Meta-Modeling and Automatic Code Generation for Computer Aided Development of Logic Control Systems

Michael Scopchanov, Krzysztof Pietrusewicz, Hristo Hristoskov
West Pomeranian University of Technology, Department of Industrial Automation and Robotics, al. Piastów 17, 70-313 Szczecin

Abstract: This article discusses some aspects of the computer aided development of logic control systems, namely the creation of a meta-model, a domain specific language serving as a base for the system modeling, as well as the formal rules for automatic transformation of the models designed by the experts using this meta-model into simulation models, a source PLC code and the relevant documentation. The problem is defined and proved as important from the point of view of the contemporary industrial software development in order to achieve more readable designs, which are later easier to be modified, to shorten the system development time, to obtain a fast proof-of-concept and to lessen the possibility of having errors in the design. Furthermore, some directions are pointed out regarding the possible future extension of the used techniques towards automatic testing and validation of the developed logic control systems.

Keywords: automatic code generation, domain specific modeling, logic control systems, meta-modeling

1. Introduction

Nowadays the development of a software for industrial applications faces an increasing complexity of problems, including integration of many additional functions, such as safety, communications, visualization etc., next to the "ordinary" control task [19, 6]. Traditional engineering practices, where software is developed and coded without modeling, might be considered obsolete, due to their inability to allow the developer to easily produce an error-free control software that matches the resources of the modern hardware [19]. Furthermore, as engineering costs dominate the project costs of an automation system [6] and the time-to-market should be reduced as much as possible [10, 13], an approach for efficient engineering is even more prominent.

To successfully meet the constantly growing requirements to the capabilities of the industrial control systems, it is important to correctly understand and appropriately apply the following design technologies:

— Domain specific modeling of the control system an important shift in the industrial practice is the adoption of the model-based design [13]. There are two main approaches. The first has been initiated in nineties by creating UML – an universal modeling language that has become a standard for software specification and has influenced the research in software engineering (examples of such research could be found in [19, 11, 16, 7, 12, 3]). One can say that UML has initiated the model-driven software engineering (MDSE). The second is the compositional approach which has grown out of the domain-specific language development. It uses visual specifications as input and is represented by tools like MetaEdit+ and CoCoViLa [18]. Since every field of expertise defines the same control system from a different point of view using its own lexicon, syntax and semantics [4], using the compositional approach tends to be more suitable from the industrial software developer's point of view.
— Automatic generation of the PLC source code and the project documentation
— Out of any doubt, the development of both the domain specific languages and the models themselves requires a certain degree of creativity, hence it could be regarded as a software tool-supported process, in which however, a human plays the central role. On the other hand, the code generation out of the developed models might be done following a formal set of rules, which allows for a complete automation of the sourcing process. This way the computers take the central role in that particular part of the system design [13].

The need for creating a PLC software, which is comprised of reusable components, is emphasized by many authors [6, 19, 7, 2, 20, 3, 5]. Once created, those components could be used in different projects with minimal or no modifications at all. For this purpose it is necessary for the developer to achieve modularity of the software through appropriate structuring of its source code, which could be accomplished applying the principles of object-oriented programming. With regard to this, one

Autor korespondujący:
Krzysztof Pietrusewicz, krzysztof.pietrusewicz@zut.edu.pl

Artykuł recenzowany
nadesłany 9.10.2015 r., przyjęty do druku 19.11.2015 r.

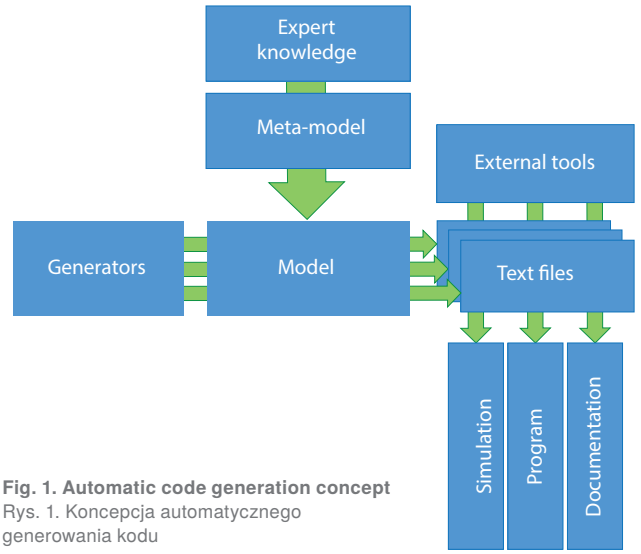

Fig. 1. Automatic code generation concept
Rys. 1. Koncepcja automatycznego generowania kodu

of the major changes in IEC 61131–3:2013 is the addition of an extension that allows functional blocks to have object-oriented features [21]. Furthermore, the ongoing technological innovation now allows the PLCs to handle the communication with the other devices in the automation system, as well as to perform safety and motion control tasks next to the standard control program. Thus, the demand for additional, problem-oriented functions that can be programmed according to IEC 61131 has grown steadily over the last few years. [15]

Another advantage of using meta-modeling and automatic code generation is the increase in the level of abstraction in the system design process [8, 1, 17]. Thus, the developers are able to concentrate on the concept instead of its implementation [14, 2]. However, the generated code might become bulkier, so an optimization phase is also necessary [2]. As a consequence the developer of the target source code generator should be also able to inspect code, understand optimizations and verify low level fixed point bit operations [13].

2. Problem Definition

Figure 1 depicts a concept in which the previously discussed techniques, namely *domain specific modeling* and *automatic code generation*, are utilized.

The knowledge of the experts needs to be extracted and provided as computer readable rules, in a form of a model [6], the structure and semantics of which are defined by a so called *meta-model* [1]. Considering the development of an automation project however, there are typically different engineers or technicians involved with different qualification levels and subjects. Thus, the notation has to be easily recognizable for specialists from a wide range of professional fields, including process, mechanical and electrical engineering. A more visionary requirement is to support the entire life cycle with one consistent model, but appropriate notation for each phase of the project [19]. Those specifics make the domain specific modeling more suitable for the development of an industrial software compared to the utilization of general modeling language.

Once the model is created it needs to be converted to a format which specifies the project's hardware setup, the program code, the associated elements of the graphical user interface, the documentation etc. It is very important that this conversion is accomplished in a way that doesn't imply manual code writing [5]. Instead, a tool is supposed to generate all the necessary files automatically and provide them in such form that they could be directly imported in the correspondent conventional development environment [19].

A commonly used methods and tools for accomplishing some of the aspects of the presented concept are:

- Process specification – various techniques for process specifications are in use today, like automata, cycle time diagrams, flow charts, Grafcet, project network techniques, Petri nets, state charts, SFC, UML–PA, etc. [6, 5, 3];
- Graphical user interface development – several frameworks have been widely accepted for that purpose based on a data flow modeling and a visual programming interface. The best known frameworks with these goals are the LabVIEW and MATLAB/Simulink frameworks. In addition, several other commercial available packages are ready for use to manage the monitoring of distributed processes, namely, supervisory, control, and data acquisition (SCADA) systems [5];
- PLC code generation – the object-oriented extension of the IEC 61131–3 in combination with a CoDeSys UML plug–in, which includes an IEC 61131–3 code generator for UML class diagrams, state charts and activity diagrams, is considered as trend-setting and suitable for industrial environment [3].

In summary, the discussed problem could be stated as: *Definition and software implementation of formal rules for automatic transformation of the expert knowledge provided in the form of a domain specific language based model into a simulation model, a source PLC code and the relevant documentation.*

3. Control System Layering

Having one flat model representing all the aspects of the modeled logic control system is not always convenient. For example, when the project starts to grow bigger in size and complexity such approach may slow down the work and lead to errors. Furthermore, it is harder for several people or group of people to work simultaneously on one model, focusing on their own part. To overcome the mentioned drawbacks some modularity should be introduced, i.e. the model should be divided into layers. The proposed layout of the logic control system is shown in fig. 2.

For each automated process or machine only one hardware configuration layer exists in the model containing all the devices used in the particular project. However, depending on the number of the used programmable logic controllers and the structure of their software, several software configuration layers may be present with their corresponding algorithm definitions and interlock logic.

On each layer only the relevant information should be displayed and be accessible to the modeler. For example on the software definition layer and the layers bellow it no device specific information should be present, hence allowing the work to be performed on a higher, conceptual level, independent from the device vendor and the particular implementation. The goal is to leave as much hardware settings as possible to be done by the code generators, which has the following advantages:

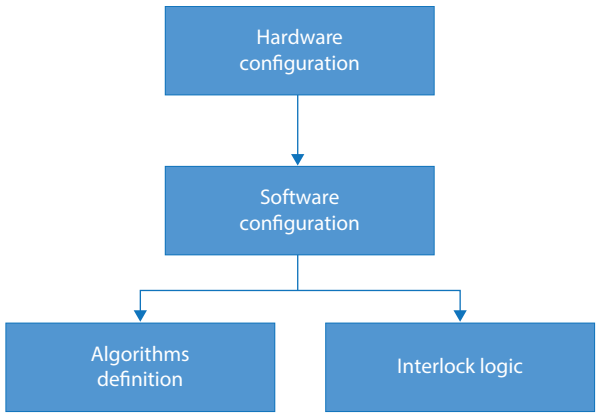

Fig. 2. Logic control system layers
Rys. 2. Warstwy systemu sterowania logicznego

– the less information needs to be entered manually, the less room for error exists and the faster the system could be ready for testing. The last is of particular importance with regard to the rapid prototyping and the fast achievement of the so called *proof-of-concept*;
– the control system could be ported to different PLC types using an appropriate code generator and leaving the already created model intact.

Having the control system represented in such a structured way makes finding information and modifying the model a lot easier in big projects. Moreover, the work could be divided and distributed across several teams, which increases the effectiveness and additionally shortens the time needed for control system development.

4. Meta-Model Concepts

To adequately model all of the aspects of the logic control system, a proper meta-model should be developed first. It starts with the definition of the objects in each layer and their parameters. Then the relations between those objects should be defined as well as the necessary constraints regarding the created models.

All concepts in the meta-model have their corresponding symbols. The examples presented here use custom symbols, built with space reduction and readability improvement in mind. However, since the graphical design is a science by itself, which is beyond the scope of this article, it will not be discussed further.

A. Hardware Configuration

This layer consists of the following objects:
– Sensor – aims to model the physical signals which enter the control system, e.g. the ones from the sensors, switches and buttons. Parameters include the name of the signal, the designation on the schematics, a description and the active logic level;
– PLC – represents a programmable logic controller, the device which executes the control program. The parameters include name and description;
– Actuator – models the physical signals which exit the control system, e.g. the ones sent to the actuators and lamps. Parameters include the name of the signal, the designation on the schematics, a description and the active logic level.

To interconnect the objects on this layer two types of relations are used: wired connection and communication link. The first one has no parameters and the second one is parametrized with respect to the requirements of the selected network type.

B. Software Configuration

The Software Configuration Layer serves as a top-level program model for each PLC present in the hardware structure of the control system (see fig. 3). On this layer the following objects are presented:
– Digital input,
– Sequence,
– Timer,
– Logic block,
– Mode selector,
– Interlock,
– Digital output.

The *Digital input* and *Digital output* objects represent the digital I/O modules of the PLC. The address mapping could be manual or automatic depending on the application. A good idea is to assign a default value for the address of each input and output pin and allow the user to change it later if necessary. The *Sequence* is a model of the control algorithm. It is further decomposed in another layer by its own.

Error monitoring is modeled using *Timer* for the time monitoring and *Logic block* for the state monitoring. The first one has time as parameter and the second one – number and polarity of its inputs.

The *Mode selector* is a switch which enables/disables the direct connection between the inputs from the manual controls and the outputs to the actuators.

The *Interlock* models the internal safety logic preventing the activation/deactivation of particular output when given safety conditions are not met. This object is decomposed in another layer by its own.

To interconnect the objects on this layer only one relation type is used, i.e. wired connection, which has no parameters. There are no restrictions regarding the occurrence of the objects in the model.

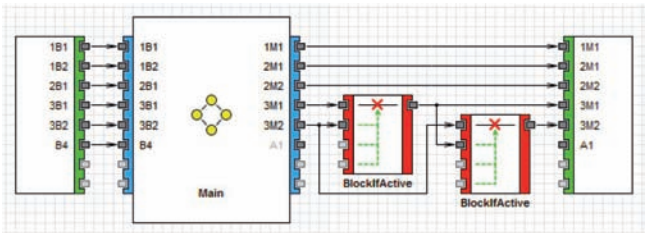

Fig. 3. Example top-level software definition
Rys. 3. Przykład definicji oprogramowania wysokiego poziomu

C. Algorithm Definition

The Algorithm Definition Layer allows the modeler to create the sequences for the automatic mode of operation of the logic control system. Only one type of object is accessible – a state with associated actions list (activation/deactivation of outputs and start of time delays).

There are two types of relations:
– Transition – connects two states designating the conditions under which the transition from the currently active one to the next one should be made. An important restriction here is that two or more relations of this type starting from the same state should not have the same or logically equal conditions;
– Synchronization – introduces an additional condition to be fulfilled for the transition from one state to another to be executed, i.e. when a particular state from another sequence is active. That way a synchronization between simultaneously running sequences could be achieved.

D. Interlock Logic

The purpose of the Interlock Logic Layer is to allow the modeler to define the conditions under which a particular output should remain interlocked in a given state, either "on" or "off". The following objects are used to model this behavior:
– Input,
– Logic AND,
– Logic OR,

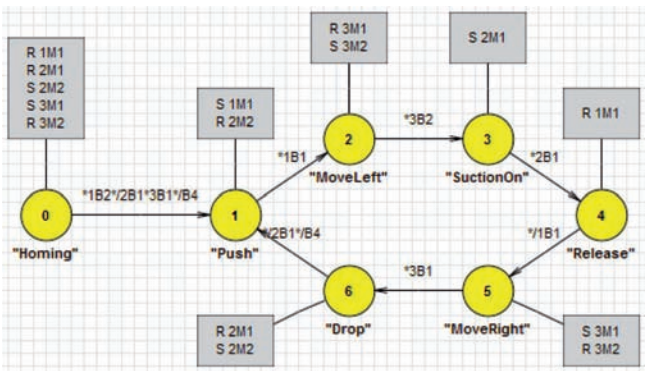

Fig. 4. Example algorithm definition in the form of a state graph
Rys. 4. Przykładowy algorytm sterowania w postaci diagramu maszyny stanów

– Logic NOT,
– Output.

Although in the general case it is a good idea to provide pre-defined logical functions, e.g. flip-flops, next to the basic ones in order to allow the modeler to create any composite logic design with a relative ease, as long as it goes about interlocks the logic is usually not that complex, hence the fully functional basis formed by the AND, OR and NOT functions is completely sufficient and there is no need to be further expanded. However, allowing the modeler to customize the AND and OR functions defining the number of inputs and having the option to invert each of the inputs or the output would make the design more readable, therefore such functionality could be implemented in the software solution as well.

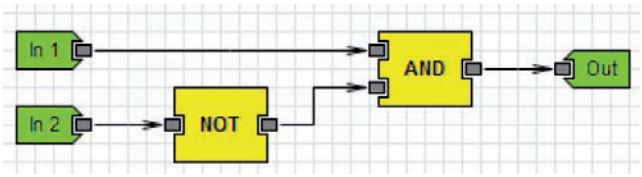

Fig. 5. Example interlock definition using Functional Block Diagram (FBD)
Rys. 5. Przykładowa definicja bloku zabezpieczeń w języku Bloków Diagramów Funkcyjnych FBD

To interconnect the objects on this layer only one relation type is used, i.e. wired connection, which has no parameters. The number of outputs on this layer is limited to one.

5. Automatic Code Generation

Having the control system designed in the previously described manner, a model is created as a set of interconnected objects with their respective parameters. This model contains a general information about the system. However, not all of this information is used for the generation of each target file, but a given sub-set, as shown in fig. 6. In other words different parts of the graphically represented logic control system are transformed following a given set of rules into different text files according to the requirements of the particular software product used for simulation, PLC programming and documentation.

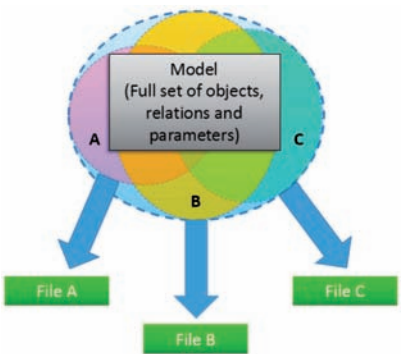

Fig. 6. Using different parts of the model for the generation of each target file
Rys. 6. Wykorzystanie różnych części modelu do generowania innego rodzaju dokumentów

A. Simulation Model

To reduce the development time and costs it is necessary to have a simulation model of the PLC acting as a *software-in-the-loop*. Being widely accepted in the scientific world, Simulink is used as a simulation environment here as well. The built-in logic blocks of Simulink together with the Stateflow toolbox allow the adequate modeling of the program logic. To automatically generate the simulation model, a MATLAB script is produced, containing the corresponding commands for creation and modification of Simulink models, as well as the Stateflow API commands. An example of automatically generated control sequence is shown in fig. 7.

The rules for generating the MATLAB script are pretty straightforward since the discussed meta-model closely resembles the one used by Simulink, i.e. blocks with inputs, outputs and parameters and connections between them. So the generator should take care of adding the corresponding blocks to the simulation model at the positions they have in the designed system, and then parametrize and interconnect them. In turn, the layers of the system are represented as sub-systems.

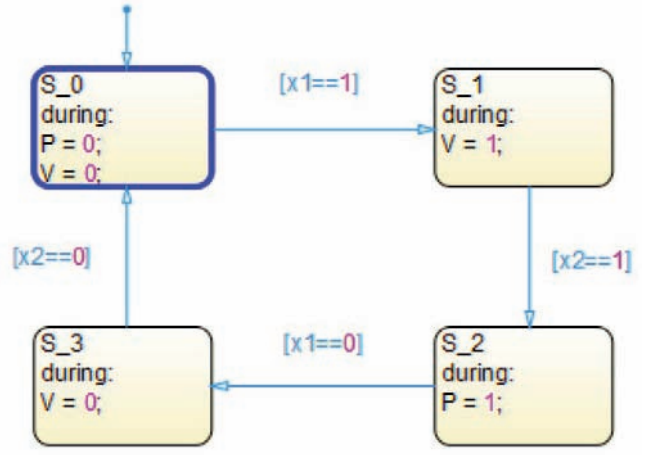

Fig. 7. Example automatic generated Stateflow model
Rys. 7. Przykład automatycznie wygenerowanego modelu Stateflow

B. Source PLC Code

Since the code generators produce text files it is convenient to use Structured Text (ST) defined in IEC 61131–3 [21] as a programming language for the implementation of the software part of the control system. To illustrate the approach Automation Studio by BnR is used.

When generating the code the following set of rules is applied:
– all *Sensors* and *Actuators* used in the Hardware configuration layer are defined as global variables;
– each *State* in every *Sequence* is defined as a global variable in the form M_XX_Step_N;
– for each *Sequence* in the Software configuration layers two corresponding program units are created named *Transitions* and *Actions*;
– each relationship of type *Transition* transforms into lines of code in the corresponding POU with the following format:

```
IF M_XX_Step_N THEN
  IF <conditions> THEN
  M_XX_Step_N+1 := 1;
  M_XX_Step_N := 0;
  END_IF
END_IF
```

– for each *Actuator* defined in the Hardware configuration layer a logical equation is written in the *Actions* program unit containing the OR-ed variables corresponding to the states in which it is activated, the auto/manual switching logic, as well as the defined interlock logic;
– for each *Timer* a program timer is defined and connected to the corresponding alarm output;
– for each *Logic block* in the Software configuration layer a logical equation is added to the *Actions* program unit activating the corresponding alarm output;

The main program then needs to initialize the used variables and to call the created sub-programs.

C. Documentation

To obtain a printable document a TEX-file should be generated and then compiled using pdflatex. The formatting might be fixed in a template which includes the automatically generated parts. That way the form of the document might be changed leaving the actual content intact.

Depending on the particular purpose, the content of the document might vary largely. However, as a general rule it is a good idea to include a symbol table, containing a description of the I/O interface of the control system, as well as tables listing the designed error monitoring functions and interlocks. Figures, representing the hardware design, control software structure and algorithms, should be included in the documentation as well.

6. Conclusion

The discussed techniques play an essential role in the creation of computer tools designed to aid the development of logic control systems. Such tools would allow the creation of well--structured, readable control system designs, which are easier to be modified later. This also shortens the development time, leads to a fast proof-of-concept and lessens the possibility of having errors in the design.

The here presented concepts have been implemented and tested in MetaEdit+ [9]. For practical use however, a standalone application would be a better solution allowing more flexibility to be achieved through an expandable functionality, e.g. towards automated checks for compliance of the design with the relevant standards. With regard to that, in order to close the development cycle in the mentioned way it is necessary to further extend the current work in the direction of automatic tests and validation of the created models, bringing simulation and experimental data back into the software tool for analysis.

Another interesting direction for future research is the integration of human-machine interface (HMI) and dedicated safety devices in the model in order to automatically generate the tags list and the process screens needed for the visualization as well as the safety logic.

Acknowledgments

The subject of this article is part of the project iLoad, *Partnership for developing energy efficient intelligent load handling system*, funded by the European Union Seventh Framework Programme for research, technological development and demonstration under grant agreement No 324496.

Bibliography

1. Baerisch S., *Domain-Specific Model-Driven Testing*, 1st edition, GWV Fachverlage GmbH, 2010.
2. Berruet P., Lallican J., Rossi A., Philippe J., *Generation of Control for Conveying Systems Based on Component Approach*, IEEE International Conference on Systems, Man and Cybernetics, 2007, 1408–1414, DOI: 10.1109/ICSMC.2007.4413766.
3. Braun S., Obermeier M., Vogel-Heuser B., *Usability Challenges in the Design Workflow of Reusable PLC Software for Machine and Plant Automation*, 9th International Multi–Conference on Systems, Signals and Devices, 2012, 1–6, DOI: 10.1109/SSD.2012.6198055.
4. Estevez E., Marcos M., *Model-Based Validation of Industrial Control Systems*, IEEE Transactions on Industrial Informatics, Vol. 8, No. 2, 2012, 302–310, DOI: 10.1109/TII.2011.2174248.
5. Gomes L., Lourenco J., *Rapid Prototyping of Graphical User Interfaces for Petri-Net-Based Controllers*, IEEE Transactions on Industrial Electronics, Vol. 57, No. 5, 2010, 1806–1813, DOI: 10.1109/TIE.2009.2031188.
6. Guttel K., Weber P., Fay A., *Automatic Generation of PLC Code Beyond the Nominal Sequence*, IEEE International Conference on Emerging Technologies and Factory Automation, 2008, 1277–1284, DOI: 10.1109/ETFA.2008.4638565.
7. Iriondo N., Estevez E., Marcos M., *Automatic Generation of the Supervisor Code for Industrial Switched-Mode Systems*, IEEE Transactions on Industrial Informatics, Vol. 9, No. 4, 2013, 1868–1878, DOI: 10.1109/TII.2012.2227491.
8. Jung E., Kapoor C., Batory D., *Automatic Code Generation for Actuator Interfacing from a Declarative Specification*, International Conference on Intelligent Robots and Systems, 2005, 2839–2844, DOI: 10.1109/IROS.2005.1545465.
9. Kelly S., Tolvanen J., *Domain-Specific Modeling: Enabling Full Code Generation*, Wiley-IEEE Computer Society Pr, 1th edition, 2008.
10. Krunic M., Letvencuk I., Povazan I., Krunic V., *An Approach to Model Dri krzysztof.pietrusewicz@zut.edu.pl ven Development and Automatic Source Code Generation of GUI Controls*, 11th International Symposium on Intelligent Systems and Informatics, IEEE, 2013, 63–68, DOI: 10.1109/SISY.2013.6662544.
11. Kundu D., Samanta D., Mall R., *Automatic Code Generation from Unified Modeling Language Sequence Diagrams*, "Software, IET", Vol. 7, No. 1, 2013, 12–28, DOI: 10.1049/iet-sen.2011.0080.
12. Mizuoka K., Koga M., *MDA Development of Manufacturing Execution System Based on Automatic Code Generation*, Proceedings of SICE Annual Conference, 2010, 3103–3106.
13. Mosterman P., *Automatic Code Generation: Facilitating New Teaching Opportunities in Engineering Education*, 36th Annual Frontiers in Education Conference, 2006, 1–6, DOI: 10.1109/FIE.2006.322699.
14. Mozumdar M., Gregoretti F., Lavagno L., Vanzago L., Olivieri S., *A Framework for Modeling, Simulation and Automatic Code Generation of Sensor Network Application*, 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2008, 515–522, DOI: 10.1109/SAHCN.2008.68.
15. Otto A., Hellmann K., *IEC 61131: A General Overview and Emerging Trends*, "Industrial Electronics Magazine", Vol. 3, No. 4, 2009, 27–31, 10.1109/MIE.2009.934793.
16. Papadopoulos G., *Automatic Code Generation: A Practical Approach*, 30th International Conference on Information Technology Interfaces, 2008, 861–866, DOI: 10.1109/ITI.2008.4588524.
17. Santos A., Cardoso J., Diniz P., Ferreira D., Petrov Z., *Specifying Dynamic Adaptations for Embedded Applications Using a DSL*, Embedded Systems Letters, IEEE, Vol. 6, No. 3, 2014, 49–52, DOI: 10.1109/LES.2014.2321325.
18. Tyugu E., Grigorenko P., *Components in Model-Based Software Development*, Computer Science and Information Technologies, 2013, 1–8, DOI: 10.1109/CSITechnol.2013.6710367.
19. Vogel-Heuser B., Witsch D., Katzke U., *Automatic Code Generation from a UML Model to IEC 61131–3 and System Configuration Tools*, International Conference on Control and Automation, Vol. 2, 2005, 1034–1039, DOI: 10.1109/ICCA.2005.1528274.
20. Werner B., *Object-Oriented Extensions for IEC 61131–3*, Industrial Electronics Magazine, IEEE, Vol. 3, No. 4, 2009, 36–39, DOI: 10.1109/MIE.2009.934795.
21. International Electrotechnical Commission – [www.iec.ch].

Meta-modelowanie oraz automatyczne generowanie kodu w projektowaniu komputerowym logicznych systemów sterowania

Streszczenie: W artykule omówiono wybrane aspekty rozwoju komputerowego wspomagania systemów sterowania logicznego, a mianowicie: utworzenie meta-modelu oraz języka konkretnego obszaru zastosowań, będącego podstawą do modelowania systemu, jak i formalnych zasad automatycznego przekształcenia modeli zaprojektowanych przez ekspertów za pomocą meta-modelu do modeli symulacyjnych, kodu źródłowego dla sterowników PLC oraz odpowiedniej do potrzeb dokumentacji. W artykule zaprezentowana jego istotność z punktu widzenia tworzenia współczesnego oprogramowania przemysłowego, w celu osiągnięcia bardziej czytelnych wzorów, które następnie mogą być prosto modyfikowane, co przyczynia się do skrócenia czasu opracowania systemu, uzyskania tak zwanego proof-of-concept w krótkim czasie tak, aby zminimalizować występowanie błędów w projekcie. Ponadto, niektóre kierunki są wskazane w odniesieniu do ewentualnego przyszłego rozszerzenia zakresu stosowanych technik do celów automatycznego testowania i walidacji opracowanych systemów sterowania logicznego.

Słowa kluczowe: automatyczne generowanie kodu, modelowanie specyficzne dla dziedziny zastosowań, systemy sterowania logicznego, meta-modelowanie

Michael Scopchanov, PhD
Michael.Scopchanov@zut.edu.pl

He is born on October the 11th 1977 in Varna, Bulgaria. In 2009 he graduated the Technical University of Varna, Bulgaria, and acquired a PhD Degree in automatics. His research interests cover topics from the broad engineering field of the digital control systems, namely programmable logic controllers, digital signal processing, machine vision, mobile robotics, embedded control, meta-modeling and fuzzy logic. He is a former associate professor at the Faculty of Computing and Automation of the Technical University of Varna, Bulgaria. Currently he is a member of the iLoad project's research team.

Krzysztof Pietrusewicz, PhD, DSc
krzysztof.pietrusewicz@zut.edu.pl

He graduated (MSc Eng.) from Faculty of Electrical Engineering of West Pomeranian University of Technology in Szczecin, Poland in 2002, where he has been working as an assistant professor since 2006. At the same university he gained PhD (2005) and DSc (2012).He also cooperates with Cargotec Sweden AB (Applied Research Department, HIAB) as a researcher.

Hristo Hristoskov, PhD
hristo.hristoskov@zut.edu.pl

He is born on April the 24th 1983 in Varna, Bulgaria. In 2009 he graduated the Technical University of Varna and acquired a Master degree in automation. His research interests are in the areas of digital control systems, industrial communications, programmable logic controllers and control of robots. Develops projects for repair and modernization of machine tools and metalworking machinery CNC. He is a former assistant professor at the Faculty of Computing and Automation of the Technical University of Varna, Bulgaria. Currently he is a member of the iLoad project's research team.