# Complexity- and Performance Analysis of Different Controller Implementations on a Soft PLC

Robert Feldmann

Technion – Israel Institute of Technology | TUM – Technical University Munich
rfeld3@gmail.com

**Abstract.** Whenever code for a Programmable Logic Unit (PLC) is generated using a model based approach there are multifarious possibilities for possible code generators. In order that the generated code is changeable and maintainable the code should have a low complexity. A high complexity complicates by hand changes that become necessary when the code generator is not available as well as it is a reason for high maintainability costs in general [1].
My research is the first to present an approach to compare between PLC implementations generated by different generators in terms of complexity using the example of a paper machine controller. Additionally the performance of the implementations is investigated and a preliminary evaluation of the generators selected is given.

**Keywords.** Model-driven Engineering, IEC 61131-3, PLC.

## 1 Research Problem and Motivation

Model-driven engineering is becoming increasingly popular, especially in the automotive domain, as it can shorten the development time by as much as 50% [2]. In model-driven engineering a model is created based on requirements. These models are created with the aid of mostly graphical, dataflow-oriented languages such as Simulink. Other than in the classical software development process the code is automatically generated from the model with the aid of code generators.

The concept of model-driven engineering is presently also used in the PLC domain. PLCs are appliances used for the automation of technical processes. Programming PLCs is based on the IEC 61131-3 standard, which includes different languages. Code generators for many PLC compatible languages have become available over the last years; three of them can be found below:

| | Code generator | Implementation language | Nature of language |
|---|---|---|---|
| **1.** | PLC coder (Simulink) | IEC 61131-3 /Structured Text (ST) | Textual |
| **2.** | Real-Time-Workshop (RTW, Simulink) | C | Textual |
| **3.** | CFC Code Generator [3] | CFC | Graphical |

C and CFC are both not included in the original IEC 61131-3 standard, but are suited as well and can be regarded as quasi standards.

Programmers that have to implement a model-based PLC application and wish to apply the concept of Model-driven engineering consequently have to choose between 3 distinct code generators that produce 3 implementations in different languages.

The aim of this research is to provide recommendations for which code generator is best. In a showcase, controller implementations in the three languages listed above are generated from a model of a paper machine controller. The implementations are then investigated and compared with regard to complexity and performance on a soft PLC. The conceptual approach is depicted in **Fig. 1**:

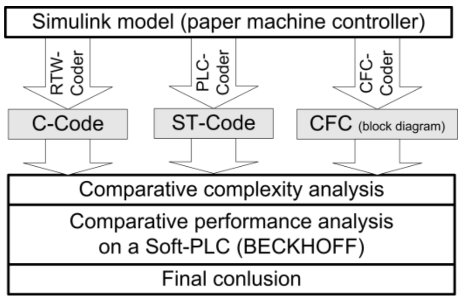
Simulink model (paper machine controller)
RTW-Coder
PLC-Coder
CFC-Coder
C-Code
ST-Code
CFC (block diagram)
Comparative complexity analysis
Comparative performance analysis on a Soft-PLC (BECKHOFF)
Final conlusion

**Fig. 1.** Conceptual approach

## 2 Background and Related Work

This is the first work to compare among various PLC based implementations generated from the same model.

Prior work mainly deals with the quality assessment of standard computer software, but does not explore characteristics of Model-driven engineering or the PLC domain. [4–8] give an excellent overview about the fundamentals of software quality assessment.

[9–11] adapt the concepts presented in the literature above to the specific needs of Model Based Engineering. [12–14] include methods to assess the quality of software in the PLC domain.

This research especially builds upon [10, 12, 8], since the findings of these publications enable comparison of implementations in different programming languages.

## 3 Approach and Uniqueness

The implementations were generated using a Simulink model of a paper machine controller. The model itself has no subsystems and uses 6 different kinds of blocks: Discrete PID controller, discrete transfer function, transport delay, sum, step and outport. There are two input signals and four output signals. Compressed in a subsystem the model looks as follows:

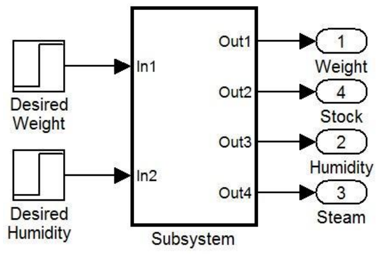

**Fig. 2.** Subsystem including the model of the paper machine controller

Delivering this subsystem as an input to the three code generators returns three implementations: The C-Code, ST-Code and the CFC block diagram are now subject to the following evaluation.

*Complexity analysis.*

Measuring software quality in general implies using software metrics [15]. Metrics are algorithms that map an attribute of a program, such as ‘complexity’, on a numerical scale. Several metrics to measure complexity have been published. The following table shows 3 complexity metrics that are suited for the application on automatically generated PLC-based implementations:

| Complexity metric | Application in the style of | Adaptability to textual languages | Adaptability to graphical languages |
|---|---|---|---|
| Lines of Code | [8] | ✓ | ✗ |
| McCabe | [10] | ✓ | ✓ |
| Halstead | [12] | ✓ | ✓ |

LOC measures the number of code lines. It can be applied on both the ST- and C-Code, but naturally cannot be applied on the CFC block diagram. Like many complexity metrics, the LOC metric is not clearly enough defined. Therefore the version described in [8] is going to be applied, which also allows intercompability between different programming languages.

The McCabe metric measures the cyclomatic complexity. It is based on the control flow graph of a program and determines the linear independent paths through the graph. In this research the cyclomatic complexity is calculated according to [10], which builds upon the original definition of the McCabe metric [16], but extends it such that it is applicable to graphical languages such as CFC.

The original Halstead metric [17] is based on the lexical structure of a program. Starting from the partitioning of the lexical elements into different operands and operators, the program volume can be calculated as a measure for complexity. [12] adapted this metric such that it can also be applied to graphical languages such as CFC.

*Performance.*

To measure performance, the three controller implementations are integrated into the programming environment TWINCAT 2.11, compiled into a Soft PLC and their time response are simulated. It is desirable that the implementation time response is close to the time response of the model, because the more alike the time responses are, the better one can estimate the performance of software in advance. Therefore the time response of the model is also measured and compared to the implementations.

As an input a unit jump was impressed on the entries and received the step function response at the exits. The duration of record was 2000 data points or 2000 seconds at a step time of 1 second.

## 4 Results and Contributions

*Complexity.*
Applying the presented metrics on the three implementations of the paper machine model delivers the following results:

| *Metric* | *LOC* | *McCabe* | *Halstead* |
|---|---|---|---|
| **ST-Code** | **238** | 10 | 2178 |
| **C-Code** | 295 | 10 | 2796 |
| **CFC** | × | **1** | **479** |

According to the LOC metric the ST code performs better than the implementation in C because it requires fewer lines of code. The CFC block diagram has the lowest value for cyclomatic complexity (McCabe) and program volume (Halstead). The implementation in CFC should therefore be clearly preferred with regard to complexity.

*Performance.*
While the implementations in C and ST do not show significant differences between the model, the time response of the CFC block diagram clearly differs from that of the model, which can be seen on the basis of the error depicted in **Fig. 3**:

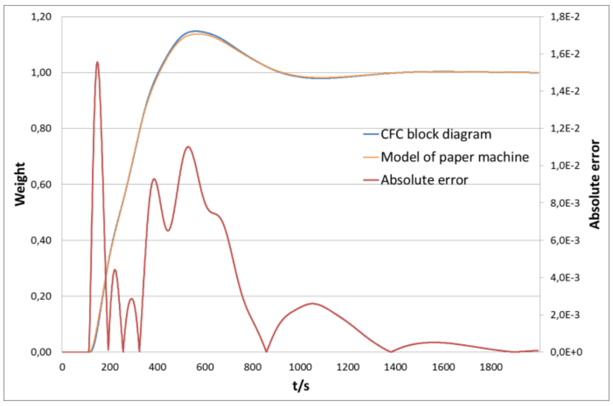

**Fig. 3.** Performance of the implementation in CFC

To quantify this discrepancy the error in least squares sense was calculated:

| **implementation** | CFC | ST | C | Model |
|---|---|---|---|---|
| **e** | 0.03775 | 8.0666E-15 | 0 | 0 (as ref.) |

The implementations in C and ST clearly perform the best, as the error function (nearly) equals zero. The performance of the CFC block diagram clearly is the worst in this assessment. Different execution orders of the Simulink and the CFC blocks are a likely reason for that.

## 5 Conclusion & Future Work

This research presented an approach for the quality assessment of three different automatically generated controller implementations on a PLC. By Assessing complexity and performance of the implementations of the paper machine controller my work shows that Continuous Function Chart performs best in terms of complexity, whereby the implementations in C and ST have the best time response. What was showed exemplarily on a medium sized model with 60 blocks should also be done with other kinds and sizes of models to help validate my findings and extend it to a broader range of models.

**Acknoledgements.** I would like to thank Lindsay Bauer (Manhattan, New York City) for her patience and gratitude to revise my paper and giving me excellent advice on how to express my ideas.

## References

1. Banker RD, Datar S, Kemerer C et al. (1993) Software Complexity and Maintenance Cost. Communications of the ACM 36(11): 81–94
2. Pohlheim, H. Stürmer I. Salecker E. (2012) Ein Ansatz zur Qualitätsbewertung von modellbasierten Entwicklungsprojekten eingebetteter Software. 8. Dagstuhl-Workshop Model-Based Development of Embedded Systems (MBEES 2012): 11–20
3. Bayrak G, Renzhin D, Vogel-Heuser B (2011) Integration of control loops in an UML based engineering environ-ment for PLC. Emerging Technologies and Factory Automation (ETFA)
4. Fenton NE (1991) Software metrics: A rigorous approach. Chapman & Hall, London [etc.]
5. Fenton NE, Pfleeger SL (1997) Software metrics: A rigorous and practical approach, 2nd edn. PWS Pub, Boston
6. Hoffmann DW (2008) Software-Qualität. Springer, Berlin and and Heidelberg
7. Oman PW, Pfleeger SL (1997) Applying software metrics. IEEE Computer Society Press, Los Alamitos and Calif
8. Thaller GE (2000) Software-Metriken: Einsetzen, bewerten, messen, 2nd edn. Verl. Technik, Berlin
9. Thomsen T (2012) MISRA C und seine Anwendbarkeit auf Seriencodegeneratoren. http://www.dspace.de/ftp/papers/dspace_El25_0312_d_f28.pdf
10. Prabhu J (2010) Complexity Analysis of Simulink Models to improve the Quality of Outsourcing in an Automotive Company. http://alexandria.tue.nl/extra1/afstversl/wsk-i/prabhu2010.pdf
11. Kabra A, Karmakar G, Joseph J (2012) ST to MISRA-C Translator and Proposed Changes in IEC 61131-3 Standard. International Journal of Information and Electronics Engineering
12. Stürmer I, Pohlheim H, Rogier T (2010) Berechnung und Visualisierung der Modellkomplexität bei der modellbasierten Entwicklung sicherheits-relevanter Software. Automotive - Safety & Security: S. 69-82
13. Christian Staron (2004) Entwicklung eines Analysewerkzeugs zur Ermitt- lung von Metriken und Qualitätskriterien sicherheitsrelevanter Software im Maschinenschutz, Fachhochschule Bonn-Rhein-Sieg
14. Krell M (2003) Bestimmung von Qualitätskriterien für sicherheitsrelevante Software im Maschinenschutz auf Basis von zertifizierten Industrieanwendungen, Fachhochschule Bonn-Rhein-Sieg
15. ISO 9126
16. McCabe T (1976) A complexity measure. IEEE Transactions on Software Engineering SE-2(4): 308–320
17. Halstead MH (1977) Elements of software science. Operating and programming systems series, vol 2. Elsevier, New York u.a