Model-Driven Engineering of Industrial Process Control Applications

Tomaž Lukman[1], Giovanni Godena[1], Jeff Gray[2], Stanko Strmčnik[1]

[1] Jožef Stefan Institute
Department of Systems and Control
Jamova 39, 1000 Ljubljana, Slovenia
tomaz.lukman, giovanni.godena,
stanko.strmcnik@ijs.si

[2] University of Alabama
Department of Computer Science
Tuscaloosa, AL 35487-0290, USA
gray@cs.ua.edu

Abstract

Software is an important part of industrial process control systems. However, the state-of-the-practice for developing industrial process control software still has several key challenges that need to be addressed (e.g., migration to platforms of different vendors, lack of automation). This paper introduces a model-driven engineering approach to the development of industrial process control software, which is based on the ProcGraph domain-specific modeling language. The paper discusses and offers solutions to several of the development challenges that have not been addressed by existing techniques in the process controls domain. The contributions of the paper are a model-driven engineering approach for the industrial process control domain and a supporting tool infrastructure. The approach is demonstrated by a case study focused on the development of a control system for a TiO_2 (titanium dioxide) pigment production subprocess.

1. Introduction

Industrial process control systems are used in many industrial sectors to achieve production improvement, process optimization and time and cost reduction [1]. These systems are responsible for controlling a particular industrial process (e.g., a chemical process), which transforms the process inputs into the desired outputs. The current state-of-the-practice for developing industrial process control systems faces several challenges (e.g., migration problems and the lack of automation) [2] which are outlined in Section 2.

Some of the approaches and concepts that are used to develop software in other domains could also be used to address the challenges of process control systems development. This paper discusses two such approaches. The first concept are Domain-Specific Modeling Languages (DSMLs) [3], which enable the modeling of software and systems using concepts (e.g., terms, symbols and pictures) and abstractions that are common to a specific domain. Practitioners and researchers have reported several advantages [4, 5] (e.g., expressing a design intent in familiar terms) of using DSMLs instead of General-Purpose Modeling Languages (GPMLs), such as the Unified Modeling Language (UML) [6]. A second promising approach that has been successfully introduced into several organizations and domains is Model-Driven Engineering (MDE) [7], which promotes a systematic development process based on precise models at different levels of abstraction from which automated analysis and synthesis is possible. Together these two approaches foster considerable gains in productivity and software quality [8].

The few existing proposals (e.g., [1, 9]) for the application of MDE to the engineering of industrial process control systems have not received widespread adoption by the industry [10]. The main contributions of this paper are: an MDE approach to the engineering of industrial process control applications (based on the ProcGraph DSML [11]) and the tool infrastructure for supporting such an approach. Both were developed in consideration of the challenges that have hindered previous adoption of MDE in the discussed domain.

In Section 2, we identify several challenges that are not being addressed by the state-of-the-practice of industrial process control systems development. Section 3 presents promising technologies that could be used to overcome these challenges. Section 4 shows how our MDE process has been made possible through the development of the necessary infrastructure. Section 5 introduces our MDE process, which is based on the ProcGraph DSML. This process is demonstrated through a case study in Section 6. A discussion of our research and its benefits is contained in Section 7. In Section 8, an overview of the related work is provided. The paper ends with a conclusion and consideration of future work.

2. Challenges of engineering industrial process control applications

The current state-of-the-practice for developing industrial process control applications is not meeting the

demands of the market (e.g., short time-to-market periods and high reliability). It has faces several challenges [2], which are the obstacles on the way to meet these demands. Based on Streitferdt et al. [2] and our experiences gained from several industrial projects in the domain of process control, we believe that the most important of these challenges are:

i. *Lack of automation in the development process* – it is hard to transform the deliverables of the design phase into an implementation of a process control system [10]. Therefore, the translation between the design and implementation is commonly performed manually. Colla et al. [10] supported this claim with a survey, which showed that none of the surveyed companies in the process control industry is employing automatic code generation. The reason for this is either the use of too informal approaches or the use of general-purpose approaches like UML without profiles, where the gap between the procedural-imperative IEC 6113-3 programming languages and object-oriented concepts is too wide.
ii. *The migration challenge* – the majority of industrial process measurement and control systems are implemented on Programmable Logical Controllers (PLCs) [12], which usually can only be programmed in the development environment of their vendors. This can be problematic because such environments are intentionally closed and often have only a proprietary import and export facility. Because of that it is very challenging to migrate a system (i.e., its implementation) from the platform of one vendor to the platform of another vendor. Efforts such as the XML interchange schema proposed by the PLCOpen worldwide association [13] have not been able to solve this challenge.
iii. *The use of improper abstractions* – the developers often use the means (e.g., valves and pumps) instead of the goals of the system (e.g., operations and activities) as the central abstractions in their system analysis and design models. Such a function-centric and low-level view, where physical devices are used instead of high-level domain-specific abstractions, can lead to a suboptimal decomposition of the system, which can lead to a suboptimal system [11]. In other words, unnecessary complex models that are too focused on implementation concerns can arise in the early phases of the development, which makes it hard to develop an optimal system. This problem can quickly occur with UML and other object-oriented approaches, because these approaches are too general and do not constrain the developers to the use of the right abstractions. Failure to determine the right abstraction entities is likely the main source of problems in the development of process-control software [11].
iv. *The lack of verification and validation* – the state-of-the-practice relies on testing instead of verification and validation to address the need for high quality (e.g., reliability) of process control systems. These systems are often tested and approved on sight (e.g., in the factory) [14]. In this setting, errors that may cause accidents on the plant floor and/or the discontinuance of production can be quickly overlooked or even introduced with the ad-hoc corrections that are made. Downtime costs of large industrial factories are very high and can go up to several million dollars per day [2].
v. *Specifics of the developers* – the majority of the software developers who work on process control systems have an electrical engineering background. They are not familiar with object-oriented concepts and with extensive modeling languages, but are accustomed to IEC (International Electrotechnical Commission) languages and concepts. Therefore, they have difficulties to learn and to use UML [15] and UML profiles. Another challenge for these developers with most of their experience in programming [2] is the shift from a code-centric way of development to a model-centric way (where models are altered instead of the code), because the mindset of the developers has to change.

These challenges corroborate the need for improvements in the development process of industrial control systems. The next section will briefly introduce a concept that has the potential to address these challenges if it is used and applied wisely.

3. Overview of model-driven engineering

MDE is a software development paradigm that has the potential to sustainably raise development productivity [16] and to reduce the complexity of software development [7]. The adoption of MDE offers a development environment that promotes the systematic and disciplined use of models throughout the lifecycle of a software system. The essential idea of MDE is to shift the attention from program code to models, such that models become the primary development artifacts [17]. MDE relies on DSMLs and model transformations, which can be realized through appropriate tool support.

3.1. Domain-specific modeling languages

Some software/systems development organizations specialize within one or only a few application domains. With each new project, such an organization can develop new expert knowledge about the domain of their core competency. Such domain-specific knowledge is often the prime intellectual property of an organization [16]. To shield the organization against losing its competitive advantage due to personnel fluctuation, this knowledge

should be made explicit. An excellent way to preserve the institutional knowledge of an organization is to develop a DSML, which can formalize the application structure, behavior, and requirements within a particular domain [7]. This formalized knowledge is reused every time when a model instance is created in a particular DSML. The two common ways to define a DSML are:

- *UML-based definition:* This approach specializes UML for a specific domain through the definition of a UML profile that consists of a set of stereotypes, tagged values and constraints.
- *Metamodel definition:* This approach defines the DSML "from scratch" with the use of a metamodeling language (e.g., Meta Object Facility [18]).

3.2. Model transformations

A model transformation generates one or more target models from a source model, based on the defined transformation rules. With model transformations, MDE can automate many of the complex but routine development tasks that are often performed manually [21]. This is possible only when formally defined and precise modeling languages are used. Examples of tasks that can be automated with model transformations are the generation of lower level models and eventually code from higher level models [22] and model refactoring [23]. A strong benefit of using model transformations together with DSMLs is that a large portion of the code (up to 100%) can be automatically generated, which is not possible with GPMLs like UML.

3.3. Tool support

To enable MDE in practice, sophisticated tools must be developed that support the used DSML(s) and the defined model transformations. If the DSML is realized through a UML profile, then tool support may be provided with the use of existing modeling tools (e.g., AndroMDA [19]). If a DSML is realized through metamodeling, then several tools that enable different development tasks (e.g., modeling and code generation,) must be developed. Based on the literature [20] and our own experience, the effort needed to develop a basic set of these tools (e.g., graphical model editor and code generator) without the use of a specialized toolset is measured in several person-years. Fortunately, metamodeling environments (e.g., GME - Generic Modeling Environment [21] and GMF - Graphical Modeling Framework [22]) facilitate the development of the tools needed to use the selected DSML(s). These metamodeling environments can automatically generate (portions of) the needed tools based on a formal definition of the DSML and the model transformations.

3.4. Using model-driven engineering in the industrial process control domain

Several attempts to convey the concepts of MDE and/or DSMLs to the engineering of industrial process control systems have already been made. However, these have not been widely adopted into the industry, mainly because of their immaturity [10] and inability to properly address the challenges that were presented in Section 2. An overview of the existing approaches can be found in the related work section (Section 7). The following section briefly describes the development and introduction of our MDE approach, which is based on the ProcGraph DSML.

4. Introducing model-driven engineering into the process control domain

In order to realize our MDE approach we had to develop several artifacts, which are the basis of our tool infrastructure. The necessary parts of this infrastructure were: a formalized version of the ProcGraph language, a model repository, a graphical model editor and a code generator (currently only for Mitsubishi PLCs). These tools are realized in the form of an integrated development environment (IDE). The components of the IDE are shown in Figure 1. A screenshot of our IDE can be seen in Figure 4.

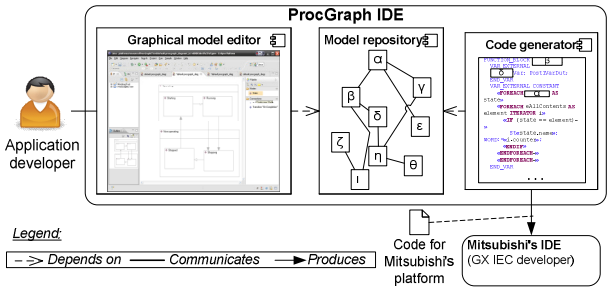

Figure 1. The structure of the tool support that enables MDE with the ProcGraph DSML.

Before we started to formally define the ProcGraph language we had to decide which of the two approaches mentioned in Section 3.3 (i.e., UML extension or metamodeling) to use. We decided to adopt the metamodeling approach, where we defined the metamodel (i.e., a model of the modeling language) and some additional models (e.g., the model of the notation representing the concrete syntax), from which a partially automatic generation of the infrastructure tools was achieved. It is important to point out that the ProcGraph language emerged in the 1990s, before the UML was officially standardized; therefore, ProcGraph was initially developed independently from any knowledge of the UML. Even before starting the current effort, we deliberated on whether to use UML profiles as our implementation approach. We decided against the UML for the following reasons: the UML is unsuitable for the developers in this domain; the UML is complex, vast and non-orthogonal; the UML extension mechanisms

complicate UML even more; the extension mechanisms do not (easily) allow to delete or omit parts of the UML metamodel; the inability to fully customize the concrete syntax (ProcGraph's concrete syntax should not change, because existing developers have grown accustomed to it); the inability to remove the UML semantics in place of the domain semantics implied by ProcGraph.

The model repository was developed with the Eclipse Modeling Framework (EMF) [23], which generates the model repository based on the formal definition of the language metamodel (i.e., its abstract syntax).

For the graphical model editor, we used the GMF (Graphical Modeling Framework) [22] metamodeling tool. GMF can automatically generate a generic visual modeling editor for an arbitrary DSML, as defined by a collection of models. These models are a domain model (i.e., the language metamodel), a notation model (that defines the visual part of the language), a tooling model (that defines what can be added to a diagram), a mapping model (that relates elements of the domain, notation and models to each other) and a generator model (that allows the fine tuning of the generated editor). Often the basic generated model editor has to be extended through a considerable amount of additional code to become suitable for the use in industrial projects. Therefore, a lot of custom code had to be written to implement some of the features of the ProcGraph language.

For the code generator, we first had to select a target programming language. PLCs can be programmed through one of the following programming languages, which are defined by the IEC 61131-3 standard:

- Instruction list (IL), a low-level assembler-like textual language.
- Structured text (ST), which is a high-level textual language similar to Pascal.
- Ladder diagram (LD), which is a graphical language that has the same elements as electromechanical relay systems (e.g., contacts and coils of relay).
- Function block diagram (FBD), which is a graphical language consisting of function blocks (they can be hierarchically arranged via decomposition) with wire-like connections between them.
- Sequential function chart (SFC), which is a graphical language that focuses on structuring sequential tasks of an application through programs and function blocks.

The most suitable target language is the one that has the closest semantics to the ProcGraph DSML (introduced in Section 5). In the beginning of our project, we eliminated the most primitive languages (i.e., the IL and LD). Because the ProcGraph language is based on hierarchical decomposition into more diagrams, the target language also has to have some kind of decomposition mechanism. Based on this requirement we decided against SFCs, which in some development environments cannot be structured into a hierarchy. In the end we chose the FBD language, which has an appropriate decomposition mechanism, with the combination of ST, which is used to specify the detailed processing actions. For the implementation of the code generator, we used the openArchitectureWare tool [24], which enables the specification of model-to-text transformations that generate code through templates. We decided to initially develop a generator for PLCs from Mitsubishi, because these are the most used PLCs of our industrial partners. Most of the development environments have import/export capabilities, which enable the definition of IEC 61131-3 compatible programs with a proprietary textual format or an XML format. With this capability it is possible to import the generated code into a development environment and execute it on proprietary PLC platforms. However, achieving this goal is not easy, because the format of these interfaces usually is not documented. This is also the case for the Mitsubishi GX IEC Developer, which has an undocumented and proprietary format of textual files that can be parsed to construct FBDs.

In addition to the components, Figure 1 also shows two interfaces. The user interface presents information that is understandable for domain experts and application developers. The other interface is used to output the generated code that is imported into the development environments of arbitrary PLC vendors, where it can be compiled into code, uploaded onto a PLC and executed.

The next section presents our MDE process for the development of process control software and the ProcGraph DSML.

5. Model-driven engineering of industrial process control applications

This section presents the activities of our MDE process, which is based on the ProcGraph DSML and the developed tool infrastructure. A detailed presentation of these activities is given through a case study, which is described in Section 6.

5.1. The ProcGraph Language

The ProcGraph language [11] is a DSML that was developed for modeling process control systems. It enables the conceptual modeling of such systems and can be used to construct specifications (i.e., models) in the early stages of the software lifecycle (i.e., in the analysis and design phases). The language gradually evolved from information gained in applying it to over 20 industrial projects.

The ProcGraph DSML addresses two of the key challenges that emerge in this domain: the challenge of using improper abstractions (iii.) and the challenge of the specific developer profile (v.), because it has only a

small number of domain-specific and goal-oriented abstractions, which are well-known by the developers.

The ProcGraph language was initially developed as a notation that was defined in a semi-formal way. It was used for the specification of control systems "on paper" (i.e., specifications were drawn in a general-purpose graphical tool, such as Visio). The idea arose to build a tool infrastructure that would enable modeling with this language and eventually computer-aided engineering of process control systems. Before the existence of the ProcGraph tool infrastructure that is described in this paper, ProcGraph models were converted into code manually by developers.

5.2. The Model-Driven Engineering Process

The development process and its activities are strongly influenced by ProcGraph and the developed tool infrastructure. This process consists of the following activities: Requirements definition, Structural modeling, Modeling of behavior, Modeling of interdependent behavior, Platform selection, Software to hardware mapping, and Testing. To demonstrate how industrial process applications are developed with ProcGraph and the associated tool support, a case study is presented in the next section.

6. Case study: the control system for drying of a TiO_2 suspension

In this case study, we present a control system for an industrial process, which is depicted through a Piping and Instrumentation Diagram (P&ID), as shown in Figure 2. The studied process (taken from [11]) is the continuous process of drying the titanium dioxide (TiO_2) suspension, which is one of the last subprocesses in TiO_2 pigment production. The TiO_2 suspension enters the storage vessel (A), from where it is pumped into the rotating vacuum-dryer (B) that dries it. The dried suspension is transported into the dispergator vessel (C) from where it is pumped to the drying chamber (D). In this drying chamber, the water instantly evaporates because of the hot gasses from the thermo-aggregate (E). The gained pigment is transported into silos H and I. This pneumatic transport is powered by fans F and G. Inside the silos, most of the pigment falls to the bottom from where it is transported by the screw-conveyors transport system (M) to the next subprocess of the whole production process. However, some pigment may remain on the bag filter on top of the silos, where it is separated from the wet flue gasses. The gasses proceed through the scrubbing system, as indicated by the Venturi scrubber (J) and water-gas separator (K) to the chimney (L).

The *requirements* for control applications are often defined through a P&ID and supporting documents (e.g., informal operational and safety related requirements).

Structural modeling can begin after the requirements are defined. The first step of these activities is to identify Procedural Control Entities (PCEs), which are the main abstractions used in ProcGraph. A PCE can abstract a process, an operation or an activity, which are all central parts of the process domain. PCEs can be identified by the system analyst communicating with domain-experts (process engineers) in a relatively easy way [11], based on high coherence and low coupling between potential PCEs. After all the relevant PCEs of an industrial process are found, the next step is to model a PCE dependency diagram. This is the main domain-specific diagram type of ProcGraph, which shows all the PCEs of the control system under design and a high-level view of the dependencies among them (these are defined on a different diagram). Figure 3 shows all the PCEs that were identified in the case study. For instance, the "Rotating Vacuum Drying" PCE abstracts the subprocess, which is carried out on the storage vessel (A) and the rotating vacuum-dryer (B) parts of the presented P&ID.

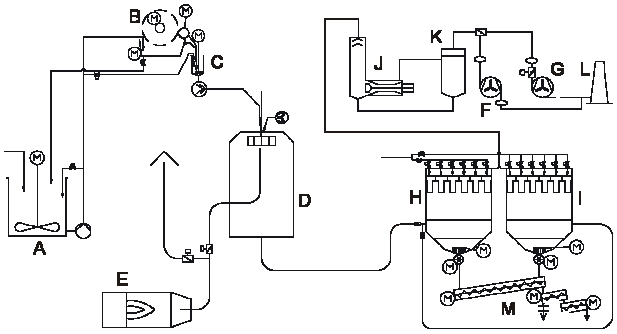

Figure 2. Process of TiO_2 suspension drying.

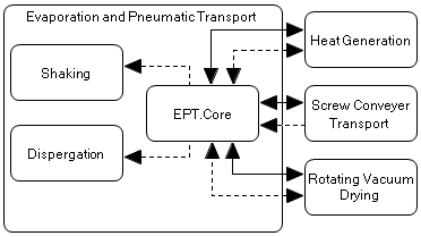

Figure 3. The PCE dependency diagram for the case study shown in Figure 2.

The next activity is the *modeling of behavior*, which is done with PCE state transition diagrams. This diagram type defines the behavior of a specific PCE. Each elementary PCE (i.e., a PCE that has no sub-PCEs) is decomposed into a state transition diagram. This diagram type is an extended Finite State Machine (FSM), which differs from other extended FSM variants (e.g., Statecharts and UML state diagrams) in the following details: a) the functionality and the behavior are not separated – there is a single unified model with all the processing being assigned to distinct parts of the behavior model as its action sequences, b) the processing

is organized into a richer set of action sequence types (entry, loop, exit, always and transient action sequences of states and one action sequence of transitions), c) all action sequences have a duration, d) overlapping superstates, and e) two transition types. Such a diagram consists of states and transitions, and also allows state hierarchies. A state transition diagram for a part of the case study is depicted in Figure 4. The two available types of transitions are: 1) the transition on event, which is denoted by a filled arrowhead, is triggered when the PCE is in the source state of the transition and the event (either an operator command or a process equipment signal) specified as the guard of the transition occurs; 2) the transition on completion, which is denoted by an empty arrowhead, is triggered when the source state of the transition is finished with its loop processing. The actions, which are the basis of the processing performed in states and transitions, are defined with the structural text language (defined in IEC 61131-3 standard).

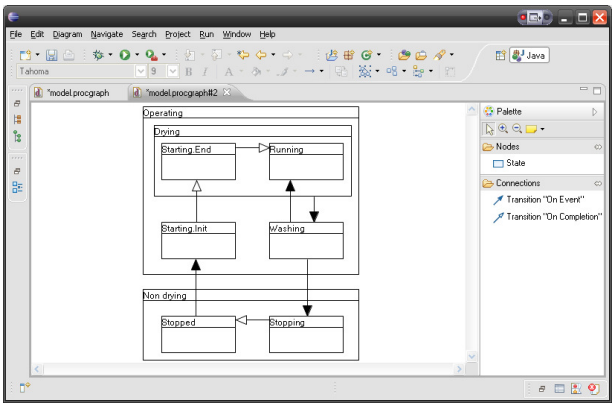

Figure 4. The PCE state transition diagram of the "EPT.Core" from Figure 3.

The *modeling of interdependent behavior* is done with the PCE dependencies state transition diagram. Such a diagram exactly defines the mutual behavior dependencies between two PCEs. A dependencies state transition diagram for a part of the case study is given in Figure 5. This diagram type consists of the state transition diagrams of two interdependent PCEs and dependency relationships, which define the behavior dependencies among them. A dependency can be either a conditional dependency, which is denoted by a normal line with a filled arrowhead, or a propagation dependency, which is denoted by a dashed line with filled arrowhead. The source state of a conditional dependency has to be the active state of its parent PCE so that a transition, which is at the end of this dependency, can be fired. The transition into which a propagation dependency sinks is fired, when the source state of this dependency is the active state of its parent PCE and if the other PCE is in the source state of the target transition of this dependency. All of the behavior interdependencies between two PCEs that are defined on a dependency state transition diagram are summarized on the dependency diagram as a union of the defined dependencies. The high-level view dependency between two PCEs on a dependency diagram is decomposed into a dependency state transition diagram.

The *platform selection* follows after a complete ProcGraph model has been constructed. A platform is determined through the selection of a code generator. A code generator for a specific platform should be developed by tool developers. In our case study, we used the code generator for Mitsubishi PLCs (i.e., for the Mitsubishi GX IEC developer). Figure 6 shows a part of the generated code for our case study.

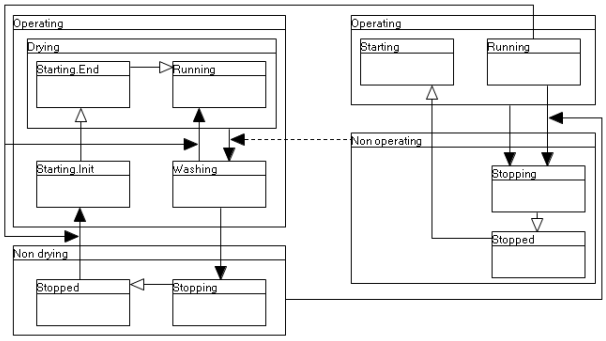

Figure 5. The PCE dependencies state transition diagram of "EPT.Core" and "Screw Conveyer Transport".

The next activity is the *mapping of software onto the hardware*, which consists of mapping software onto computation nodes, processors and tasks. Currently, our MDE process based on ProcGraph does not allow the visual modeling of this mapping. The mapping is described in either the generator or in the development environment of the PCE vendor.

The *testing* of the application can begin after the generated code is imported into the IDE of the selected platform vendor, compiled and uploaded on the PLC.

7. Discussion

An earlier attempt to implement a modeling environment for ProcGraph was made (reported in [25]), but was only partly successful and was eventually discontinued. In that attempt the tool was built from scratch, without the use of any metamodeling tools. That attempt was problematic because the result was hard-coded and not flexible enough to evolve based on requests for new features. Also, the first attempt at ProcGraph tool support did not provide many of the needed features and was developed inefficiently (i.e., it was labor intensive and challenging to develop). This motivated us to start another attempt to provide tool support that was based on a formal definition of ProcGraph and the use of existing metamodeling tools.

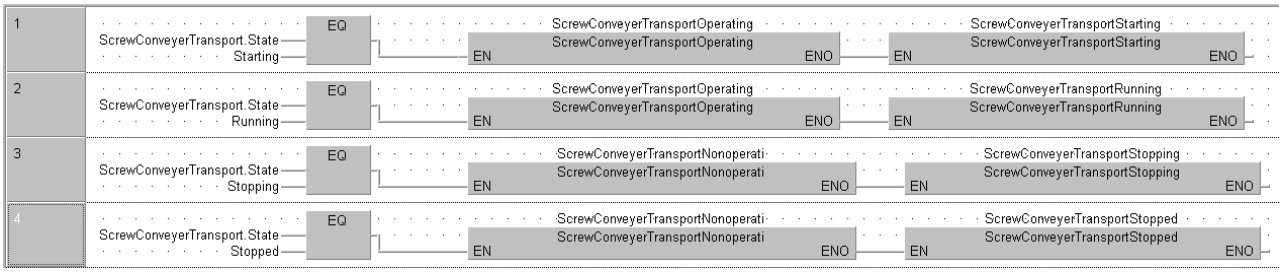

Figure 6. The generated FBD for the state transition diagram of "Screw Conveyer Transport", which is a part of the whole generated code.

The adoption of MDE, which is based on the ProcGraph DSML, has brought the following benefits:

- *Increased software quality*: Because the transformations are repeated each time the code is generated, the code does not contain human coding errors. Because ProcGraph is used, the knowledge about how to optimally decompose the system is reused every time a model instance is created. Both of these factors increase the software quality.
- *Increased productivity*: Because the model-to-code transformations are formally defined the translation into the implementation could be automated with the code generator for Mitsubishi PLCs. This addresses one of the identified challenges (i.) and saves development time. Another factor that improves productivity is the inherent reuse of knowledge that occurs when the ProcGraph language is used.
- *Platform independence and platform migration*: The developed tool infrastructure allows the migration to a new PLC platform (it addresses challenge ii.), which can be achieved by developing a new code generator. This enables the platform-independent ProcGraph models to be reused across different PLC environments and thus offers consistent and safe platform migration, without losing functionality.
- *Improved communication and interaction between development participants*: Due to the improved formalization of the ProcGraph language, the communication between development participants is improved because common misunderstandings are reduced due to the used domain abstractions.

8. Related work

In the domain of industrial process control, MDE is still an upcoming technology [26], although a few approaches were already proposed. The one proposed by Estevez et al. [27] is based on two different UML profiles that are used to model three different views on the system (the functional, hardware architecture and the software-to-hardware mapping view) and to achieve code generation. Compared to our approach, it also allows the modeling of the software to hardware mapping. However, our approach is on a higher level of abstraction and is more suitable for the developers in this domain. Another approach [28] from the same research group, where XML is used as the modeling language. However XML is not easily comprehensible by humans.

Thramboulidis [15] relies on the transformation of UML models to function block models (defined with a UML profile) and eventually to an implementation in the C programming language. This approach does not enable the generation of code, which is executable on PLCs. In [9], the authors present a function block based DSML with tool support (developed with the GME metamodeling tool), which enables the model-to-model transformation into CORBA (Common Object Requesting Broker Architecture) component models. However, they do not mention how code for PLCs can be generated from CORBA component models.

The usage of the UML-PA profile, which is promoted in [12], enables automatic code generation into a combination of structured text and sequential function charts. This approach has the same drawback as other UML based approaches and is therefore relatively unnatural for control system developers.

Maurmaier [29] proposes a MDE framework, which is extended trough specific model transformations and a project specific platform to enable the modeling of the software to hardware mapping. However this proposal does not mentioned, which modeling language it uses and if automatic code generation is already achieved.

9. Conclusion and future work

This paper presents an approach to the engineering of industrial process control systems, which is based on the MDE paradigm and uses the ProcGraph DSML for modeling. We identified the current challenges that negatively influence the engineering of process control systems, which are not covered by the state-of-the-practice. We described the development of our MDE approach through the important design decisions that were made during the development of the software tool infrastructure, which enables this approach. We presented our MDE process through the activities of which it consists. To demonstrate this process, we introduced a case study where a control system from a real industrial project was engineered. An observation and discussion of our approach revealed several benefits, which were experienced through its usage, and showed

that our approach addresses the identified challenges. It is our belief from using ProcGraph and the associated tool infrastructure that we realized improved software quality, increased productivity, platform independence and easier platform migration, and improved communication between development participants.

Our plans for the future are to develop additional tools that support new engineering tasks. More concretely, we want to enable the verification of these systems, which is currently not carried out in the state-of-the practice. This will allow us to shift from a construct-by-correction way of development, which relies on testing, to a correct-by-construction process, which minimizes the need for testing, since verification is employed (and it addresses the challenge iv. that was identified in Section 2).

References

[1] M. Marcos and E. Estevez, "Model-driven design of Industrial Control Systems," in International Conference on Computer-Aided Control Systems (CACSD 2008), 2008, pp. 1253-1258.

[2] D. Streitferdt, G. Wendt, P. Nenninger, A. Nyβen, and L. Horst, "Model Driven Development Challenges in the Automation Domain," in 32nd IEEE International Computer Software and Applications Conference, 2008, pp. 1372-1375.

[3] J. Sprinkle, M. Mernik, J.-P. Tolvanen, and D. Spinellis, "What Kinds of Nails Need a Domain-Specific Hammer?," IEEE Software, vol. 26, pp. 15-18, 2009.

[4] R. Acerbis, A. Bongio, M. Brambilla, S. Butti, S. Ceri, and P. Fraternali, "Web applications design and development with WebML and WebRatio 5.0," in TOOLS Europe 2008, 2008.

[5] A. Evans, M. A. Fernández, and P. Mohagheghi, "Experiences of Developing a Network Modeling Tool Using the Eclipse Environment," in 5th European Conference on Model Driven Architecture - Foundations and Applications, 2009, pp. 301-312.

[6] The Object Management Group, Unified Modeling Language: Superstructure, Version 2.0, 2004.

[7] D. C. Schmidt, "Model-Driven Engineering," IEEE Computer, vol. 39, pp. 25-31, 2006.

[8] M. Volter and T. Stahl, Model-Driven Software Development. John Wiley & Sons, 2006.

[9] K. Thramboulidis, D. Perdikis, and S. Kantas, "Model driven development of distributed control applications," The International Journal of Advanced Manufacturing Technology, vol. 33, pp. 233-242, 2007.

[10] M. Colla, T. Leidi, and M. Semo, "Design and implementation of industrial automation control systems: A survey," in 7th IEEE International Conference on International Conference on Industrial Informatics, 2009, pp. 570-575.

[11] G. Godena, "ProcGraph: a procedure-oriented graphical notation for process-control software specification," Control Engineering Practice, vol. 12, pp. 99-111, 2004.

[12] B. Vogel-Heuser, D. Witsch, and U. Katzke, "Automatic code generation from a UML model to IEC 61131-3 and system configuration tools," in International Conference on Control and Automation, 2005, pp. 1034-1039.

[13] PLCOpen international organization, "PLCOpen," http://www.plcopen.org/, accessed on 15.01.2010.

[14] K. Fischer, G. Hordys, and B. Vogel-Heuser, "Evaluation of an UML Software Engineering Tool by Means of a Distributed Real Time Application in Process Automation," in Modellierung 2004, 2004.

[15] K. C. Thramboulidis, "Using UML in control and automation: a model driven approach," in International Conference on Industrial Informatics, 2004, pp. 587-593.

[16] B. Selic, "The pragmatics of model-driven development," IEEE Software, vol. 20, pp. 19-25, 2003.

[17] E. Seidewitz, "What models mean," IEEE Software, vol. 20, pp. 26-32, 2003.

[18] The Object Management Group, Meta Object Facility: MOF Core specification, Version 2.0, 2006.

[19] J. Kozikowski, "A Bird's Eye View of AndroMDA," http://docs.andromda.org/contrib/birds-eye-view.html, accessed on 15.01.2010.

[20] S. Kelly and J.-P. Tolvanen, Domain-Specific Modeling. John Wiley & Sons, 2007.

[21] Á. Lédeczi, Á. Bakay, M. Maróti, P. Völgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing Domain-Specific Design Environments," IEEE Computer, vol. 34, pp. 44-51, 2001.

[22] R. C. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional, 2009.

[23] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, Eclipse Modeling Framework. Addison-Wesley Professional, 2003.

[24] A. Haase, M. Volter, S. Efftinge, and B. Kolb, "Introduction to openArchitectureWare 4.1. 2," in MDD Tool Implementers Forum, 2007.

[25] G. Kandare, G. Godena, and S. Strmčnik, "A new approach to PLC software design," ISA Transactions, vol. 42, pp. 279-288, 2003.

[26] D. Streitferdt, G. Wendt, P. Nenninger, A. Nyβen, and L. Horst, "Model Driven Development Challenges in the Automation Domain," in 32nd Annual IEEE International Computer Software and Applications Conference, 2008, pp. 1372-1375.

[27] E. Estevez, M. Marcos, I. Sarachaga, and D. Orive, "A Methodology for Multidisciplinary Modeling of Industrial Control Systems using UML," in 5th IEEE International Conference on Industrial Informatics, 2007, pp. 171-176.

[28] E. Estévez, M. Marcos, and D. Orive, "Automatic generation of PLC automation projects from component-based models," The International Journal of Advanced Manufacturing Technology, vol. 35, pp. 527-540, 2007.

[29] M. Maurmaier, "Leveraging model-driven development for automation systems development," in International Conference on Emerging Technologies and Factory Automation (ETFA 2008), 2008, pp. 733-737.