SysML-based Optimisation of Global Variables Arrangement for Visualisation in Distributed Control Systems Oriented Towards Communication Performance

Marcin Jamro[1], *Dariusz* Rzonca[2,*]

[1]TITUTO Sp. z o.o. [Ltd.], Zelwerowicza 52G, 35-601 Rzeszow, Poland
[2] Rzeszow University of Technology, Department of Computer and Control Engineering, Al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland

Abstract. Distributed Control Systems (DCSs) often perform complex and critical operations in the industry. To make work of operators easier, they are often equipped with Human-Machine Interface (HMI) panels that allow observing the current state of a process, as well as adjusting the configuration. However, when visualisation consists of several displays with many controls and a high number of calculations, it is possible to encounter some performance-related problems due to communication between a controller and the HMI panel. This problem can be limited by arranging global variables in the way that decreases the number of requests or transmitted data. The paper shows an approach of solving this problem, based on a set of SysML models that specify visualisation displays and auxiliary programs, together with involved global variables. The proposed approach can be expanded to operate as a part of a comprehensive methodology of modelling, implementation, visualisation, and testing of a DCS.

1 Introduction

Nowadays, Human-Machine Interface (HMI) panels are commonly used in the industry. They visualise a control process state and facilitate parametrisation adjustment. A screen layout is based on graphical primitives, either simple (e.g., circle, line) or complex ones (e.g., chart, dial graph). Their current appearance (e.g., colour, size, displayed text) depends on actual values of connected variables. Such values are read from a remote controller. Thus, efficient communication is crucial for proper operation of the system.

Typically, an industrial communication network [1] is based on a fieldbus [2] and one of the field protocols defined in the IEC 61158 standard [3]. Non-standard approaches, such as using Hypertext Transfer Protocol (HTTP) instead of specialised field protocol, have been also described [4], but they are rarely used in practice. Configuration of communication involves defining multiple communication tasks to exchange all visualised variables. In some cases, depending on the communication protocol and variables, data can be exchanged in different ways, which influence communication performance. Such an impact is discussed in the paper. To facilitate optimal configuration of the communication channel, an approach based on the Model-Driven Development (MDD) paradigm and SysML diagrams is introduced in the following sections.

The proposed approach could be also integrated with the CPDev engineering environment [5] to apply it for real industrial systems. CPDev is suitable for PLC programming in languages defined in the IEC 61131-3 [6] standard. One of its functionalities covers designing of graphical HMI interfaces in a similar way to developing control programs, using the common environment [7]. Therefore, mutual connections between control program and visualisation are closer as both parts are developed together, rather than separately.

The paper is organised as follows. The next section presents information about related work, especially application of models during control software development process. Then, an overall concept of the introduced approach is discussed. Such a topic is detailed in two following sections. First, proposed SysML models are described, then possible optimisations of variable arrangement are briefly discussed. The paper is concluded in Sec. 6.

2 Related Work

The Model-Driven Development (MDD) [8] is a commonly used paradigm to improve software quality. Such an approach increases a level of abstraction while designing system architecture. The models, created in early stages of development process, show various aspects of the system and can be used for automatic or semi-automatic generation of source code or configuration, for testing purposes, as well as for performance analysis. Other paradigms related to

* Corresponding author: drzonca@kia.prz.edu.pl

modelling have been also described in the literature. Among them, Model Driven Architecture (MDA) [9, 10], Model Driven Engineering (MDE) [11, 12], and Model Based Testing (MBT) [13] seem to be common.

Models can be created in different languages, either formal (e.g., various classes of Petri Nets, including Hierarchical Timed Coloured Petri Nets [14], RTCP Nets [15], or Fuzzy Petri Nets [16]), or semi-formal ones. The latter approach typically uses Unified Modelling Language (UML) [17] or Systems Modelling Language (SysML) [18]. SysML is based on UML and provides nine types of diagrams for modelling behaviour (Activity, Sequence, State Machine, and Use Case Diagrams), requirements (Requirement Diagram), and structure of the system (Block Definition, Internal Block, Parametric, and Package Diagrams).

The MDD paradigm, based on SysML or UML diagrams, can be also applied during development of industrial control software. Some of mentioned approaches are directly related to the communication subsystem, whereas others are dedicated to other parts, e.g., visualisation or control. The application of SysML diagrams for modelling of an IEC 61131-3 software has been presented in [19]. Another approach, including the Round-Trip Engineering (RTE) concept and partial generation of the source code based on SysML diagrams has been described in [20]. SysML-AT (SysML for automation), a specialised SysML language profile that allows automated software generation for run-time environments conforming to IEC 61131-3, has been described in [21, 22]. SysML can be also applied for modelling of a distributed manufacturing control system [23], developed according to the IEC 61499 [24] standard. Application of the UML diagrams for industrial system modelling has been presented in [25]. Automatic generation of software adapters source code for communication in digital home environment, based on templates and a SysML model of the system, has been described in [26]. The methodology dedicated for SysML modelling of functional and non-functional requirements of the IEC 61131-3 control software, such as behaviour of Program Organisation Units (POUs), performance requirements for POU execution and communication between devices in DCSs, as well as expected operation of HMI panels has been introduced in [27].

The communication in DCS can utilise different paradigms of access to the communication link, among them master-slave, producer-distributor-consumer, and token passing are the most common [28]. All of them are based on the Time Division Multiplexing (TDM) scheme. Alternative Frequency Division Multiplexing (FDM) method has been also discussed [29], but it was not adopted in practice. A scenario of data exchanges in such paradigms, as well as communication subsystem with an industrial protocol, can be also modelled in SysML. Such approaches have been described for token-passing [30], as well as for master-slave including some tests of communication performance [31]. The methodology for SysML-based tests in DCSs has been further developed and described in [32].

3 Overall Concept

DCSs are often equipped with HMI panels to present the current state of the process, as well as to adjust its configuration according to the current needs or preferences of the operator. Of course, visualisation involves presentation of controls which design depends on the current value of variables. As presented in Fig. 1, the visualisation part (shown on the right) consists of displays, including only one that is currently active (Display #1 in the example). Each display contains controls (e.g., labels or dial graphs), configurable using their parameters, such as a background colour or a current value. Of course, it is possible not only to use fixed values as values of parameters, but also current values of global variables (according to the IEC 61131-3 standard, global variables can be addressed from all POUs to facilitate exchange of data). In such a case, the display is automatically refreshed as soon as the associated value has changed. In the example, shown in Fig. 1, the first variable (Variable #1) is assigned to some parameters of controls in two displays, namely Display #1 and #2, Variable #5 is used by two controls in Display #3, while Variable #3 is used in the control programs, not in the visualisation part.

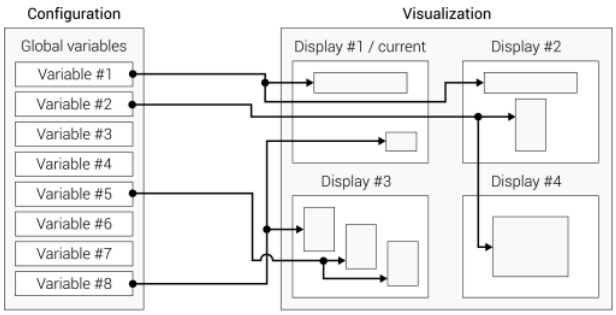

Fig. 1. Overall concept of proposed solution.

One of the important topics in this area is communication between the controller and the HMI part to read data of necessary variables in the way that minimizes the delays. In some common industrial master-slave protocols (e.g., Modbus) it is possible to read in one communication transaction not only a single register, but a block of variables. Therefore, multiple variables for HMI visualisation can be read together. However, such an approach is possible only if addresses of variables are close to each other. If the variables are dispersed over the whole address space, each of them must be read separately. Sometimes several dispersed variables may be grouped in a named list and read together, but many industrial protocols do not implement such a functionality. Thus, even an order of global variables definition could have a significant impact on performance. For this reason, it is the interesting research problem to arrange global variables in the way that improves performance.

To make the proposed approach easy to use in the real-world industrial scenarios, the dedicated process has been proposed, as shown in Fig. 2. At the beginning, the designer prepares a set of SysML diagrams to model the visualisation part, namely display configurations, layouts

of controls, transitions between displays, as well as visualisation programs that make it possible to prepare values of global variables in the form suitable for presentation on the visualisation displays. Then, such models can be used to automatically or semi-automatically generate visualisation displays and programs, which should be adjusted by an engineer. What is more, the SysML models are used to adjust configuration of global variables, defined in the DCS, by correcting their addresses to improve performance of communication-related tasks of the visualisation part.

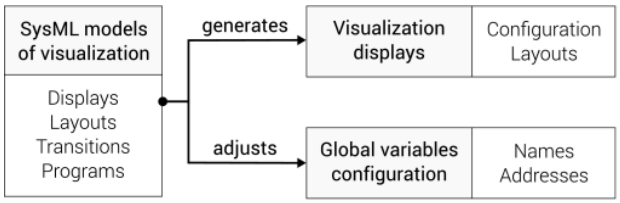

Fig. 2. Process supported by the proposed solution.

By using the proposed approach together with the supporting process, it is possible to eliminate the necessity of adjusting global variables configuration directly by the engineer. Such a task, quite complicated in the case of advanced and comprehensive visualisation, could be performed automatically and adjusted each time when the visualisation part is modified, e.g., by adding new controls to a display and assigning some global variables to its parameters.

What is more, the described process can be further expanded by tests that check whether various parts of the projects, including visualisation, are consistent with the requirements. There are various possibilities of achieving this goal, such as using the dedicated test definition language [33].

4 SysML-based Modelling

The first stage of the proposed process is creation of a few models in SysML. The described approach is a part of the overall modelling methodology, supporting also the visualisation area, which allows using the following diagrams:

- Block Definition Diagrams (BDDs) for displays,
- Block Definition Diagrams (BDDs) for programs,
- Internal Block Diagrams (IBDs) for layouts of displays,
- State Machine Diagrams (STMs) for transitions between displays.

However, this paper focuses only on adjusting global variables configuration and – therefore – only BDDs are used, namely for modelling of visualisation displays and programs. It is worth noting that models of displays are grouped in the `HMI Displays` package, while programs – in `HMI Programs`.

4.1 Displays

Each display, available in the visualisation part of the system, is modelled on a separate BDD, as presented in Fig. 3. Such a diagram has a name (marked as bold `NAME`) the same as the name of the display and is in the `HMI Displays` package. The diagram contains only one block, namely «*hmiDisplay*» with the name equal to `VD_` followed by the display name.

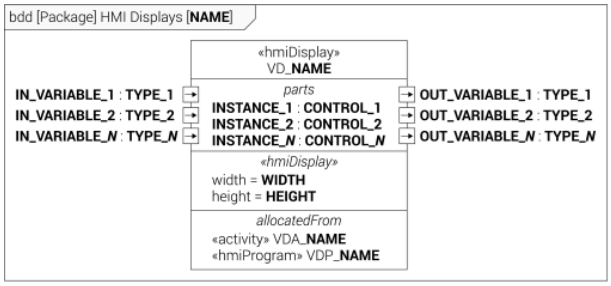

Fig. 3. Model of the visualisation display.

The «*hmiDisplay*» block can contain proxy flow ports that indicate which global variables are used as input parameters for controls on the display or which are used to save values returned as output parameters from controls. This bit of information is crucial for the approach, described in this paper, because is used to check relationships between global variables and displays.

What is more, the «*hmiDisplay*» block contains three additional compartments. The first (i.e., `parts`) specifies controls that are placed on the diagram, in the form `INSTANCE_NAME : CONTROL_NAME`. For example, if the display contains a button and a dial graph, the `parts` compartment contains two entries, namely `BTN_START : BUTTON` and `DIAL : DIAL_GRAPH`.

The other compartment (marked as «*hmiDisplay*») configures basic data of the display, namely its width (`WIDTH`) and height (`HEIGHT`) in pixels. Such piece of information is used while generating the initial visualisation to set proper dimensions.

The last compartment (i.e., `allocatedFrom`) contains references to other modelling elements, such as an activity used on the STM regarding transitions between displays («*activity*» `VDA_NAME`) and additional auxiliary visualisation program («*hmiProgram*» `VDP_NAME`) that prepares data in a form suitable for presentation on the display. Of course, such references are not mandatory.

4.2 Programs

Apart from modelling displays, it is also possible to model auxiliary visualisation programs. Their role is important, because they allow adjusting values of global variables to a form that is suitable for presentation on the display. For example, they can be used to convert a Boolean variable into an index of the colour.

The model of the visualisation program is created on the BDD diagram, as shown in Fig. 4 and is quite similar to the previously described model of the display. In this case, each visualisation program is modelled on a separate diagram, in the `HMI Programs` package, with a name equal to the name of the display.

The diagram contains only one block, namely «*hmiProgram*», which name is the same as the

display, preceded by `VDP_`, which stands for *Visualisation Display Program*. The block contains additional stereotype that specifies the language used for implementation of the program from the languages supported by the IEC 61131-3 standard, either textual, graphical, or mixed. The first group consists of ST (*Structured Text, «st»*) and IL (*Instruction List, «il»*). The graphical languages are FBD (*Function Block Diagram, «fbd»*), LD (*Ladder Diagram, «ld»*), while SFC (*Sequential Function Chart, «sfc»*) is mixed.

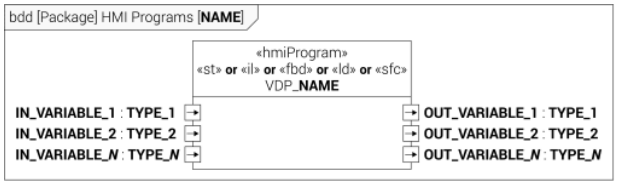

Fig. 4. Model of the visualisation auxiliary program.

The «`hmiProgram`» block can also contain proxy flow ports, shown on its both sides. Flow ports located on the left indicate global variables that are read by the visualisation program, while ones presented on the right-side variables, which are written by the program. Such configuration performs important role for the approach described in the paper, because global variables used by the program associated with the given display, should be read in each cycle when the display is presented to the user.

5 Optimisation of Global Variables Allocation

It has been mentioned that in some cases addresses of global variables, exchanged between the controller and the HMI, may have impact on communication performance. Such an issue is discussed in this section.

Typically, the HMI panel periodically reads the variables that are necessary for visualisation from the controller. Usually, only the variables which are visualised on the current screen are read. However, such variables can be read in several ways. For brevity, the paper focuses on the commonly used industrial master-slave protocol, namely Modbus. However, similar issues can be found in other protocols, as well. Each of data exchange transactions requires a pair of messages (i.e., request and response) to be sent through the network. A master device (HMI) sends a request, asking a controller about values of the variables. The controller replies.

Unfortunately, in many cases such a reply cannot be generated immediately. The controller must decode the request and prepare an answer. Moreover, the controller cyclically executes a control program and, in some cases, reading of the variables must be delayed till the end of the current control loop, to ensure consistency of the values. Thus, it is usually better to read as many variables as possible in one request. Unfortunately, the Modbus protocol does not allow reading of two or more variables which are not close to each other. A single Modbus request includes the address of the first variable and the number of consecutive registers (variables) to read. Thus, reading of all visualised variables at once is possible only if these variables are grouped together, as seen in Fig. 5.

It is worth noting that in some cases, when required variables are separated by a few other variables (which are not necessary for visualisation), it is also better to read them as well, and discard, just to make possible reading of multiple variables in one transaction. The additional time required for transmission of redundant variables will be negligible in comparison to the delay involved in reading of the variables in multiple separate queries. Of course, it depends on the number of such redundant variables, transmission speed, message processing delay in the controller, etc., and can be calculated in a case.

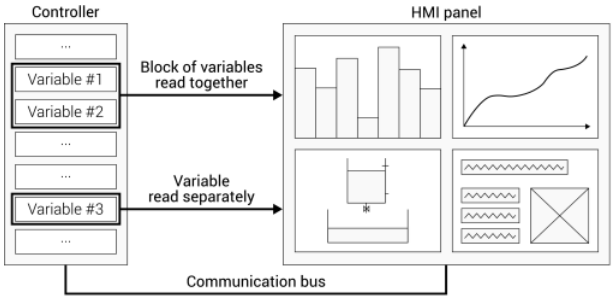

Fig. 5. Influence of variable arrangement on communication tasks.

Another example, where transmission of a single block with some redundant data may be faster than a few separate queries is related to the Modbus data types. In Modbus it is possible to use four different functions for reading a variable, depending on a variable type, namely Read Coils (FC1), Read Discrete Inputs (FC2), Read Holding Registers (FC3), and Read Input Registers (FC4). The first two read binary variables, whereas the other two operate on 16-bit registers. However, in some controllers it is also possible to read binary variable as a 16-bit register, where 15 most significant bits are zeros and only the least significant bit is informative. Such a possibility seems to be slower than using the dedicated binary functions. However, in some cases it can decrease the number of exchanged bytes and accelerate transmission. For example, if such a binary variable, read as a 16-bit long register, adjoins block of the registers which are already read, reading it in the same transaction will require transmitting of two bytes more, whereas reading it in separate transaction, using FC1 function will require eight bytes of the request (station address, function code, variable address – 2B, number of variables – 2B, CRC – 2B) and six of the response (station address, function code, number of bytes, variable value, CRC – 2B).

The model of displays can be used to detect the variables which are necessary for visualisation and should be possibly read together. Thus, the communication delay during different scenarios of data exchange can be calculated. Then, an optimal arrangement of the variables in the controller can be suggested for an engineer or even automatically applied in the control program. Therefore, a negative impact on

performance of communication with HMI can be minimized.

6 Summary

The communication performance has an important role in DCSs. In some cases, the communication overhead can be greatly reduced by slight changes in arrangement of the transmitted variables. To facilitate an early detection of the necessity of such changes, related to the communication between an HMI panel and a controller, the approach based on the SysML diagrams and the MDD paradigm has been proposed.

Such an approach can be even further expanded to operate as a part of the comprehensive methodology of modelling, implementation, visualisation, and testing of a DCS. What is more, it could be integrated with the CPDev engineering environment, as well as its various parts, including the CPModel modelling tool. The planned future work includes incorporation of the described approach into the Round-Trip Engineering (RTE) methodology [20] to automate introducing changes in the source code related to addresses of the variables.

References

1. M. Silva, F. Pereira, F. Soares, C. Leao, J. Machado, V. Carvalho, *New Trends in Mechanism and Machine Science*, ser. Mechanisms and Machine Science, P. Flores and F. Viadero, Eds., Springer International Publishing **24**, 933-940 (2015).
2. J.P. Thomesse, *Proceedings of the IEEE* **93** (6), 1073-1101 (2005).
3. *IEC 61158 Standard: Industrial Communication Networks - Fieldbus Specifications* (2007).
4. A. Jestratjew, A. Kwiecien, *Computer Networks*, ser. *Communications in Computer and Information Science*, A. Kwiecien, P. Gaj, and P. Stera, Eds. Springer International Publishing **160**, 306-313 (2011).
5. M. Jamro, D. Rzonca, J. Sadolewski, A. Stec, Z. Swider, B. Trybus, L. Trybus, *Recent Advances in Automation, Robotics and Measuring Techniques*, ser. Advances in Intelligent Systems and Computing, R. Szewczyk, C. Zielinski, and M. Kaliczynska, Eds. Springer **267**, 81-90 (2014).
6. *IEC 61131-3 - Programmable controllers - Part 3: Programming languages* (2013).
7. M. Jamro, B. Trybus, *The 6th International Conference on Human System Interaction* (HSI), 48-55 (2013).
8. D. Hastbacka, T. Vepsalainen, S. Kuikka, J Syst Softw. **84** (7), 1100-1113 (2011).
9. OMG MDA Specifications (OMG 2014).
10. M. Azmoodeh, N. Georgalas, S. Fisher, BT Technol J. **23**(3), 96-110 (2005).
11. M. Marcos, E. Estevez, N. Iriondo, D. Orive, *IEEE Conference on Emerging Technologies and Factory Automation* (ETFA) 1-8 (2010).
12. A. Enrici, L. Apvrille, R. Pacalet, ACM Trans. Des. Autom. Electron. Syst. **22**(2), 34:1-34:25 (2017).
13. A. Saifan, J. Dingel, *Advanced Techniques in Computing Sciences and Software Engineering*, (Springer Netherlands, 283-288, 2010).
14. K. Jensen, L. Kristensen, *Coloured Petri Nets. Modeling and Validation of Concurrent Systems*. (Springer, Berlin, Heidelberg, 2009).
15. M. Szpyrka, J. Biernacki, and A. Biernacka, Arch. Control Sci. **26** (3), 339-365 (2016).
16. M. Markiewicz, L. Gniewek, "Conception of Hierarchical Fuzzy Interpreted PETRI Net", Stud Inform Control **26**, 2 (2017)
17. OMG Unified Modeling Language (OMG UML), Infrastructure V2.4.1 (2011).
18. OMG Systems Modeling Language (OMG SysML), V1.3 (2012).
19. K. Thramboulidis, G. Frey, IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA) 1-8 (2011)
20. M. Jamro, D. Rzonca, Comput Ind. **96**, 1-9 (2018)
21. B. Vogel-Heuser, D. Schutz, T. Frank, C. Legat, Mechatronics **24**, 7 (2014)
22. D. Schutz, C. Legat, B. Vogel-Heuser, *12th IEEE International Conference on Industrial Informatics* (INDIN) 267-273 (2014).
23. M. Hirsch, *Hallenser Schriften Zur Automatisierungstechnik*. Logos Verlag Berlin GmbH (2010).
24. *IEC 61499-1 - Function blocks - Part 1: Architecture* (2005).
25. J. Fernandes, R. Machado, H. Santos, *Proceedings of the Eighth International Workshop on Hardware/Software Codesign* (CODES) 18-22 (2000).
26. M. R. Fernandez, I. G. Alonso, E. Z. Casanova, J Intell Robot Syst. **86**, 3 (2017)
27. M. Jamro, *Progress in Automation, Robotics and Measuring Techniques: Control and Automation* (Springer International Publishing, 2015)
28. P. Gaj, J. Jasperneite, M. Felser, Ieee T Ind Inform. **9**, 1 (2013)
29. J. Stoj, *Communications in Computer and Information Science*, (Springer International Publishing 291 (2012)
30. M. Jamro, D. Rzonca, *Computer Networks: CN 2015* ser. *Communications in Computer and Information Science* (Springer International Publishing **522**, 139-149 (2015).
31. M. Jamro, D. Rzonca, *Computer Networks*, ser. *Communications in Computer and Information Science* (Springer International Publishing **431**, 147-156 (2014).
32. M. Jamro, W. Rzasa, D. Rzonca, Comput Ind. **71** (2015)
33. M. Jamro, Ieee T Ind Inform. **11**, 5 (2015).