LUND UNIVERSITY

High-Level Grafcet and Batch Control

Johnsson, Charlotta; Årzén, Karl-Erik

1994

Link to publication

Citation for published version (APA):
Johnsson, C., & Årzén, K.-E. (1994). *High-Level Grafcet and Batch Control*. Paper presented at Automation of Mixed Process: Dynamic Hybrid Systems (ADPM), Brussels, Belgium.

Total number of authors:
2

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

Presented at the Symposium ADPM'94: Automation of Mixed Processes: Dynamical Hybrid Systems, Bryssel, November 1994.

High-Level Grafcet and Batch Control

Charlotta Johnsson and Karl-Erik Årzén

Dept. of Automatic Control
Lund Institute of Technology
Box 118, S-221 00 Lund, Sweden
Email: (lotta,karlerik)@control.lth.se

Abstract: The application of Grafcet to supervisory control applications with special emphasis on batch control is described. Grafchart, a Grafcet based G2 toolbox is presented. An industrial application where it is currently used on-line is described. High-Level Grafchart is an extension of Grafchart that is based on High-Level Petri nets and Object-Oriented Programming. It increases the parameterization and structuring possibilities of Grafchart. The SP88 draft batch control standard is shown to be well suited for representation by High-Level Grafchart.

1. INTRODUCTION

Grafcet, or Sequential Function Charts (SFC), has been widely accepted in industry as a representation format for sequential control logic at the local PLC level through the standards IEC 848 and IEC 1131-3. There is, however, also a need for a common representation format for the sequential elements at the supervisory control level. Supervisory applications such as set-point control, monitoring, fault detection, diagnosis, planning, batch control management and production optimization receive increasing attention from both the academic control community and industry. Reasons for this are 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 run in different operating modes. In the simplest case these can consist of start-up, operation, and production. In the different modes the process and its components may function differently. This means, e.g., that a rule-based supervisory monitoring and diagnosis system must contain different rules for the different operating modes of the process. As the process changes its operating mode the rule-based system must change its set of active rules.

The second situation concerns the case when the problem that the supervisory systems should solve itself can be decomposed into sequential steps. Assume that we want to implement an on-line production optimization system. The system should at regular time intervals perform measurement and production analysis, calculate new optimized parameter settings, and execute the parameter changes. Although the process may operate in the same mode all the time, this problem is still of sequential nature.

Grafchart is the name of a toolbox developed at Lund Institute of Technology that combines the function chart formalism of Grafcet with real-time knowledge-based systems [2], [5], [3]. It is implemented in G2, an object-oriented graphical programming environment primarily aimed at intelligent supervisory control applications [9]. Grafchart is currently used on-line in an oil refinery application [4].

High-Level Grafchart is an extension of Grafchart inspired by High-Level Petri Nets [7] and object-oriented programming (OOP) that is currently under implementation. The implementation of Grafchart is object-oriented. Steps, transitions, etc., and even entire function charts are represented as objects that are defined in a class hierarchy. High-Level Grafchart is based on the possibility to specialize these object classes and thereby adding attributes and refining the behaviour of the object. In this way it is, e.g., possible to parameterize steps and macro steps, use function charts as object methods and let the tokens in a function chart be information carrying objects that themselves may contain function charts as methods.

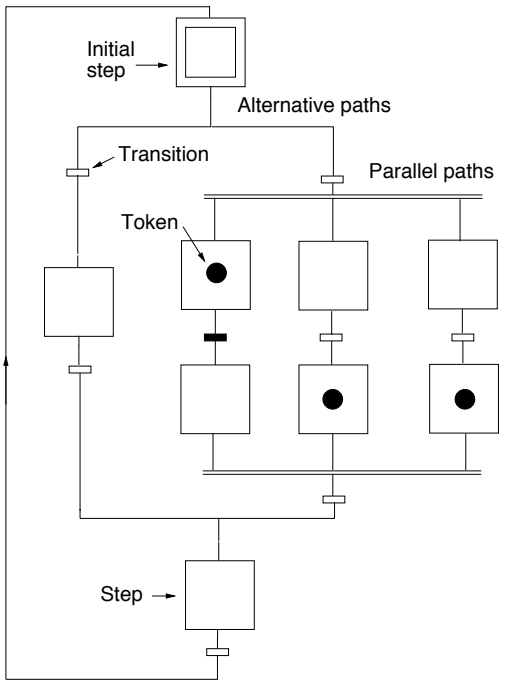

Figure 1. Grafchart graphical syntax

This increases the structuring facilities and makes it possible to reuse large parts of an application.

Sequential control is very important in batch control. Batch control is currently the subject of large interest. The ISA draft standard SP88, currently under development, is one attempt to define the terminology, models and functionality of batch control systems. SP88 already mentions Grafcet but only at the very lowest level for representing the phases in a recipe.

"High-Level Grafcet for supervisory sequential control" is the name of a project with the aim to develop High-Level Grafchart and to apply it to general supervisory control applications with special focus on batch control.

In Section 2 Grafchart is described and compared to standard Grafcet. The industrial oil refinery application is described in Section 3. High-Level Grafchart is described in Section 4. The mapping between SP88 and High-Level Grafchart is finally given in Section 5.

2. GRAFCHART

Grafchart uses the graphical function chart syntax from Grafcet [6]. It supports steps, transitions, alternative branches and parallel branches according to Fig. 1. An entire function chart is represented as a `Grafcet process` object. Grafchart charts can be closed or non-closed according to Fig. 2

The main difference between Grafcet and Grafchart concerns the representation of step

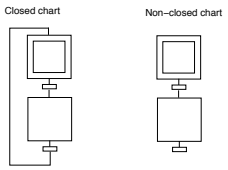

Figure 2. Closed and non-closed charts

actions. Grafcet actions operate upon boolean variables. Different action types exist, e.g., level actions, impulse (stored) actions, conditional actions, time-limited actions and time-delayed actions. In Grafchart step actions are represented as G2 rules that are associated with the step. The rules can be conditional or unconditional. They can be defined to only be fired once when the step becomes active. This gives a behaviour similar to impulse actions. By associating a scan interval with a rule the rule will be fired at a frequency determined by this scan interval as long as the step is active. It is also possible for a rule to be invoked through forward or backward chaining.

The full G2 rule syntax can be used in Grafchart. However, in order to further simplify for the user the somewhat annoying syntax of G2 expressions has been replaced by a Pascal-like dot-notation. That means that instead of referring to an object attribute using the standard G2 syntax `the attribute1 of object1` the shorter `object1.attribute1` is used. The actions that can be performed include all of G2's built in rule actions. For example, it is possible to assign values to variables (the conclude action), to start procedures, to create and delete objects, hide and show information, perform animation actions, etc.

The implementation of the toolbox is based on the G2 concept of *activatable subworkspaces*. A workspace is 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. 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 inference engine. This means, e.g., that rules placed on a deactivated subworkspace cannot be invoked.

Both steps and transitions are represented as G2 objects that have activatable subworkspaces. The rules that should be active when a step is active are placed on the subworkspace of the step. The subworkspace is only active when the step is active, i.e., the rules are only executed when the step is active. Each transition contains an attribute named "condition". Here, the user enters the logical condition for when the transition should fire expressed as a text string. This string may contain the previously mentioned dot-notation. During initialization the condition attribute is used to

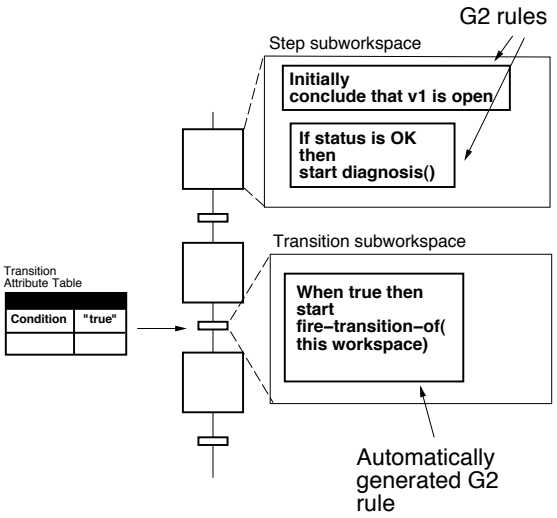

Figure 3. Steps and transitions with their subworkspaces

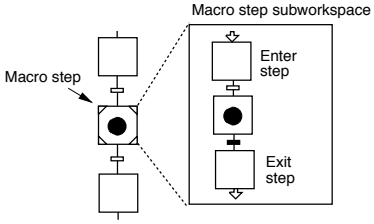

Figure 4. A macro step

automatically generate the rule that executes the transition firing. This rule is located on the subworkspace of the transition. In this way the rule will only be tested when the transition is active, i.e., when the step preceding the transition is active. The situation is shown in Fig. 3.

2.1 *Macro steps and procedure steps*

Macro steps are used to represent steps that have an internal structure of (sub)steps, transitions and macro steps. The internal structure is placed on the subworkspace of the macro step, see Fig. 4. Special enter-step and exit-step objects are used to indicate the first and the last substep of a macro step. A macro step must have exactly one enter-step and at least one exit-step. When the transition preceding a macro step becomes true the enter-step upon the subworkspace of the macro step and 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 an exit-step.

Sequences that are executed in more than one place in a function chart are represented as Grafchart procedures. The common sequence is represented as a function chart on the subworkspace of a Grafchart procedure object. Procedures have enter-steps and exit-steps in the same way as macro-steps. The call to a procedure is represented by a procedure step. The situation is shown in Fig. 5.

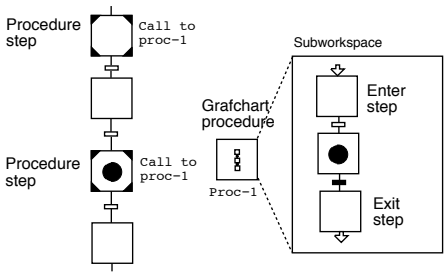

Figure 5. Procedure step and Grafchart procedure

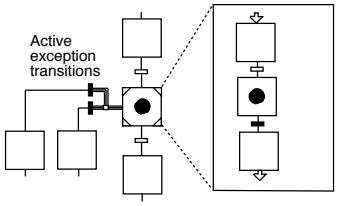

Figure 6. Exception transitions connected to a macro step.

Grafchart procedures should be used to represent standard sequences that are called at several places in a function chart. In plain Grafchart, procedures are not reentrant, i.e., they can only be called from one procedure step at the same time. It is the user's responsibility to ensure that not more than one procedure step calls the same procedure at the same time.

2.2 *Exception transitions*

Exception transitions are an abbreviation of the formal Grafcet model that is used in the situation when it is desired that the same transition should apply to multiple steps. An exception transition is a special type of transition that only may be connected to macro steps, procedure steps, and Grafchart procedures. An ordinary transition connected after a macro step or procedure step will not become active until the execution has reached an exit step. An exception transition is active all the time that the macro step or procedure step is active. If the exception transition becomes true while the corresponding macro step is executing the execution will be aborted and the step following the exception transition becomes active. This also applies recursively, i.e., if the macro step contains other macro steps or procedure steps that are currently active, these will also be aborted. An example of exception transitions is shown in Fig. 6

An exception transition connected to a procedure call gives an exception exit from that specific

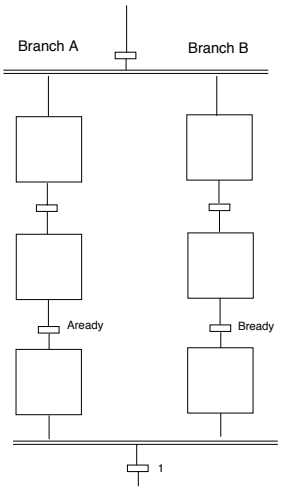

Figure 7. Do in parallel.

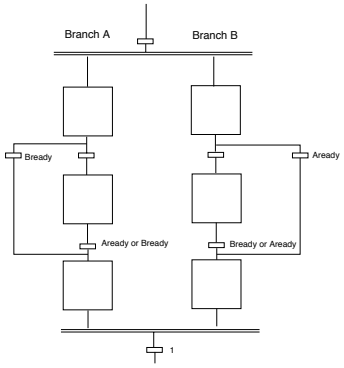

Figure 8. Do in parallel until one completes.

call to the associated Grafchart procedure. An exception transition connected to a Grafchart procedure gives an exception exit that is common to all calls to that procedure.

Exception transition can also be used in the case where we want parallel execution of a number of branches until the first of them completes. This is inconvenient to express with Grafcet's parallel constructs. Consider the example in Fig. 7. Here we have two parallel branches A and B. The transition following the synchronization of the parallel branches will not become enabled until both branches have completed. If we want to express that we want to continue as soon as one of the branches has completed using Grafcet we get a quite complex function chart according to Fig. 8 By encapsulating the parallel construct in a macro step and using an exception transition a much cleaner solution is obtained, see Fig. 9. In order to avoid the additional hierarchical level that the macro step imposes one could alternatively consider the simplified graphical syntax of Fig. 10. This is currently not allowed in Grafchart.

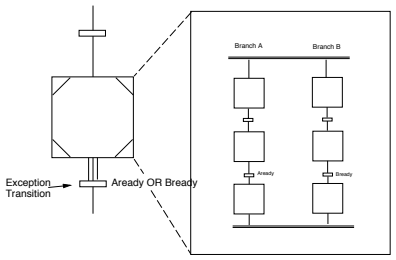

Figure 9. Do in parallel until one completes with exception transitions.

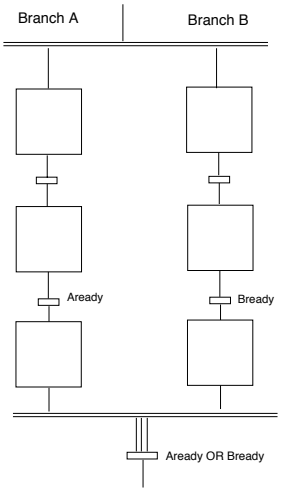

Figure 10. Alternative syntax

3. AN INDUSTRIAL APPLICATION

Grafchart has been used to implement a knowledge-based system that generates on-line advice for operators regarding the distribution of hydrogen resources at the Star Enterprise Delaware City Refinery [10]. The system uses KBS techniques coupled with numerical optimization. The application is an example of the second situation where sequential processes are important. 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 hydrocracker unit consumes high purity hydrogen and vents low purity hydrogen. Hydrogen from these units is used to satisfy the needs of the hydrogen consuming hydrotreaters, sulphur recovery, methanol, and naphtalene units. Any additional hydrogen needs must be met by a hydrogen production unit.

The solution to the problem consists of three sequential steps [10, 11].

- Attempt to recover any suction venting associated with the compressors.
- Determine the best operating policy based on the current operating state.
- Generate advice for the operators on adjust-

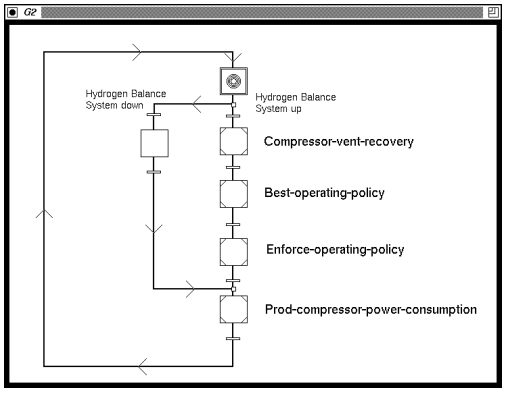

Figure 11. Hydrogen application Grafchart

ments necessary to enforce the optimal operating philosophy.

Each of these steps can be broken down into substeps. The first and the third steps are solved by heuristic rules. The second step employs linear programming to solve a numerical optimization problem formulated by the KBS. In the case a numerical solution cannot be achieved heuristic rules are used as a backup.

The sequential steps of the problem are represented as Grafchart macro steps according to Fig. 11. The entire system contains more than 18 macro steps and over 80 ordinary steps. The system contains numberous alternative and parallel constructs. The reasoning path is traced by changing the colour of a step that has been active. The trace colours are reset each time a new cycle of the main Grafchart in Fig. 11 is started. The main reasoning cycle is executed once every 2 minutes. The program formulates suggested flow rates at two splits in the hydrogen network and also recommends compressor settings (or loadings) for all of the compressors in the network.

The application is currently running on a Sun Sparcstation that is linked to a Foxboro IA process control system. The experiences of the system [10] are that

> *"the use of the Grafcet toolbox is very helpful from both a development and operational standpoint. The various functions within the knowledge base are built in a straightforward fashion by associating rules, formulas, and procedures with Grafcet objects. This makes revision and maintenance of the system possible. Additionally, review of the reasoning path through the network is a clear indication of how the KBS arrived at its advice."*

4. HIGH-LEVEL GRAFCHART

The development of High-Level (H-L) Grafchart is inspired by High-Level Petri nets and object-oriented programming. It was decided to use the Grafcet syntax in favor of a general Petri Net syntax. The reason for this is the wide industrial acceptance that Grafcet already has gained. High-Level Grafchart builds upon the object-oriented implementation of Grafchart by, mainly, allowing specialization of Grafchart element object classes. H-L Grafchart is also inspired by previous work in the area such as [1] and [8].

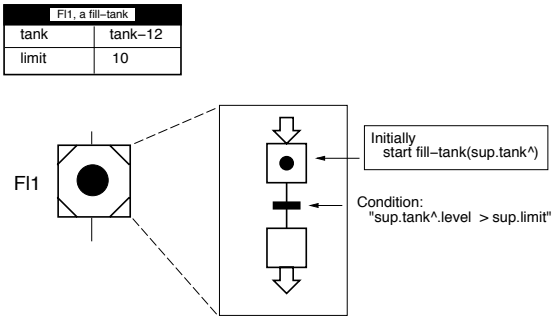

Figure 12. Parameterization

4.1 *Parameterization*

Ordinary Grafchart has no means for parameterization of steps, transitions or macro steps. The rules within a step are specific, i.e., they contains 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 step, transitions and macro steps are objects defined in class definitions. The user can specialize these classes to subclasses in which additional attributes have been added. These attributes act as parameters which can be referenced from rules within the step using simple dot notation. By instantiating the step subclasses with different values of the parameters the step can be reused.

Consider the example shown in Fig. 12. The class named `fill-tank` is a specialization of a macro step with two new attributes: `tank` and `limit`. `Fl1` 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 `tank` attribute contains the name of the tank that should be filled, i.e., the value of this attribute acts as a name reference. 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 using the `sup.tank` notation (sup is short for superior). It refers to the tank referenced by the tank attribute using the Pascal-style notation `sup.tank^`. Similarly the transition condition upon the workspace refers to the level of the tank referenced by the tank attribute and to the value of the limit attribute.

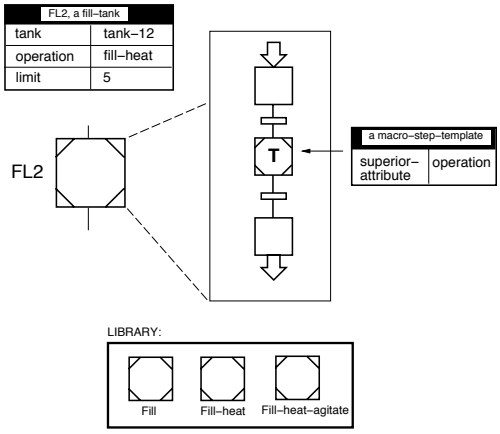

Figure 13. Procedure parameterization

The `sup.attribute` notation is translated and replaced by a corresponding G2 expression during initialization.

4.2 *Procedure Parameterization*

With procedure parameterization it is possible to have entire steps or macro steps as parameters to, e.g., another macro step or an entire Grafcet chart. Procedure parameterization can currently be achieved in two ways: using procedure template objects or using procedure steps.

In the first case the solution looks as follows. In the macro step the step is represented by a template step object that contains a reference to an attribute of the Grafcet process object. During initialization or execution of the network, the template step object is replaced by an instance of the class denoted by the attribute value, i.e., by some step or macro step class.

Consider the small example shown in Fig. 13. FL2 is an instance of a macro step subclass with three attributes. It contains a special macro step template object that contains one pre-defined attribute named `superior-attribute`. The value of this attribute is operation, the name of one of the attributes of FL2. During initialization, the macro step template will be substituted by a copy of the macro step referenced by the operation attribute. In this case the value of `operation` is `fill-heat`. Assume that there exists a library that contains three different macro steps: `fill`, `fill-heat`, and `fill-heat-agitate` which contains different operations that may be performed on a tank. The three macro steps can have totally different internal structure. With the procedure parameterization described, FL2 can be parameterized both with respect to which tank object it should operate on, what the level limit should be, and which operation that should be performed on the tank.

Procedure parameterization is similar to the possibility to have procedures as arguments to procedures in ordinary programming languages. Templates are available for steps, transitions, and macro steps.

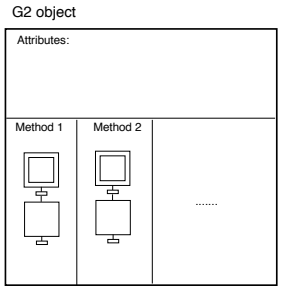

Figure 14. Grafchart methods

The second alternative to procedure parameterization is based on procedure steps. In this case the attribute that determines which Grafcet procedure that should be executed when the procedure step becomes active is given by a parameter. The drawback with this approach is that the Grafcet procedure is not reentrant.

4.3 *Function charts as methods*

It is possible to have function charts that are methods of general G2 objects according to Fig. 14. It is possible to call the methods by calling an appropriate function with the object and the method name as arguments. Within the methods it is possible to reference the object attributes using the `self.attribute` notation.

4.4 *Coloured tokens*

In Grafcet a token is merely a boolean indicator telling whether a step is active or not. In High-Level Petri nets, however, a token can be viewed as an object that carries information. Since a Grafchart token is implemented as a G2 object that may contain attributes, i.e. carry information, this functionality is very natural also for H-L Grafchart. The tokens in H-L Grafchart can now be viewed as objects that move around in the function chart.

Coloured tokens can be used in many ways. In the simplest case the tokens in a chart are all of the same class. The step actions and transition conditions can refer to the token attributes using the `inv.attribute` notation. If all step actions and transition conditions only refer to token attributes we get a behaviour that is equivalent to a graphical re-entrant procedure language. Each token represents an invocation of the graphical procedure. It carries the equivalent to the local variables and procedure arguments of a procedure in an ordinary procedural language. The tokens move around in the function chart totally independently of each other.

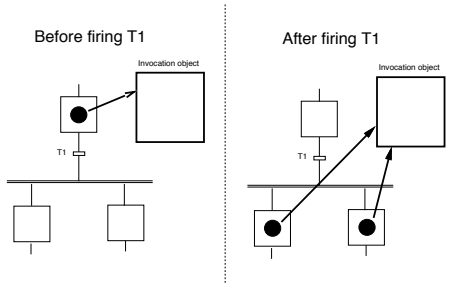

Figure 15. Parallel branches and invocation objects.

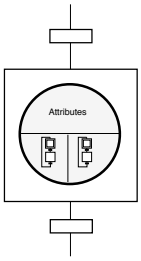

Figure 16. Tokens with Grafchart methods.

In the implementation it is not the tokens themselves that constitute the invocation objects. Instead the tokens act as pointers to the actual invocation objects. This is utilized when handling parallel branches. When the execution splits up into parallel branches, the tokens of the different branches all point to the same invocation object according to Fig. 15.

By allowing step actions and transition conditions to not only be strictly local to the invocation object but also refer to global variables and objects it is possible to, e.g., model systems consisting of objects that have a preserved internal ordering such as elements in a FIFO queue or parts on a moving conveyor belt. This has been used to model a LEGO car factory producing LEGO cars [4].

An extension that is currently under implementation is to allow token objects of different classes in the same chart. In this case it is necessary to have transition conditions, i.e. guards, that refer to the existence of token objects of different classes in the steps enabling the transition. The syntax for this has not been settled yet. It is an open question if arc expressions are needed or if everything can be solved by the guard expressions alone. Avoiding arc expressions would be more in-line with ordinary Grafcet.

Since tokens are objects and objects may contain Grafchart methods it is also possible for tokens to have Grafchart attributes. The structure achieved by this is shown in Fig. 16.

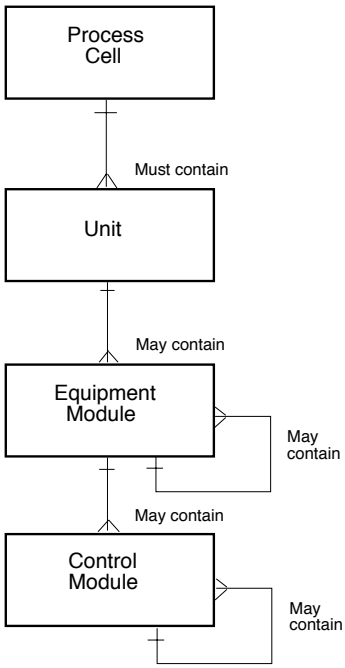

Figure 17. SP88 physical model.

5. SP88 AND H-L GRAFCHART

The forthcoming ISA batch control standards aims at standardizing the terminology, models and functionality of batch control systems. The lower levels of the physical model of SP88 are shown in Fig. 17. The Entity-Relationship notation should be interpreted in the following way. A process cell must contain one or more units. Each unit is only contained in one process cell. An equipment module may contain one or more equipment modules or control modules. A process cell contains all the units, equipment modules and control modules needed to produce a batch. A unit carries out one or more major processing activities such as react, crystallize, etc. An equipment module carries out a finite number of specific minor processing activities such as dosing and weighing. Finally, a control module could, e.g., be a regulating device. This model describes a structural decomposition of the process into hierarchical levels. It can naturally be represented using G2's possibility to have objects with attributes whose values themselves are objects.

The procedural control model is also defined hierarchically as seen in Fig. 18. This model can naturally be described by a hierarchical Grafchart where the entire function chart corresponds to the procedure, macro or procedure steps are used to represent unit procedures and operations and steps are used to represent phases. SP88, however, only mentions Grafcet at the phase level.

The recipe types of SP88 are shown in Fig. 19. Here, we will only consider the master and control recipes. The master recipe is targeted to a process

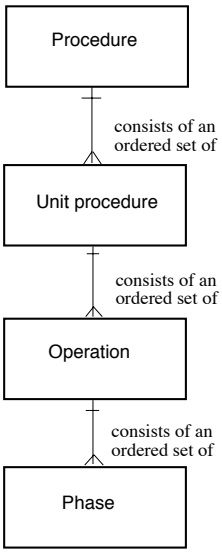

Figure 18. SP88 procedural model.

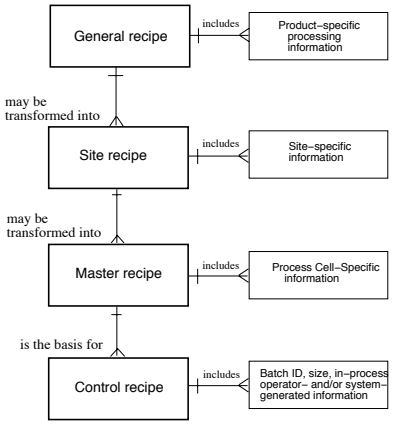

Figure 19. SP88 recipe model.

cell. The control recipe starts a copy of a master recipe that then is modified with scheduling and operational information. The recipes contain header information, formula information about the process inputs, parameters and process outputs, information about the equipment requirements of the recipe, and finally the recipe procedure. A recipe can naturally be represented as an object that contains the recipe procedure represented as a Grafchart method. The recipe procedure is structured according to the procedural model. A recipe phase may contain a reference to an equipment phase. This is a representation of the actions that should be taken in the phase that is stored together with the equipment object, i.e., as a Grafchart method of the equipment object. The recipe phase can be made to execute the equipment phase in several ways in Grafchart. Similar references may be found between recipe operations and equipment operations and between recipe unit procedures and equipment unit procedures.

The possibility to use multiple tokens in a function

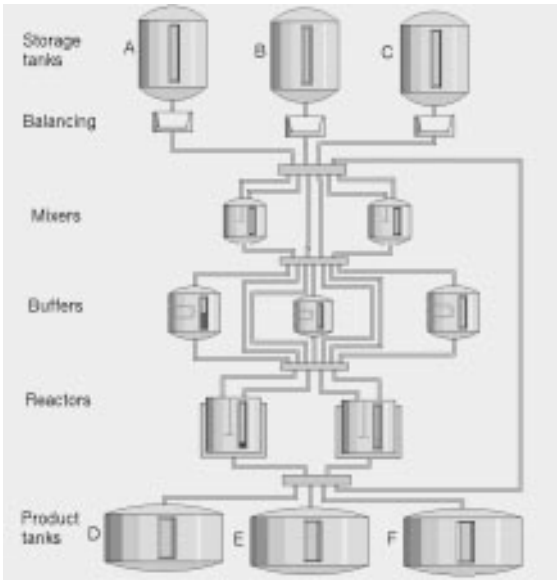

Figure 20. Batch plant configuration

chart can be used in several ways. It is for example possible to have a batch train represented as a function chart and to let the tokens represent the batches currently under production in the train. It is also possible to let the control recipes be represented by Grafchart tokens. These tokens contain the recipe procedure as a Grafchart method. They move between different steps that may represent resource batch preparation, allocation, arbitration, batch processing (here the batch recipe is executed) and inventory management.

The "High-Level Grafcet for supervisory sequential control" project has the goal to implement a batch control system based on High-Level Grafchart. The system is under implementation. The system runs against a real-time simulation of a multi-product, network structured batch process implemented in G2. It consists of storages, mixers and buffers, reactors, and storages according to Fig. 20. It can be operated both in batch, semi-batch and continuous mode. On top of the simulation model the regulatory control system and the batch control management will be implemented.

6. CONCLUSIONS

Combining the function chart formalism of Grafcet with the real-time expert system environment G2 has proved very successful also in industrial practice. The increased parameterization and structuring facilities of High-Level Grafchart further extends the application areas and increases the possibilities for reuse of generic solutions.

Batch control is a supervisory control application where sequence control is very important. The forthcoming SP88 standard is well suited for a Grafchart based representation.

6.1 *Acknowledgements*

This work was supported by the NUTEK REGINA project "High-Level Grafcet for supervisory sequential control". The authors want to thank Bernt Nilsson for many valuable discussions about batch control and modelling of chemical processes.

REFERENCES

[1] S. AGAOUA. *Spécification et commande des systèmes à événements discrets, le grafcet coloré.* PhD thesis, Grenoble University (INPG), 1987.

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

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

[4] K.-E. ÅRZÉN. "Grafcet for intelligent supervisory control applications." *Automatica*, 1994. Accepted for publication.

[5] K.-E. ÅRZÉN. "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*, 1994.

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

[7] K. JENSEN and G. ROZENBERG. *High-level Petri Nets.* Springer Verlag, 1991.

[8] C. LAKOS and C. KEEN. "LOOPN++: A new language for object-oriented petri nets." Technical Report, Department of Computer Science, University of Tasmania, Australia, 1994.

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

[10] T. PETTI and P. DHURJATI. "A coupled knowledge based system using fuzzy optimization for advisory control." *AIChE Journal*, **38:9**, pp. 1369–1378, 1992.

[11] T. F. PETTI. *Using Mathematical Models in Knowledge-Based Control Systems.* PhD thesis, University of Delaware, 1992.