Chinese Journal of Chemical Engineering, **16**(1) 43—47 (2008)

Systematic Logic Controller Design as Sequential Function Chart Starting from Informal Specifications*

Sven Lohmann** **and Sebastian Engell**
Process Dynamics and Operations (dyn.), Department of Biochemical and Chemical Engineering, Technical University Dortmund, 44221 Dortmund, Germany

Abstract Today's automation industry is driven by the need for an increased productivity, higher flexibility, and higher individuality, and characterized by tailor-made and more complex control solutions. In the processing industry, logic controller design is often a manual, experience-based, and thus an error-prone procedure. Typically, the specifications are given by a set of informal requirements and a technical flowchart and both are used to be directly translated into the control code. This paper proposes a method in which the control program is constructed as a sequential function chart (SFC) by transforming the requirements *via* clearly defined intermediate formats. For the purpose of analysis, the resulting SFC can be translated algorithmically into timed automata. A rigorous verification can be used to determine whether all specifications are satisfied if a formal model of the plant is available which is then composed with the automata model of the logic controller (LC).
Keywords logic controller design, systematic approach, informal requirements, verification

1 INTRODUCTION

In many processing as well as in manufacturing plants, the largest part of the automation software is represented by logic control functions that supervise and coordinate local controllers, establish sequential procedures, and ensure a safe operation and high productivity. However, in most cases, the design of logic controllers does not follow a rigorous design pattern based on precisely defined models—rather the informal requirements are translated manually [1] into the commonly used programming languages (according to the standard IEC 61131-3 [2]). The aim of this article is to present a method that: (a) systematically transforms the given requirements into formal specifications using clearly defined data formats, (b) algorithmically translates these specifications into a logic controller as sequential function chart (SFC) [3], (c) documents every design step, and (d) enables a rigorous functionality check through algorithmic analysis incorporating a plant model.

Figure 1 provides an overview of the different representations and models involved in the procedure, which is separated into three parts: (I) logic controller design, (II) translation into timed automata (TA) [4], and (III) algorithmic analysis. The starting point is the complete set of informal operational and safety-relevant requirements, a piping and instrumentation diagram (PID), and applicable standards and guidelines (1). The requirements are subsequently refined and formalized using two data formats: dependency chart (DC) and function table (FT). In the first step, the intended process behavior is described coarsely with processing functions [5] using a dependency chart (DC) (2). The DC determines the temporal relation between different functions, as for example, required sequential or parallel execution. To each DC a function table (FT) (3) is assigned that describes each function of the DC in more detail with a set of actions, each of which

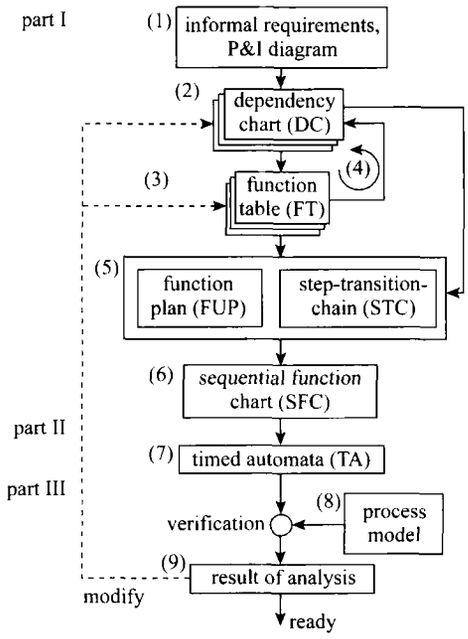

Figure 1 Scheme of the systematic logic controller design procedure

represents an operation that has to be executed when a precondition applies for the relevant sensors and actuators. Both data formats DC and FT are iteratively refined (4) until a final degree of detail is reached. Then the two data formats are algorithmically translated into the SFC (6) using two intermediate formats: function plan (FUP) and step-transition chains (STC) (5).

To analyze whether the controller does satisfy the requirements, formal verification is applied: the SFC is transformed into communicating timed automata [6] (TA) (7), composed with a plant model (given as timed or hybrid automata) (8), and analyzed by model checking [7]. If all requirements are satisfied, the SFC controller can be implemented on a programmable

Received 2007-05-10, accepted 2007-10-27.
* Supported by the European Union through the Network of Excellence Hybrid Control (HYCON) under contract IST-511368.
** To whom correspondence should be addressed. E-mail: sven.lohmann@bci.tu-dortmund.de

logic controller (PLC)—otherwise the analysis result is used to identify in which design steps modifications have to be introduced (9).

Alternative design methods, for example, Lee and Tilbury [8] proposes a systematic procedure but aims at deriving logic controllers as ladder diagrams. The methodology by Frey and Wagner [9] proposes an algorithmic controller synthesis based on a model of the controlled plant formulated as Petri-Nets. This approach supports the verification of properties based on Petri-Nets. Besides Petri-Nets is not an official programming language for programmable logic controllers. The disadvantage of both approaches is that they do not incorporate the documentation of each design step.

2 SEQUENTIAL CONTROL OF A BATCH-EVAPORATION SYSTEM

Figure 2 shows the flowchart of the batch evaporation system for which the design procedure was carried out. The system consists of two tanks (T1, T2) with heating devices (H1, H2), a condenser with cooling (C1), connecting pipes equipped with valves (V1, V2, V3), and a pump (P1). In addition, the plant contains different sensors for liquid levels (LIS), temperatures (TI), and the concentration in T1 (QIS). The desired operation is to evaporate liquid from a mixture in T1 until a required concentration is reached, to collect three batches of the product in T2, and to empty the latter afterwards through P1.

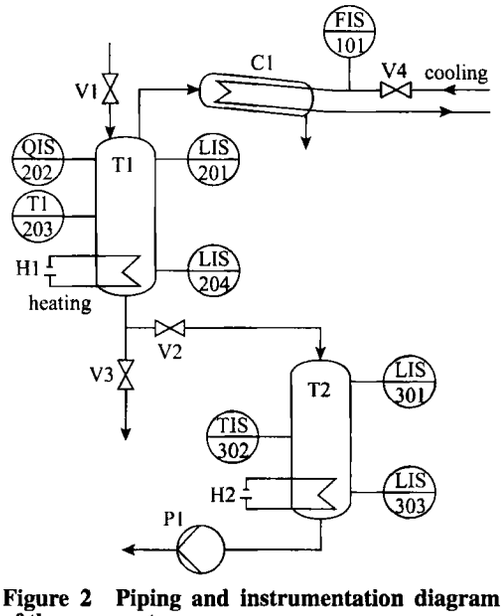

Figure 2 Piping and instrumentation diagram of the evaporator

Two different disturbance scenarios are considered: (a) in case of a cooling failure (detected by FIS101, error 1), the evaporation is continued for a short period of time and, if the concentration target is not reached by then, the remaining content of T1 is disposed through V3; (b) in case of a heating failure, T1 is emptied immediately through V3 (because the production goal cannot be reached anymore). In both cases the nominal operation is resumed if the faulty devices are repaired or replaced. Two critical states can be identified: a continuing heat supply for scenario (a) may lead to overpressure in the condenser, and for scenario (b), the content of T1 crystallizes when a lower temperature bound is reached.

3 DATA FORMATS OF THE DESIGN PROCEDURE

The proposed method aims at designing logic controllers as SFC, a graphical high-level programming language that supports hierarchy and modularization, in which any of the other languages for logic control programs defined in the standard [2] (instruction list, function block diagram, ladder diagram or structured text) can be embedded. The building elements of an SFC are steps with an attached action block, transitions with associated conditions, parallel executions enclosed by double horizontal lines, and exclusively alternative executions marked by single horizontal lines. The vertical lines represent the control flow starting from top to bottom, with the exception of loops that point in the opposite direction, jumps lead to specified steps directly. One action qualifier is assigned to each action and defines when the corresponding action is executed and for how long. There are time-dependent and time-independent qualifiers. For the methodological approach, a clearly defined syntax and semantics of SFC are necessary, which are not provided by the standard [2], but were specified in a rigorous manner in [3].

The information basis for the method is the set of all relevant safety and production-related requirements that are usually formulated in natural language and the PID. As the PID adheres to certain standards and contains the plant components and all actuators and sensors involved, the natural language requirements are usually characterized by a varying degree of detail, ambiguity, and incompleteness, making the direct translation into programmable control code difficult. First, the requirements are analyzed [10] and divided into requirements for the nominal operation and for error handling.

Then the first dependency chart, called root-DC, is created by the designer defining the temporal dependencies between processing functions [5]. The root-DC describes the complete process with freely defined processing functions on a coarse level for a deductive refinement approach. The functions are shown as rectangles over an abscissa that represents time in a qualitative fashion. Graphical symbols as shown in Fig. 3 denote: strict sequential execution (single arrows), strictly parallel execution (vertical bold lines connecting the beginning and/or ending of two or more functions), or the alternative execution of functions (multiple arrows). Along the ordinate all functions are listed in chronological order (where applicable) from top to bottom. The root-DC comprises the functions F1–F4. The refinement of function F2 is shown by the functions F5–F7 in a new DC at a higher level of detail. The refinement is done step-by-step until all requirements are refined to that degree at

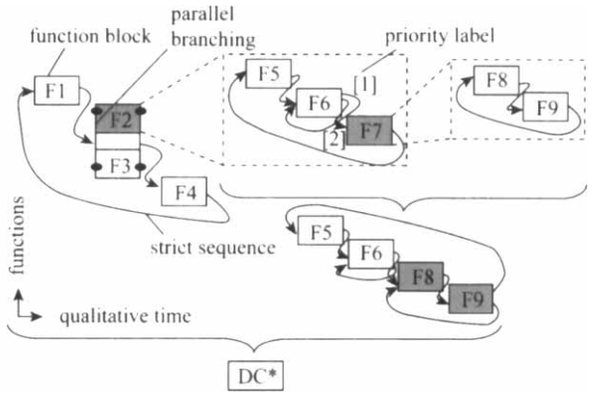

Figure 3 Example of a dependency chart

which they can be formally described. Parallel to the DC the FT is developed.

To each DC, one FT is assigned that details every function that occurs in the DC by a set of actions. An example for a FT is shown in Fig. 4. Each line in the table represents one action (not to be confused with the SFC-action). Each line is divided roughly into three parts: (A) the function name of the correlated DC-function, the precondition part (B+C), and the operation part (D+E). The precondition to each operation consists of a textual, documenting part (B) and formal description of that same precondition by a Boolean function or inequality (C). The operation part is likewise divided into a textual part (D) and a formal part that comprises a list of SFC-actions paired with the respective action qualifier (and optional time constant for time-dependent qualifiers) (E). Thus, each function is detailed by a sequence of actions separated by conditions.

The DC and the FT are refined iteratively until all requirements are considered and a suitable level of detail is reached. Subsequently, FT and DC are transformed into the SFC using two intermediate formats: function plan (FUP), and step-transition-chain (STC). The FUP is generated from the DC and represents the structure of the SFC. The syntax of the FUP is similar to that of the SFC except that there are neither action blocks nor transition conditions; a step represents a processing function, and follows the grammar specified for SFC in Ref. [3]. The STC then replace steps of the FUP by sequences of steps and transitions representing the actions listed for the corresponding function in the FT. Each STC consists of the SFC-elements transitions, steps, and action blocks, and an STC always starts with a transition and ends with a step. The transition conditions and contents of the action blocks follow from the fields sensor and actuator of the FT. In the end, the insertion of the STC into the FUP generates the logic controller as SFC, as shown in Fig. 5 for the example of the evaporation system.

4 VERIFICATION OF REQUIRED PROPERTIES

For analysis, the SFC obtained by the above procedure is translated into timed automata (TA) [6]. A suitable transformation scheme considers the cyclic scanning mode of the PLC as well as the action control, controlling the order in which the actions are executed, and the dynamics of the action qualifiers. Two different transformation schemes were compared in Ref.[4] concluding that an efficient execution model helps reduce the computational effort needed for the verification. The controller model can be composed with a TA model of the plant. The composed model is then used to verify whether the production goal is reached while the critical states are never encountered. This means, the critical states, namely over-pressure and crystallization are explicitly modeled in the automaton for the state of aggregation in reactor T1 (see Fig. 6) and these critical (or forbidden) states are never reached for all possible evolutions of the controlled system.

The properties to be checked are formulated using computational tree logic (CTL) formulae. Safety-related properties have to be fulfilled for all possible evolutions of the system. For production-related requirements, it is checked whether the final state (end of

A function	B precondition	C sensor	D operation	E actuator
F1 action 1 ⋮ action *n*	textual description of the precondition	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 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 B12 with base	S-V113_open
	⋮	⋮	⋮	⋮

Figure 4 Example of a function table

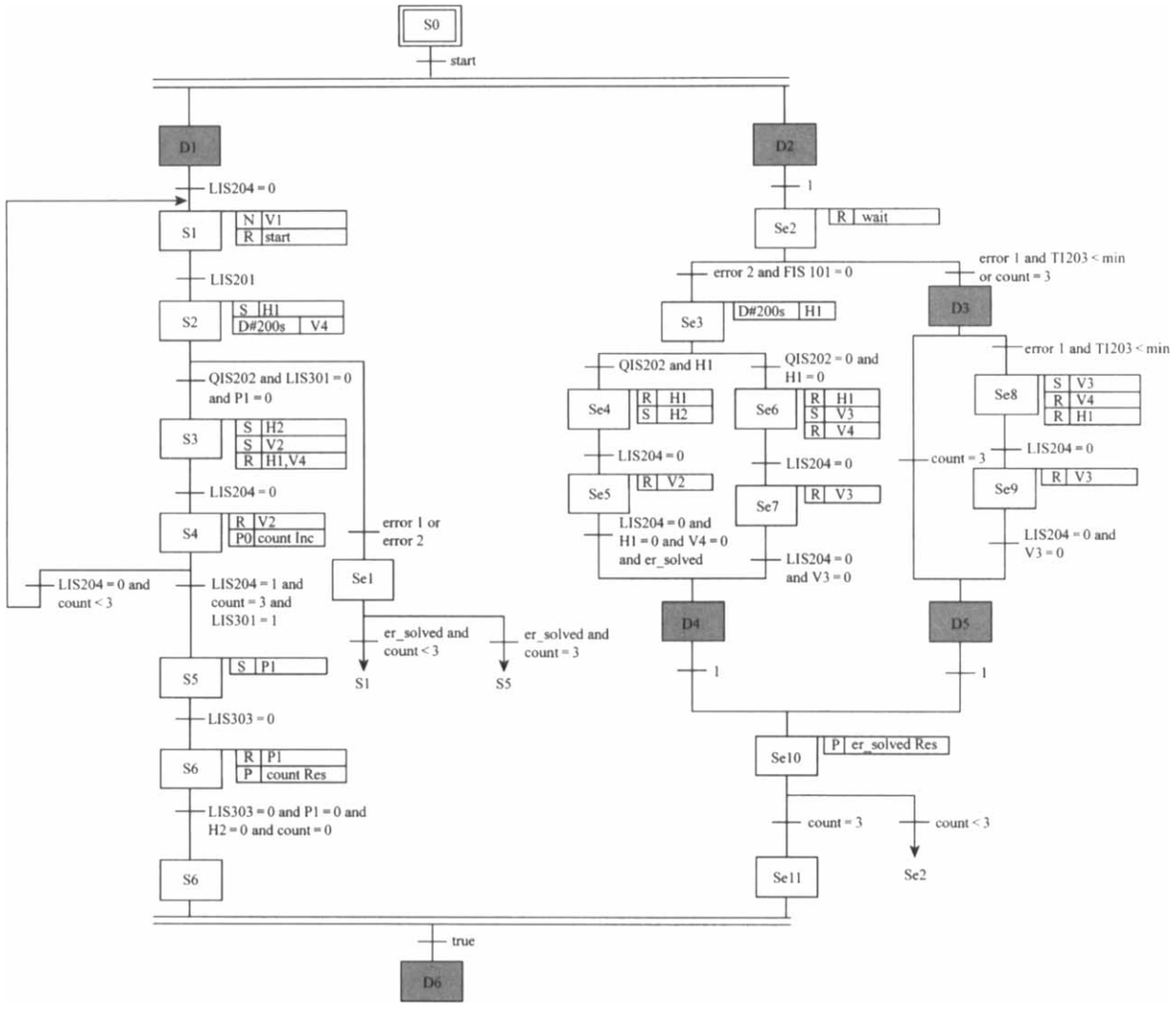

Figure 5 Complete SFC that adheres to the requirements given in chapter 2

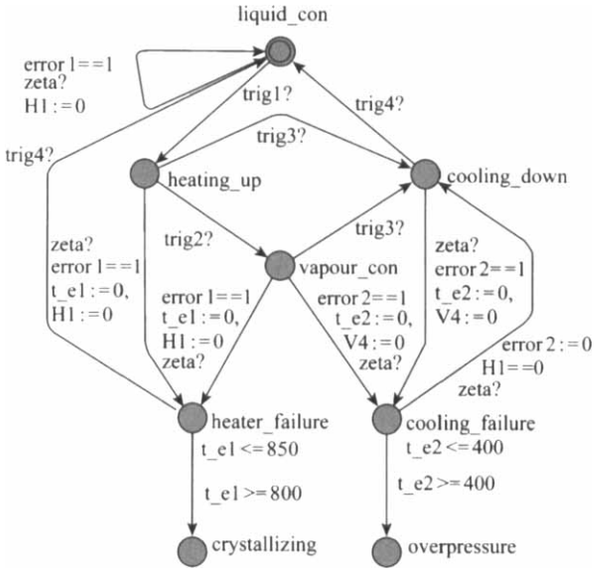

Figure 6 TA-model of the reactant of the process

production) is reachable. The state-space explosion problem [7] usually poses a severe limitation on the total model size. A suitable decomposition of the system and subsequent part-wise verification helps overcome this problem. A suitable abstraction helps focus on crucial specifications and helps keep the model sufficiently small. The verification with Uppaal [11] led to the result that the critical states are not reachable, and that the production goal is met if no breakdown of the cooling system occurs. Thus, the SFC controller shown in Fig. 5 meets the requirements.

5 CONCLUSIONS

A systematic procedure for the design of logic controllers has been presented. A set of intermediate formats is used to proceed step by step from the initial informal requirements to a SFC controller code that can be implemented. The two data formats DC and FT allow for an intuitive and deductive approach to a logic controller design and the documentation of each design step, thus enabling untrained personnel to specify requirements for their processes and plants more clearly and consistently. The systematic design procedure can shorten the development time of a

control system considerably and also minimize the effort for later redesign or modifications because of the complete consistency between documentation and the logic controller code. The life-cycle cost of a logic controller is reduced by the systematic design and tool support is currently being developed.

REFERENCES

1 Lucas, M.R., Tilbury, D.M., "The practice of industrial logic design", In: Proc. American Control Conference (ACC), Boston, 1350-1355 (2004).

2 IEC, Programmable Controllers—Part 3: Programming Languages, International Electrotechnical Commission (IEC), 61131-3, Geneva (2003).

3 Stursberg, O., Lohmann, S., "Analysis of logic controllers by transformation of SFC into timed automata", In: Proc. 44th IEEE CDC/ECC, Sevilla, 7720-7725 (2005).

4 Lohmann, S., Stursberg, O., Engell, S., "Comparison of event-triggered and cycle-driven models for verifying SFC programs", In: Proc. American Control Conference, New York (2007).

5 NAMUR—Recommendation NE 33: Requirements to be Met by Systems for Recipe-Based Operations. http://www.namur.de (last visited: 04/30/2007).

6 Alur, R., Dill, D.L., "A theory of timed automata", *Theor. Comp. Sci.*, **126**, 183-235 (1994).

7 Clarke, E.M., Grumberg, O., Peled, D.A., Model Checking, MIT Press, Cambridge (1999).

8 Lee, S., Tilbury, D.M., "A modular control design method for a flexible manufacturing cell including error handling", In: 44th IEEE Conference on Decision and Control, Sevilla, Spain, 8355-8360 (2005).

9 Frey, G., Wagner, F., "A toolbox for the development of logic controllers using petri-Nets", In: 8th International Workshop on Discrete Event Systems, Ann Arbor, Michigan, 473-474 (2006).

10 Robertson, S., Robertson, J., Mastering the Requirements Process, 2nd ed., Addison Wesley, Boston (2006).

11 Behrmann, G., David, A., Larsen, K.G., "A tutorial on Uppaal", *Design Comp., Comm., Software Systems (LNCS)*, **3185**, 200-236 (2004).