_D \\ &= -K_2 \cdot e^{-\frac{E}{R(T+273)}} \cdot \frac{\rho \cdot x_C}{M_C} \cdot \frac{\rho \cdot x_D}{M_D} \end{aligned}$$

B.6 Implementation

The objects are defined in a simulation class hierarchy, see Figure B.3.

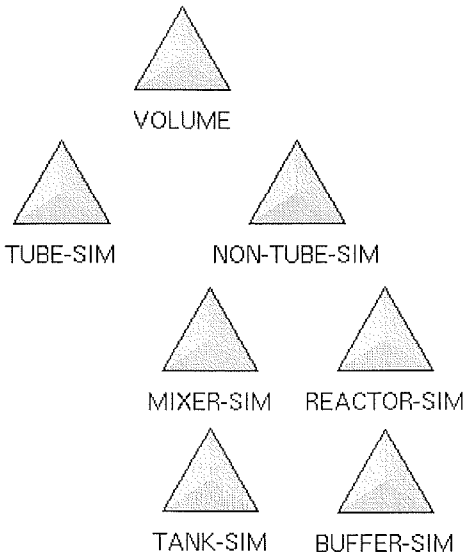


Figure B.3 Simulation classes.

The top class, volume, has two subclasses, 'tube-sim' and 'non-tube-sim'. 'tube-sim' corresponds to the pipes in the cell whereas 'non-tube-sim' corresponds to the units. The class names are extended with '-sim' to indicate that these classes are used in the dynamic simulator.

On the subworkspace of each class the equations specific for this class are placed. Equations that are common to all units are placed on the subworkspace of 'non-tube-sim', these are e.g. the equations for the volume, the level, the component-mass-balances, the energy-balances, the total mass balance and the reaction-rates. In Figure B.4 two of the eight equations associated with the class 'mixer-sim' are shown.

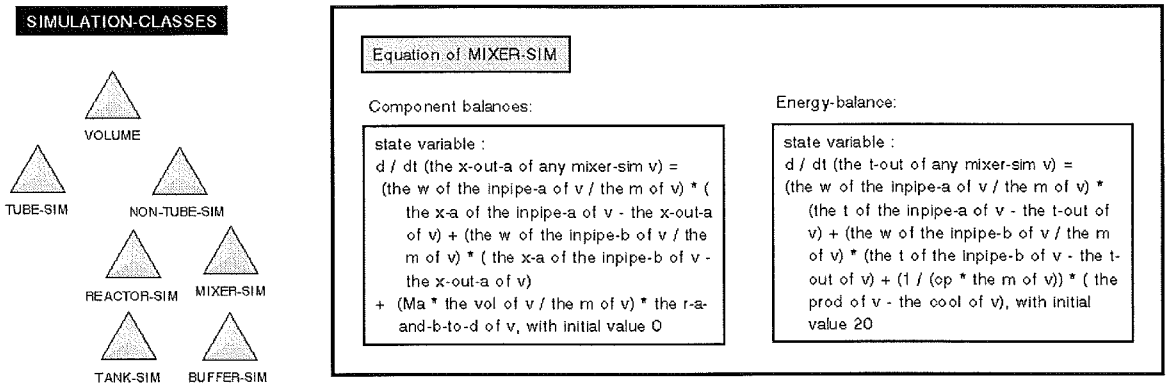
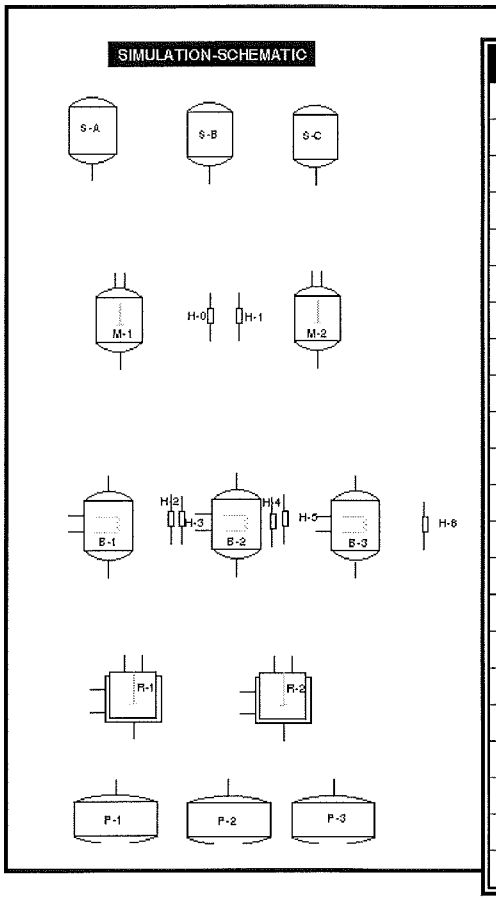


Figure B.4 Equations associated with the class "mixer".

To each class a number of attributes are associated. The attributes common to all classes are defined in the class 'volume', whereas certain attributes specific to only one class are defined within the class itself. Attributes shared by the classes are e.g. the name, the mass fraction of each substance, the volume and the level, the reaction rates of the two reactions and the temperature. Attributes specific to a class are, e.g., be the number of inpipes.

When the program is started the equations are calculated and the differential equations are updated periodically. The results are show as values of the attributes to the objects. In Figure B.5 the simulation view of the batch scenario is shown together with the attribute table of one of the mixers. The table contains the values of the simulated parameters.



M-2, a mixer-sim	
Notes	OK
Item configuration	none
Names	M-2
X out a	0.507
X out b	0.507
X out c	0.0
X out d	4.451e-4
X out e	-0.014
M	279.4
Area	2
Vol	0.279
Level	0.116
Wout	0.0
R a and b to d	-7.307e-4
R c and d to e	-0.0
T out	20.0
Prod	0.0
Cool	0.0
Outpump	0
Outvalve	0
Inpipe a	a pipe-connection
Inpipe b	a pipe-connection

Figure B.5 Attribute table of the mixer-sim named "M-2".

B.7 Notation and Constants

Notation	Unit		Notation in Batch scenario
V	m^3	Volume	Vol
W	kg/sek	mass flow	Wout
X	$(kg/total\ kg)$	mass fraction	x-out-(a,b,c,d,e)
M	$kg/kmol$	mole weight	MA,MB,MC,MD,ME
m	kg	mass	M
r	$kmol/(sek \cdot m^3)$	reaction rate	r-a-and-b-to-e etc.
c	$kmol/m^3$	concentration	$(C_A = \frac{\rho x_A}{M_a})$
ρ	kg/m^3	density	density

E	$J/kmol$	activation energy	e-a-and-b-to-d etc
K	$m^3/(kmol \cdot sek)$	reaction-constant	k-a-and-b-to-d etc
R	$J/kmol \cdot K$	gas law constant	r
T	$^\circ C$	temperature	t out
h	J/kg	enthalpy	$(h = c_p \cdot T)$
u	J	internal energy	$(u \approx H = c_p \cdot T)$
Q_p	J/sek	produced reaction rate	Prod
Q_c	J/sek	cooling heat	Cool
ΔH_{reac}	$J/kmol$	—	h-a-and-b-to-d etc
$\mathcal{H}$	$J/(sek \cdot m^2 \cdot K)$	heat transfer	cooling-coefficient
k_w	m	wall coefficient	wall-area-coefficient
c_p	$J/(kg \cdot K)$	heat capacity	c_p

Constant	
c_p	4180 $J/(kg \cdot K)$
density	1000 kg/m^3
wall-area-coef.	40 m
R	8314 $J/(kmol \cdot K)$
mol-weights:	
Ma	50 $kg/kmol$
Mb	60 $kg/kmol$
Mc	40 $kg/kmol$
Md	50 $kg/kmol$
Me	40 $kg/kmol$
reaction-constants:	
k-a-and-b-to-d	$2 \cdot 10^7$ $m^3/(kmol \cdot sek)$
k-c-and-d-to-e	$7 \cdot 10^{10}$ $m^3/(kmol \cdot sek)$
activation-energy:	
e-a-and-b-to-d	$69.418 \cdot 10^6$ $J/kmol$
e-c-and-d-to-e	$75.00 \cdot 10^6$ $J/kmol$
reaction-energy:	
h-a-and-b-to-d	$-6.99 \cdot 10^7$ $J/kmol$
h-c-and-d-to-e	-75900 $J/kmol$