PROGRAMMABLE LOGIC CONTROLLERS

Thiele G., **Renner L. and Neimeier R.**
Institute of Automation, University of Bremen, Germany

Keywords: SW-PLC, embedded PLC, graphic programming language, online task configuration, derived function block, fault-tolerant function block, online parameterisation, feasible computing-sequence, multi-tasking, SW-in-the-loop simulation, IEC 61131, configured fuzzy controller, PLC system-SW architecture, pre-programmed PLC task, object-orientation, FB-class library, IEC 61508, graceful degradation, SW safety, SW certification

Contents

Summary

Section 1 of this article provides an introduction to the conceptual origins of PLCs as a significant class of widely-used industrial controllers employed in process automation, whilst Section 2 discusses some decisive historical aspects of PLC evolution.

Section 3 focuses on graphic programming languages for PLCs, the emphasis being on the latest standard IEC 6 1131-3. Acceptance of special features by PLC programmers and engineers is discussed from the aspect of understandability. A special subsection deals with real-time multi-tasking, including special topics, such as for example online configuration and SW/HW-in-the-loop co-simulation. The in-depth study finishes with PLC SW- and HW-architectures used in professional practice. The paper concludes with a discussion on future trends and perspectives of PLC software, highlighting fault-tolerance and safety.

1. Introduction

Originally, programmable logic controllers (PLCs) were specialized computers for the mapping of hardware relay-logic to software in order to save costs and reduce necessary efforts for modification and maintenance of hardware-logic. After this concept had gained broad acceptance PLCs became soon one of the most commonly used types of automation elements in industry. The main reason behind this was the high acceptance by control engineers who could continue to use their relay-logic designs using a graphic programming language called Ladder Diagram (LD). Figure 1 illustrates the mapping of a hard-wired relay-logic (a) to its LD-counterpart which is based on the related connections of switches (A,B,C) and a coil (D) to digital inputs and an output, using their physical representations %IX1, %IX2, %IX3 and %QX1, respectively (b). In (c) the LD-diagram equivalent to the logic (a) is given with standardized symbols of IEC 61131-3.

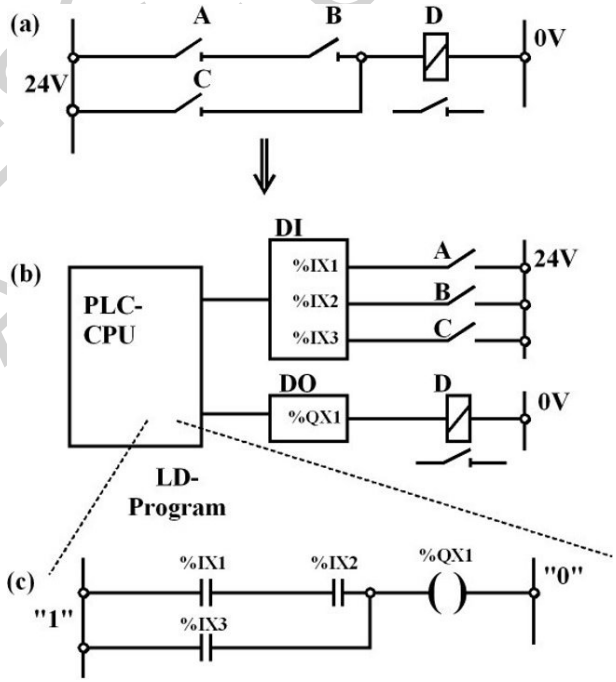

Figure 1. Ladder Diagram equivalent (c) to a given hard-wired relay-logic (a) using a PLC and its digital I/O (b)

The only drawback was that the inputs and outputs of the software-logic could no longer be simultaneously checked and adjusted, but only sequentially “as fast as possible”, because of the finite computing time of the PLC. The consistency of PLC digital input signals during a computation cycle had to be guaranteed by freezing their values in a process-map and by simultaneously updating the PLC output-signals, at the beginning and at the end of each new cycle, respectively. This basic permanent cyclic principle of digital signal processing (see Figure 2) has been used in PLCs up to now, and the increasing efficiency of microprocessors resulted for a long time mainly in the ability of computing more logic in the same cycle time.

It is only since the advent of the first international standard IEC 1131-3 for programmable controller languages in 1993 that the introduction of modern real-time programming principles has been discussed with growing interest also with respect to PLC-software. Although a first step towards industrial application was made in this direction by adding parallel functionality to PLCs for feedback-control, the concurrent task execution of feedback and sequential control tasks, which is common practice with embedded micro-computers and is a forced trend with embedded PCs, did not however have any influence on PLC software design. Therefore, PLC efficiency improvements were achieved essentially only via the hardware.

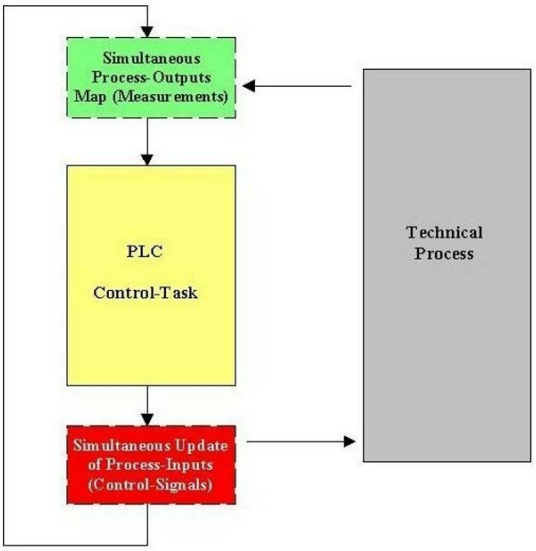

Figure 2. Permanent cyclic execution of a PLC controller task

On the other hand, the range of problems solvable by PLCs increased significantly by extending the PLC software to include additional graphic languages, e.g., Sequential Function Chart (SFC) and Function Block Diagram (FBD) for sequential and feedback control, respectively. The graphic languages once again play a decisive role in the high acceptance of PLCs, however the textual languages ST and IL are also often used. The introduction of the IEC 1131-3 standard resulted once again in an expected "acceptance jump" by PLC engineers and programmers due the significant savings in training which were achieved by unifying the existing proprietary industrial language dialects of different PLC manufacturers.

Apart from conventional permanent cyclic task execution, this standard introduced, for the first time, concurrent task execution in PLCs. Although the latter is professional practice in the field of micro computers and industrial PCs (IPCs), there is up to now only small acceptance of this feature by PLC engineers. This is at least partly due to the complexity of the IEC 6 1131-3 task model. On the other hand however, as known from problem-oriented real-time languages such as PEARL 90, it can be shown that conceptual simplification of this task model is possible, resulting in easier understanding. Simplification is also required in the field of safe real-time software.

Besides specialized PLC systems, the advantages of graphic configuration of real-time software over conventional real-time programming are also available for a broad spectrum of micro- computers and IPCs known as (embedded) Software PLCs. The communication between different embedded PLCs in distributed automation systems, e.g., the mechatronic wheel drives of heavy duty off-road vehicles (e.g., VTT), can also be graphically configured with respect to IEC 61131-5. Communication protocols can also be used for synchronisation of tasks designed graphically as FBD.

In addition to the graphic configuration of conventional controllers in FBD-language like PID (see *PID-control*), self-tuning controllers (see Figure 3) (see *Self-tuning Control*) and fuzzy controllers (see Figure 4) (see *Fuzzy Control Systems*) are also easily configurable.

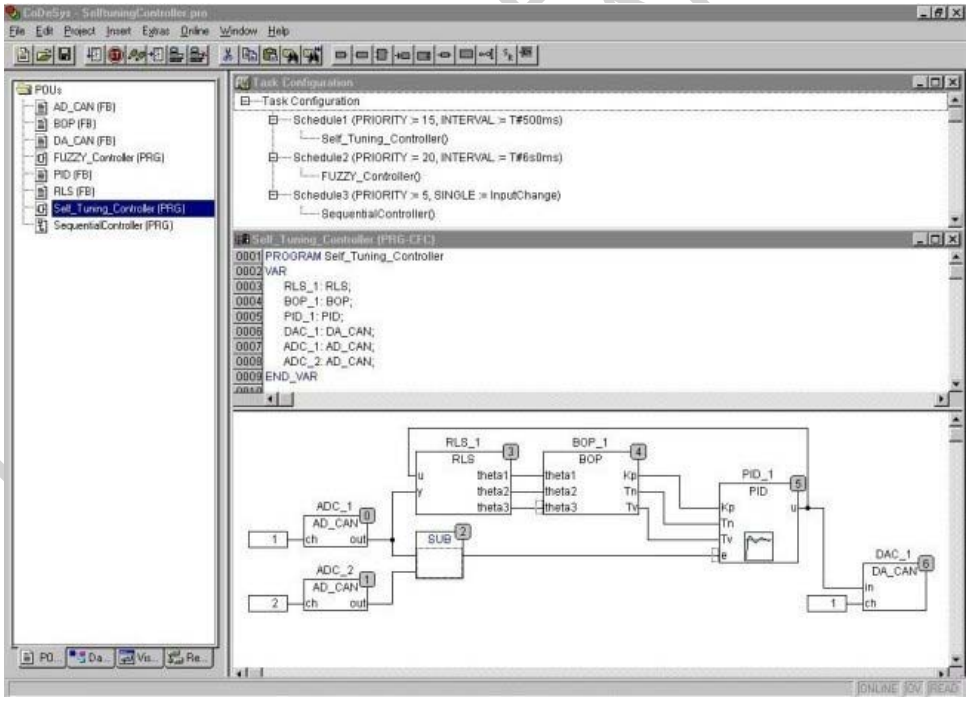

Figure 3. FBD of a self-tuning controller (PRG-CFC Window) scheduled as a 500 ms periodic task of priority 15 (Task Configuration Window) in a CoDeSys system

This is of special interest because of the growing importance of the configuration of real-time software with respect to certification. The verifiability of real-time SW by inspection is, here, a necessary pre-condition, in particular with respect to the demand

for the highest safety integrity levels.

2. Historical Aspects

Up to the sixties, automation in automotive industries was based almost solely on hardware relay-logic for sequential control of production processes. Therefore, each change of design resulted in an expensive and time-consuming implementation. It was this situation that, in 1968, led to the introduction of the concept of a Programmable Logic Controller by Richard E. Morley of Bedford Associates, a manufacturer of industrial controllers. In this context, Morley later on founded the company Modicon (Modular Industrial Controllers).

The first transistor-based PLC implementation was delivered to General Motors (GM) already in the following year. Nevertheless, it took about ten more years, i.e., up to 1977, until the first microprocessor-based PLC was introduced. However, from this time on, the number of PLCs in the chemical industry, for example, grew almost tenfold within five years. A further indicator of significance of PLCs for industrial production is the fact that there were 150 producers of PLCs in 1989.

These facts underline the necessity for standardising PLC programming languages, which had already been anticipated ten years previously when the IEC established the working group WG 7 for PLC standardisation in 1979. It took, though, until 1993 for the WG 7 activities to result in the International Standard (IS) IEC 1131-3 (later on IEC 61131-3 using a world-wide numbering system). Meanwhile there has been a Committee Draft for Vote (CDV) of the Second Edition of IEC 61131-3, which has become International Standard in 2001.

PLCs comprise a broad spectrum of programmable controllers ranging from large and medium size down to micro size PLCs, the latter being preferred for use as embedded components in field-bus oriented automation systems.

The current price for micro PLCs is below US$500, and the price/performance ratio trend is downwards. Besides proprietary hardware PLCs, the advent of the IEC 6 1131-3 standard promoted the break-through of so-called Software PLCs, i.e., automation components with PLC functionality based on standard hardware like PCs or other controller hardware.

3. PLC Programming Languages

3.6. Importance

The conceptual idea of PLCs implies that PLC programming languages play a particular role for the successful PLC application. Furthermore, besides the importance of PLCs for low cost development and modification of sequential controllers, the standard promotes the introduction of modern real-time programming principles in PLC programming for the first time. Nevertheless, low acceptance of these new features by PLC programmers and engineers raises questions to be discussed which are also important with respect to the increasing demand for SW-safety and SW-certifiability by

institutions such as, e.g., the German TÜV (society responsible for monitoring standards).

3.7. Extent of Conformity

The standard IEC 6 1131 introduces two textual and three graphic programming languages. The textual programming languages, being conventionally represented by the Instruction Language (IL) which is also known as the "assembler language of PLCs", have been complemented by the higher-level programming language Structured Text (ST).

On the other hand, the classical graphic language Ladder Diagram (LD) has been complemented by the languages Function Block Diagram (FBD) and Sequential Function Chart (SFC). The certification of the extent of conformity of programming languages of proprietary PLCs with the standard IEC 61131-3 is one of the areas of competence of PLCopen, i.e., TC 3. The members of this organisation comprise PLC and SW manufacturers, TÜV and, recently, also educational institutes.

Their aim is to promote the standard and define recommendations for its use, e.g., for a safe language subset (TC5). In the following, FBD and SFC graphic languages will be discussed in further detail, due to their particular importance and impact on PLC programming.

Graphic languages are distinguished from textual languages by their higher transparency, consequently making them easier to be understood by a broad spectrum of industrial staff, thus achieving one of the main objectives of the IEC 6 1131 standard. This is also a fundamental aspect for safety-oriented applications (see Section 5: Future Trends and Perspectives).

3.8.Function Block Diagram (FBD) Language

3.8.1. Features

In the case of FBD-language, function-blocks (FB) and functions, which together with programs form the so-called program organisation units (POU), are placed as nodes at the desired positions of the respective window. Then they are parameterised and connected by lines, i.e. connection configured. FBDs are, to a large extent, verifiable by inspection, assuming correctness of FBs and functions themselves.

The verification of FBs is facilitated by using high-level languages for their design, e.g. by PLC language ST. If POUs are of low complexity, their formal verification can also be occasionally taken into account. FBs of higher complexity can be created as "derived FBs", i.e. as FBDs from FBs of lower complexity, e.g., in the case of FBs for fuzzy control (see Figure 4) (see *Fuzzy Control Systems*).

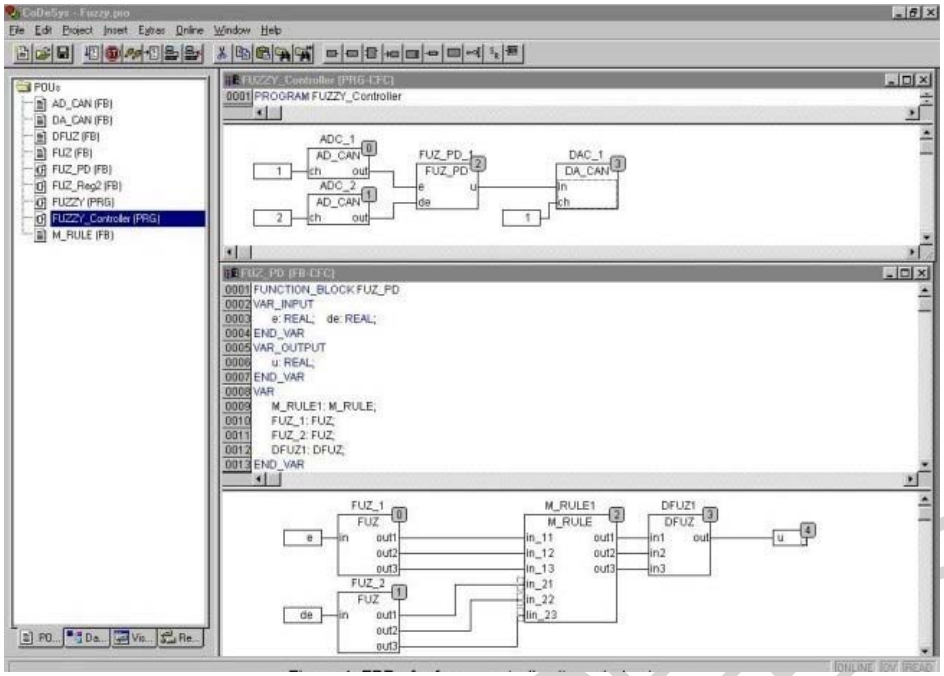

Figure 4. FBD of a fuzzy-controller (top window) including a FB of type FUZ_PD (bottom window) as a "derived FB"

If matrix-rule FBs (M_RULE) of appropriate dimensions are introduced, configuration of fuzzy rule conclusions can be mapped (even online) to parameterisation. If FUZ(zification), M(atrix)_RULE and DFUZ(zification) are used as basic FBs and if connections and problem parameters are correct, the derived FB FUZ_PD approaches (in the sense of cause-effect diagrams) safety-integrity level 4 (SIL 4), as defined in the standard IEC 6 1508.

3.8.2. Function-Blocks as Objects

FBs can be interpreted as SW-ICs, which can be connected via inputs and outputs similarly to HW-ICs. As opposed to HW-ICs, it makes sense in SW to create FB-instances from FB-classes, i.e., abstract data-types, which is a basic idea of object orientation. Implicitly, as introduced in IEC 6 1131, object orientation is also the background of FBs and, even more, of tasks. New FB-classes can be defined in ST. The AD_CAN class definition for an I/O-FB for the fieldbus (see *Bus Systems*) connection with a CAN-based sensor, e.g., reads

```
FUNCTION_BLOCK  AD_CAN
. . .
END_FUNCTION_BLOCK
```

It should be pointed out that the more complete and mnemonically better keywords

```
FUNCTION_BLOCK_CLASS  AD_CAN
. . .
```

END_FUNCTION_BLOCK**_CLASS**

have not been introduced. This may be to avoid the explicit introduction of object-methods in order to ease acceptance by PLC programmers. However on the other hand, due to the absence of explicit pointers in ST, FBs cannot link themselves via input pointers to the output of a connected FB. Instead, the information exchange between FBs has to be organized explicitly by additional instructions between their respective FB-method calls, or by respective parameter lists of these methods, circumventing the OO principle of data encapsulation.

The explicit call of the FB-procedure as an object-method is, furthermore, simplified by using the FB-class name itself (implicitly) also as method name, so that a program segment for executing the FB RLS_1 from Figure 3, e.g., reads in

OO as

CALL RLS_1**.**RLS ;

and

ST(-oriented) as

RLS_1**.**y : = ADC_1**.**out ;

RLS_1**.**u : = PID_1**.**u ;

CALL RLS_1 ;

or as

CALL RLS_1(y:=ADC_1.out; u:=PID_1.u);

3.8.3. Online Parameterisation

The number of additional instructions of the ST-program is furthermore increased by the parameter-input values. This increases not only the complexity of the respective ST-program, but has the disadvantage that a true online FB parameterisation needs the introduction of respective global variables. This could also be made implicitly possible by complementing each parameter-input by a respective parameter-output, which will be menu driven accessible for parameterisation. On default initialisation, each parameter-input pointer points to the respective parameter output. These implicit parameter-input connections have only to be changed if they are to be explicitly connected with other FBs, e.g., in the case of self-tuning control (see Figure 3). This particular design leads to, especially in the case of FBs with a large number of parameters, a significant simplification, e.g., for FBs of FUZ and DFUZ class with respect to the parameterisation of membership functions defining fuzzy sets. Further consequences of the demand for FB-implementation in a pointer-free programming language will be discussed later in the context of tasks.

-
-
-

Bibliography

Beestermöller H. J., Thiele G., Becker J. (1995). Fuzzy-Control with a PEARL-based Multi-Loop Controller. *EUROMICRO Workshop on Real-Time Systems* (Proceedings of the 7th EUROMICRO Workshop on Real-Time Systems, Odense, DK, 1995), (ed. The Institute of Electrical and Electronics Engineers, Inc., IEEE), pp. 66-70. Los Alamitos, CA: IEEE Computer Society Press. [Introduces M_RULE FB, complementing the Fuzzy-FBs by Preuß, see this bibliography].

Bonfatti F., Monari P.D., Sampiari U. (1997). *IEC 1131-3 Programming Methodology*. 377 pp. Seyssins: CJ International. [The books of F. Bonfatti and R. W. Lewis (see this bibliography) are *easy to understand*, structured texts on PLC programming with IEC 1131-3].

International Electrotechnical Commission (1999). *International Standard IEC 61131-3. Programmable Controllers*. Part 3, Second Edition (CDV: TC 65B/WG7/TF3). [Second Edition of the valid IS from 1993, Geneva: IEC. Available from PLCopen, www.plcopen.org].

International Electrotechnical Commission (1998). *International Standard IEC 61508-3. Functional safety of electrical / electronic / programmable electronic safety-related systems*, Part 3: Software requirements. Geneva: IEC. [Defines Safety Integrity Levels SIL 1-4].

John K.H., Tiegelkamp M. (2001). *IEC 61131-3: Programming Industrial Automation Systems*. 376 pp. Berlin: Springer-Verlag. [Represents at present the most actual textbook on IEC 61131 on tutorial as well as professional level].

Lewis R.W. (1995). *Programming industrial control systems using IEC 1131-3*. 293 pp. London: IEE.

Neimeier R., Thiele G., Schulz-Ekloff G., Vielhaben T., Höpfner B. (1998). Object-based Simulation and Implementation of the Control of a Chemical Process with PLC following IEC 1131. *PEARL 98 Real-Time Systems in the Web*, (ed. P. Holleczek), pp. 59-69. Berlin: Springer-Verlag. [A PLC case study of fuzzy control applied to a chemical process with knowledge based verification as well as extensive further literature on fault-tolerant PLC research. In German].

GI-FG 4.4.2 (1998). *PEARL 90 Language Report*. 152 pp. ftp://ftp.irt.uni-hannover.de/pub/pearl/report.pdf. [Defines high-level real-time constructs for "one-shot-tasking" as propagated by K. Tindall (1998). Embedded Systems in the Automotive Industry (Proceedings of the Embedded Systems Conference Europe ESCE, Berkshire, UK, 1998), Vol. 1, pp. 256-287. London: Miller Freeman].

Preuß H.-P. (1993). Fuzzy Control in Process Automation. *International Journal of System Science* **24**, 1849-1861. London, New York: Taylor and Francis Ltd. [Introduces problem-oriented FBs FUZ, RULE and DFUZ for PLC Fuzzy Control of high understandability].

Pullum L. L. (1997). Software Fault Tolerance. *International Symposium on Software Reliability Engineering* (Tutorial Notes of the International Symposium on Software Reliability Engineering, Albuquerque, NM, USA, 1997). [Distinguishes SW Fault-Tolerance from SW-Fault Tolerance. See also: M. R. Lyu (1995). *Software Fault Tolerance*. Chichester: Wiley].

Scharf A. (1989). Programmable Logic Controllers: More Power and Comfort. *Hard and Soft*, **6 (7/8)**, 8-15. Düsseldorf: VDI-Verlag. [Collects the relevant historical information on the foundations of PLCs. In German].

Thiele G., Neimeier R., Renner L., Wendland E., Schulz-Ekloff G. (2001). On the Development of

certifiable fault-tolerant Real-time Software with Function-Block Diagrams for Automation. *PEARL 2001 Real-Time Communication and Ethernet/Internet*, (eds. P. Holleczek, B. Vogel-Heuser), pp. 77-86. Berlin: Springer-Verlag. [Outlines modern PLC-trends in fault-tolerance-oriented software concepts. In German].

Van den Bliek E. G., Molag M., Rouvroye J. L., Brombacher A. C. (2000). Safety integrity: design versus validation. *International Symposium on Programmable Electronic Systems in Safety Related Applications* (Proceedings of the 4th International Symposium on Programmable Electronic Systems in Safety Related Applications, Colonge, 2000), (ed. TÜV Nord, EWICS TC7 and TÜV Rheinland). Colonge: TÜV Rheinland. [Concentrates on the integration of safety in the PLC development phases. The proceedings comprise 26 contributions on experiences with PLCs, especially in safety-related applications].

Biographical Sketches

Georg Thiele graduated in Electrical Engineering in1966 at the Technical University of Berlin and received a Dr.-Ing. in Control-Engineering in 1972 from the Ruhr-University of Bochum. After his post-doctoral thesis for Automation Systems in 1986, he became, in 1996, a Professor at the University of Bremen. He is a member of the GI-WG 4.4.2 Real-time Programming, PEARL, and a member of the DIN NI-22 Programming Languages, and of the Technical Committee TC5 (Safe Software) of PLCopen. His special interests and experiences are mainly in the area of fault-tolerant and safe real-time control software of embedded systems with PLCs, especially Fuzzy Control, and in the identifiability and identification of phenomenological models for real processes, combined with substantial experience in the interdisciplinary research and education in co-operation with the Institute of Applied and Physical Chemistry, University of Bremen. He has made contributions to more than 50 papers, including 3 books, and the DIN standard PEARL 90.

Lothar Renner received the degree Dipl.-Ing. for Engineering Informatics from the University of Applied Sciences of Lemgo in 1976. He was then employed at Siemens AG as a digital communication engineer and changed in 1980 to Krupp Atlas Electronic, Bremen. Here, he was responsible for the development of real-time software in industrial projects, mainly with safety-oriented real-time language PEARL. Since 1984, he has worked with the technical management of the Institute of Automation, University of Bremen. His responsibilities are in the development and application of hardware and real-time software for control-systems with special competence in PLCs and Software-PLCs. He is member of GI WG 4.4.2 Real-time Programming, PEARL. He has made contributions to several papers on real-time programming.

Rolf Neimeier received the degree Dipl.-Chem. at the University of Bremen in 1994. Since 1995, he has been with the Institute of Applied and Physical Chemistry with responsibility in the automation of chemical reactors. In 1998 he joined the Institute of Automation of the Department of Electrical Engineering, University of Bremen, with main interests in safety-relevant and fault-tolerant PLC-controlled processes, which are also the topics of his PhD-thesis. He has made contributions to a number of papers on this subject.