17th European Symposium on Computer Aided Process Engineering – ESCAPE17
V. Plesu and P.S. Agachi (Editors)

Iterative Specification Refinement in Deriving Logic Controllers

Sven Lohmann, [a] Lan Anh Dinh Thi, [a] Thanh Ha Tran, [a] Olaf Stursberg, [b] Sebastian Engell, [a]

[a]*Process Control Laboratory, Department of Biochemical and Chemical Engineering, Universität Dortmund, s.lohmann@bci.uni-dortmund.de*
[b]*Institute of Automatic Control Engineering, Department of Electrical Engineering, Technische Universität München*

Abstract

In this paper the refinement procedure of informal requirements in the context of an earlier proposed systematic procedure for logic controller design as sequential function chart (SFC) is described in detail. The use of two data formats is proposed: dependency charts (DC) and function tables (FT) that support hierarchy and modularization and are refined iteratively until a final degree of detail is reached from which the logic controller as SFC can be generated algorithmically.

Keywords logic controller design, requirements engineering, refinement, hierarchy, sequential function chart

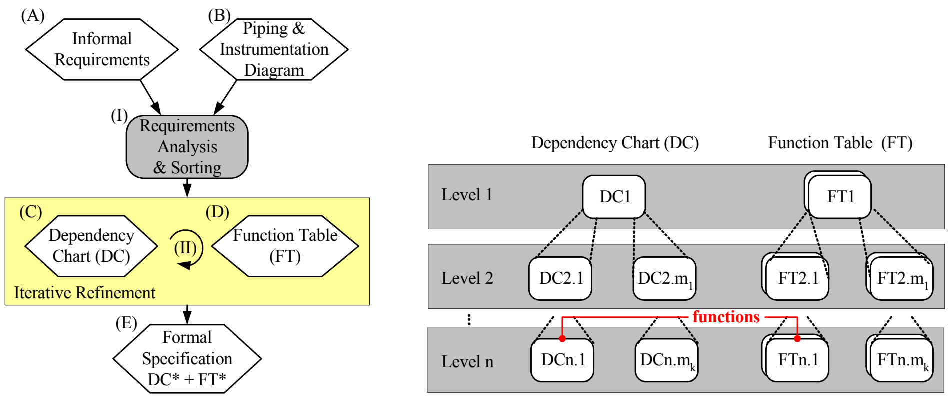
1. Introduction

Automation tasks in today's process industry are characterized by the need for increasingly complex controllers to further increase productivity and to accommodate the market's demand for more and more sophisticated products. The logic controller design is in industrial practice a manual, unsystematic and

error-prone procedure which highly depends on the experience and process knowledge of the designer. In [5] a systematic procedure for logic controller design was proposed where, starting from the informal requirements, a logic controller as SFC [6] is derived using a systematic procedure.
In software design, hierarchy [2], modularization [3] and documentation are well established means to tackle complexity and to ease the task of good design. This contribution applies ideas from software engineering to the problem of logic controller design. The presented work focuses on the aspect of formalizing the natural-language requirements that are introduced and used in the refinement process given initially such that a set of specifications[1] is obtained that can be translated algorithmically into a logic controller as SFC. Two data formats are: Dependency charts (DC) and function tables (FT).
This paper is structured as follows: The refinement procedure in the context of logic controller design and its underlying data formats are described in detail in Sec. 2. The application to an experimental batch plant [7] with requirements that include production sequencing and scheduling as well as procedures for error handling is reported in Sec. 3. Finally, Sec. 4 concludes the paper.

2. The Refinement Procedure

A procedure is proposed here in which the requirements are collected, formalized and systematically refined using two data formats: Dependency charts (DC) and function tables (FT). As shown in Fig. 1.(a), the information that is available for the logic controller design consists of an informal set of



a) Refinement scheme for the requirements   b) Hierarchical development of DC and FT

Figure 1: The refinement scheme using DC and FT starting from informal requirements

[1] The term specifications is used here in contrast to requirements, where the latter refers to an informal description in contrast to a formal one

requirements (A) and a piping and instrumentation diagram (P&ID) (B) of the plant. Before the requirements can be formulated in terms of DC (C) and FT (D), it is required to perform a *requirement analysis* (I) [1]. The data formats DC and FT support hierarchy and modularization and are iteratively refined (II) until finally a degree of detail (E) is reached from which a SFC can be generated algorithmically. Firstly, the requirements are analyzed to ensure that they are correct, complete and precise. Then the requirements are ordered with respect to time, where possible, and separated into those concerning the *nominal operation* of the plant and those concerning *error handling*. Additionally the requirements are sorted into *functional*[2] and *non-functional*[3] requirements. Then the representation of all requirements as FT and DC begins.

The DC describes the interdependencies between different functions (ordinate) over a qualitative time axis (abscissa). Independent functions or groups of functions are described in separate DCs. The building elements of a DC (see Fig. 2) are: *rectangles* which denote procedural functions, *arrows* which denote strict sequential execution, *function connectors* which denote concurrencies, *labels* which denote an order of priorities, and a *terminal point* which denotes the end of the DC. These elements are arranged in a graph where the functions are ordered from top to bottom with respect to time, as far as possible. Alternative branches in the sequence of functions are denoted by multiple arrows originating from the terminal edge of a function rectangle. The last function in the graph is reachable over all paths starting from the initial function.

Each function in a DC is described by an entry in the *function table* (FT) (see Fig. 3). Each entry consists of a number of actions, specified by the identifier of the function (A), the precondition for the execution of an action as described

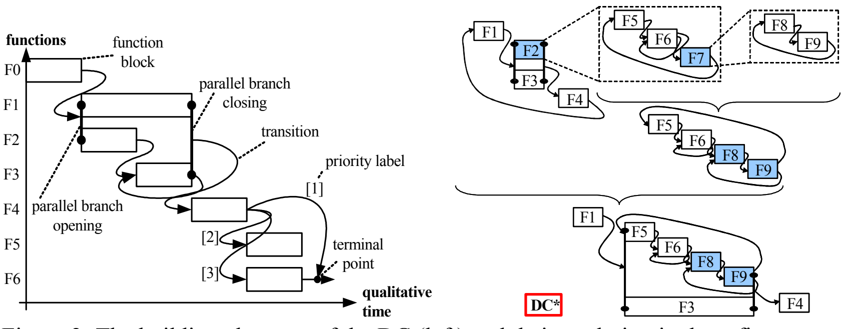


Figure 2: The building elements of the DC (left) and their evolution in the refinement process (right).

[2] statement of some function or feature that should be implemented in a system

[3] statement of a constraint or expected behavior that applies to a system

textually (B), the sensor information corresponding to this condition using only defined sensor or internal variables, formulated as a Boolean formula or inequality (C), the description of the operation (D) carried out, and a list of actions with qualifiers (E) that are defined in [6] and determine when an action is executed. Both data formats are related by the function names (Fig.1.b). For each newly introduced function in the DC, a new function is defined in the corresponding FT.
The starting point of the refinement is the first DC (level 1), called *root-DC* which describes the controller on a coarse level. Fig. 2 (right) shows a root-DC consisting of F1-F4. The functions can be defined freely, however, good practice is to use procedural *basis functions* [8], e.g. *dose* or *temper*. For each root-DC one SFC is created. The set of input (sensor) and output (actuator) variables is taken from the P&ID. Additional internal variables can be declared as needed. Using the DC, the interdependencies between the different functions are defined. Hereby, the line-by-line arrangement of the functions in the graph provides a good overview of the functions. Each function has a defined point in time (event) for its activation and deactivation. The control structure is first described on an abstract level, thereafter the refinement of the DC is done using an hierarchical graphical representation. As illustrated in Fig. 2, F2 is refined by newly defined functions F5-F7. A new function must be defined when a conflict occurs where two or more optional paths in the controller can be selected (loop or alternative branch). To maintain a clear and manageable code, large functions can always be separated into smaller modules.
In parallel to the design of the structure of the controller using the DC, the FT is used for the specification of the details of each function as well as for documentation. During the overall refinement process, each function in the DC has its counterpart as a FT (Fig. 1.b). Fig. 3 shows an example of a FT. The field (A) contains the function name as defined in the DC (e.g.: Fill_R23). The fields (B) and (D) contain explanatory text stating the *precondition* and *operation* of the action. In the early stages of the refinement, the description of the controller is yet too coarse to define suitable sensors or actuators. Therefore

A functions	B precondition	C sensor	D operation	E actuator
F1 action 1 ⋮ action n	textual description of the condition for the action	Boolean formula or inequality statement	textual description of the resulting actions	List of actions: action name + qualifier
Fill_R23	The reactor R23 is empty. The sensor has an accuracy of 10%	LIS23 <= 120	The reactor is filled first with the acid from B12.	S - V123_open
	The reactor R23 is filled half-full	LIS23 >= 4550	Fill level is reached. Acknowledge.	R - V123_open S - Ack_B12
	The filling of acid has stopped. Fill level is acknowledged	LIS23 >= 4550 AND Ack_B12	Fill B23 with base	S - V113_open
	⋮	⋮	⋮	⋮

Figure 3: Example of a function table

(C) and (D) are filled only in the later stage of the refinement process. Each action is directly linked to its explanation, hence each design step can be fully documented.

The requirements refinement is completed if no more functions have to be defined and all preconditions and operations are formalized. The different DCs are taken to form one DC* for each root-DC. The DC* is obtained by inserting refined DC into the coarser ones, level-by-level, starting from the most refined (see Fig. 2 and Fig. 1.b). Corresponding to this DC*, a subset of FT exist that holds the description of all functions appearing in the DC*. This function table is called FT*. Both pieces of data represent the formal specification from which the logic controller can be generated algorithmically [7,8].

3. The Example

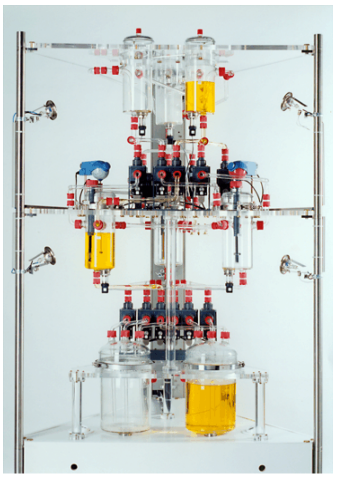


Figure 4: The example plant

The presented methodology was used to design a logic controller for a experimental multi-product batch plant [7] where two different products are produced in a three-train plant (Fig. 4). The plant consists of three buffer tanks on the top level that hold the raw materials, three reactors on the mid-level and two storage tanks on the bottom level. Each vessel of a level is connected with every vessel on the level below. Three different feeds are provided by the buffer tanks (A,B,C). Each product is a mixture of equal parts of two of the feeds (A+B and A+C). The buffer tanks can hold each the feed for two batches and are filled from an external source. Every tank is equipped with a level sensor and the reactors additionally have a mixer attached. A reactor is filled from a single buffer tank at a time. The objective is to use the plant's assets to produce the required amount of batches of each product as fast as possible while considering the possibility that one reactor fails. In this case, the production is continued without interruption because the spare reactor replaces the blocked part of the plant immediately. The resulting SFC comprises 25 inputs and 23 outputs. In the design process, 26 internal variables and 83 different functions were defined over all levels of detail which led to a logic controller with 111 states and 148 transitions in three SFCs. The correctness of the controller was verified using a timed automata [4] model. The SFC was implemented on a programmable logic controller (SIMATIC-PLC S7-300) using the Step7 programming tool and all findings were confirmed experimentally at the real plant.

4. Conclusions

The requirements refinement phase of a systematic logic controller design procedure that uses well-defined data formats, incorporating a documentation of each design step, was discussed. The step-by-step refinement of the specification, in which the structure of the controller is designed using DCs and the straight sequences are specified using FTs, accommodates the need for thorough and methodical LC-design. Hierarchy and modularization are the means of choice to keep the description clear and accessible. Eventually, the final degree of detail of the specification is reached and the fully documented design can be translated algorithmically into the SFC controller. The time needed for the design can be considerably shortened while providing a well-documented and well-structured logic controller, which will result in lower investment and operational costs. Maintenance works, smaller or bigger modifications which are common for the long life-cycles of chemical plants are improved. Changes are introduced by modifying the specification not the control code directly in order to keep the documentation consistent with the SFC. To check whether the operation of the logic controller does result in the intended plant behavior can be done at a subsequent stage; e.g. in a subsequent verification of the controller using a formal plant model [4,5].

References

1. Kotonya, G.; Sommerville, I.: *Requirements Engineering - Processes and Techniques*. John Wiley and Sons, 282 p., Chichester, 2002.
2. Brooks, F.P.: *No Silver Bullet - Essence and Accidents of Software Engineering*. Information Processing 1986, H.J. Kugler, Ed., Elsevia Science Publishers B.V. (Holland) IFIP 1986.
3. Parnas, P.L.: *On the Criteria to be Used in Decomposing Systems into Modules*. Communications of the ACM, Vol.15, No.12, 1053-1058, 1972.
4. Lohmann, S.; Stursberg, O.; Engell, S.: *Comparison of Event-Triggered and Cycle-Driven Models for Verifying SFC Programs*. Proc. American Control Conference 2007, 2007 - 11.-13.07. 2007, New York, accepted.
5. Lohmann, S; Dinh Thi, L.A.; Stursberg, O.: *Design of Verified Logic Control Programs*. Proc. IEEE International Symposium on IEEE International Conference on Control Applications, Munich, 2006.
6. Int. Electrotechnical Commission: *Programmable Controllers - Programming Languages*. Standard IEC 61131-3, 2003.
7. Bauer, N.; Kowalewski, S.; Sand, G.; Löhl, T.: *A case study: Multi product batch plant for the demonstration of scheduling and control problems*. Proc. 4th Int. Conf. on Automation of Mixed Processes: Hybrid Dynamic Systems/ Shaker, Dortmund, p.383-388, 2000.
8. NAMUR: Recommendation NE33 - Requirements to be met by systems for recipe-based operations. 2003.