Reengineering Programmable Logic Controllers Using Synchronous Programming Languages

Marcel Christian Werner
Bombardier Transportation GmbH
Mannheim, Germany
marcel.christian.werner@gmail.com

Klaus Schneider
University of Kaiserslautern
Kaiserslautern, Germany
klaus.schneider@cs.uni-kl.de

***Abstract*—The growing complexity of programmable logic controllers (PLCs) implemented in IEC 61131-3 Function Block Diagrams leads to increasing demands on software quality, testing, verification, reusability, and maintenance. The current reengineering approaches focus on transforming existing IEC 61131-3 applications to the newer standard for distributed systems and on applying formal verification methods. This paper discusses whether modeling of existing applications with synchronous programming languages is more suitable than with Function Block Diagrams to meet these requirements. In particular, this paper outlines the potential of code optimization when existing IEC 61131-3 Function Block Diagrams will be modeled using the imperative synchronous programming language Quartz.**

***Index Terms*—programmable logic devices, model driven engineering, real-time systems, software quality, software reuse**

I. Introduction and Background

Programmable logic controllers (PLCs) are embedded systems primarily used as control systems in industrial environments with real-time requirements. Their underlying design and implementation is based on industrial standards such as IEC 61131-3. In this paper, applications implemented in the graphical programming language *Function Block Diagram* (FBD) are considered which already found a wide acceptance [1]. However, the growing complexity of FBDs leads to new challenges in terms of correctness, quality, reuse, and maintenance. Hence, the question arises whether graphical FBDs are suitable to meet those challenges or whether transforming existing FBDs into other models can open new opportunities to meet the mentioned challenges.

This paper therefore investigates the potential of transforming existing FBDs into synchronous *Quartz* programs [2]. To this end, two alternatives are considered: A first one that is based on a completely time-driven *Quartz* model, and a second one that is based on an event-driven *Quartz* model which is a synchronous program whose reactions are guarded by further trigger conditions. Both models are compared by common software metrics, and the potential for code optimizations will be discussed.

A. IEC 61131-3 Function Block Diagrams

A FBD is a sequentially executed state transition system of *Function Blocks* (FBs) [3]. A FB $\mathcal{F}_{\mathcal{B}} = (\mathcal{N}, \mathcal{I}, \mathcal{O}, \mathcal{B})$ is specified by its name $\mathcal{N}$, its input ports $\mathcal{I}$, its output ports $\mathcal{O}$, and its behavior description $\mathcal{B}$ ($\mathcal{B}$ is commonly provided by libraries). A FBD $\mathcal{F}_{\mathcal{BD}} = (\mathcal{F}_{\mathcal{B}}, \mathcal{T}, \mathcal{I}_{\mathcal{P}}, \mathcal{O}_{\mathcal{P}})$ is specified by a finite set of FBs $\mathcal{F}_{\mathcal{B}}$, a set of transitions $\mathcal{T}$ between the FBs, a set of input ports $\mathcal{I}_{\mathcal{P}}$ and a set of output ports $\mathcal{O}_{\mathcal{P}}$.

Although IEC 61131-3 supports an event-driven execution of FBs [4], PLCs can be classified as reactive systems with a cyclic data processing behavior [5]. In each cycle, all inputs $\mathcal{I}_{\mathcal{P}}$ are read, then all outputs $\mathcal{O}_{\mathcal{P}}$ are computed w.r.t. the internal state $\mathcal{S}$, and finally, the internal state $\mathcal{S}$ is updated.

B. Related Work

Transforming existing IEC 61131-3 FBDs into formal models is a research area for more than 15 years. Reference [1] summarizes the existing works until 2015 and shows that most of the transformations are performed in the context of verification purposes. Also more recent papers confirm this focus of research [6], [7]. Related work shows, inter alia, approaches of modeling existing IEC 61131-3 FBDs in the polychronous language *SIGNAL* [8] and languages used by model checkers like *UPPAAL* [9] or *NuSMV* [10]. Furthermore, IEC 61131-3 FBDs can be transformed to the newer standard IEC 61499 and the latter to the synchronous language *Esterel* [11], [12].

C. The Quartz Programming Language

Quartz [2] is a synchronous programming language that was derived from the classic synchronous language Esterel [13], [14]. Compared to *Esterel*, *Quartz* comes with different concepts of signals and causality, delayed assignments, synchronous and asynchronous concurrency of threads and explicit modeling of non-determinism [2]. Several concepts of *Quartz* can also be found in graphical IEC 61131-3 FBDs such as backward flow of information within the same cycle and delayed assignments.

While *Quartz* programs are clearly synchronous programs, we nevertheless distinguish in the next section between time-driven and event-driven *Quartz* programs: A time-driven model is thereby a synchronous program that reacts at every clock tick with some computations, while an event-driven one is a synchronous program whose actual reaction steps depend on further conditions which are evaluated first in a reaction step.

II. Function Block Diagram to *Quartz*

In this paper, arithmetic, logical and FBs with internal time behavior $\mathcal{B}$ are investigated. Specific libraries and nested FBDs

are not considered. The considered FBs can be translated to either a completely time-driven (synchronous) model or alternatively to an event-driven model.

A. Time-driven Models

The transformation of FBDs to completely time-driven *Quartz* programs is straightforward: Even the core statements of *Quartz* are powerful enough to directly represent existing IEC 61131-3 applications. The code is similar to common imperative programming languages with a synchronous model of time to define the reaction steps. It can be used to analyze potential causality conflicts and to analyze a fine-grained level of concurrency within the reaction steps. Moreover, there are code optimizations possible once a FBD is available as *Quartz* code. First experimental studies have shown that core statements for interacting synchronous parallelism can be used to reduce the termination time for cyclic data processing,

B. (Semi) Event-driven Models

The objective of a semi event-driven *Quartz* model is an execution scheme which avoids unnecessary computational overhead by associating each FB with a trigger condition σ (a boolean expression). The FB is only executed if its trigger condition σ (which is evaluated at every point of time) holds. Controlled by σ, the execution scheme may switch between the time-driven and the event-driven execution scheme. The latter can minimize the overhead to recompute the FBs. Nesting both execution schemes inside a thread is possible and needed for example for FBs with internal time behavior.

III. Comparison and future work

Experimental case studies compare both introduced *Quartz* models using the following software metrics: The *Halstead volume* (HV) [15] shows that a minimum size of implementation is needed to set up the introduced semi event-driven model which is more extensive as its time-driven pendant for trivial arithmetic and logical FBs. The *cyclomatic complexity* (CC) [16] of trivial FBs with two logical states in both models is equal. In contrast, the CC of FBs with time dependencies like timer FBs will be reduced, but CC of trivial arithmetic FBs with one logical state significantly be increased. Under the assumption that the *source lines of code* (LOC) represent the number of lines with at least one physical and at most one logical statement, the LOC of time-driven and logical FBs can be reduced by the semi event-driven model. In contrast, the LOC for trivial arithmetic FBs will be significantly increased. The *maintainability index* (MI) of both models is calculated in consideration of CC, LOC and HV [17]. The MI of trivial logical and arithmetic FBs will be impaired by the semi event-driven model and be improved for more complex FBs like timer FBs. Furthermore, the semi event-driven model can reduce the number of micro steps per macro step and the overall recomputing overhead. The time-driven model comes with a significant recomputing overhead in each macro step when the conditions for recomputing the FBs have not changed.

Since *Quartz* allows concurrency of time-driven and semi event-driven threads, the introduced models lead to the research question how existing IEC 61131-3 FBDs can be analyzed and optimized after they have been transformed into *Quartz*. To answer this question, in a next step, the FBs of a IEC 61131-3 library will be transformed to *Quartz* and the code optimization using core statements will be investigated. Based on the optimizations on FB level, the overall optimization potential of existing IEC 61131-3 applications and *Quartz* models will be investigated.

References

[1] S. Rösch, S. Ulewicz, J. Provost, and B. Vogel-Heuser, "Review of model-based testing approaches in production automation and adjacent domains – current challenges and research gaps," *Journal of Software Engineering and Applications (JSEA)*, vol. 8, no. 9, pp. 499–519, 2015.

[2] K. Schneider, "The synchronous programming language Quartz," Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, Internal Report 375, December 2009.

[3] J. Yoo, E.-S. Kim, and J.-S. Lee, "A behavior-preserving translation from FBD design to C implementation for reactor protection system software," *Nuclear Engineering and Technology*, vol. 45, no. 4, pp. 489–504, August 2013.

[4] K. Thramboulidis, "IEC 61499 vs. 61131: A comparison based on misperceptions," *Journal of Software Engineering and Applications (JSEA)*, vol. 6, no. 8, pp. 405–415, August 2013.

[5] B. Beckert, M. Ulbrich, and B. Vogel-Heuser, "Regression verification for programmable logic controller software," in *International Conference on Formal Engineering Methods (ICFEM)*, ser. LNCS, M. Butler, S. Conchon, and F. Zaïdi, Eds., vol. 9407. Paris, France: Springer, 2015, pp. 234–251.

[6] J. Newell, L. Pang, D. Tremaine, A. Wassyng, and M. Lawford, "Translation of IEC 61131-3 function block diagrams to PVS for formal verification with real-time nuclear application," *Journal of Automated Reasoning*, vol. 60, no. 1, pp. 63–84, January 2018.

[7] S. Hun Lee, S. Jun Lee, S. Min Shin, E.-C. Lee, and H. Goo Kang, "Exhaustive testing of safety-critical software for reactor protection system," *Reliability Engineering and System Safety*, vol. 193, pp. 1–15, 2020.

[8] F. Jimenez-Fraustro and E. Rutten, "A synchronous model of IEC 61131 PLC languages in SIGNAL," in *Euromicro Conference on Real-Time Systems*. Delft, The Netherlands: IEEE Computer Society, 2001, pp. 135–142.

[9] D. Soliman, K. Thramboulidis, and G. Frey, "Function block diagram to UPPAAL timed automata transformation based on formal models," *IFAC Proceedings Volumes*, vol. 45, no. 6, pp. 23–25, May 2012.

[10] O. Pavlovic and H.-D. Ehrich, "Model checking PLC software written in function block diagram," in *International Conference on Software Testing, Verification and Validation*. Paris, France: IEEE Computer Society, 2010, pp. 439–448.

[11] C. Sunder, M. Wenger, C. Hanni, I. Gosetti, H. Steininger, and J. Fritsche, "Transformation of existing IEC 61131-3 automation projects into control logic according to IEC 61499," in *Emerging Technologies and Factory Automation (ETFA)*. Hamburg, Germany: IEEE Computer Society, 2008, pp. 369–376.

[12] L. Hsien Yoong and P. Roop, "Verifying IEC 61499 function blocks using Esterel," *IEEE Embedded Systems Letters*, vol. 2, no. 1, pp. 1–4, March 2010.

[13] F. Boussinot and R. de Simone, "The Esterel language," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1293–1304, 1991.

[14] G. Berry, "The Esterel v5 language primer," http://www.inria.fr/meije/esterel/, April 1997.

[15] T. Hariprasad, G. Vidhyagaran, K. Seenu, and C. Thirumalai, "Software complexity analysis using Halstead metrics," in *International Conference on Trends in Electronics and Informatics (ICEI)*. Tirunelveli, India: IEEE Computer Society, 2018, pp. 1109–1113.

[16] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, December 1976.

[17] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44–49, August 1994.