Computers in Industry 96 (2018) 1–9

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

Agile and hierarchical round-trip engineering of IEC 61131-3 control software

Marcin Jamro[a], Dariusz Rzonca[b,*]

[a] TITUTO Sp. z o.o. [Ltd.], Zelwerowicza 52G Street, 35-601 Rzeszow, Poland
[b] Rzeszow University of Technology, al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland

ARTICLE INFO

Article history:
Received 2 July 2017
Received in revised form 7 October 2017
Accepted 9 January 2018
Available online xxx

Keywords:
Control software
IEC 61131-3
Implementation
Modeling
Round-trip engineering
Synchronization
Testing

ABSTRACT

The control software often performs complex, important, and responsible operations in industrial manufacturing systems. The size and complexity of such software are still increasing, thus it is important to provide engineers with methods and tools that simplify the development process. The situation can be improved by modeling using the Model-Driven Development (MDD) approach, standardized implementation process, as well as various testing methods. In the paper, the authors propose the further step, which consists of the comprehensive agile hierarchical round-trip engineering approach dedicated to the IEC 61131-3 control software. It divides the project into four connected parts – model, configuration, implementation, and tests. Such a solution allows developers to work independently and iteratively on various project parts, because changes are discovered automatically and are propagated to suitable views inside the project. The synchronization mechanism has been introduced into the CPDev engineering environment for programming industrial controllers.

1. Introduction

The control software is commonly used in many areas of industry. It is typically based on Programmable Logic Controllers (PLCs) that perform a crucial role in automated and semi-automated manufacturing systems [1]. Such a kind of software performs important and complex tasks, which improper operation may cause serious negative consequences, including danger for operators.

Nowadays, requirements for projects could change frequently and need to be adjusted even quite late in the development process. Apart from it, there is a visible trend of replacing hardware implementation with software [2]. What is more, complexity of software is still increasing. As stated by Vanderperren et al., the complexity of embedded software is doubled every 10 months [3]. Currently, more than 80% of embedded system features are conducted by software [4].

Similar observations are possible for control systems. As mentioned by Estevez et al., integration, reuse, flexibility, and optimization are required [5]. Alvarez et al. indicate that clear methodologies and procedures are demanded [6]. Thus, it is necessary to introduce concepts that can simplify and shorten the development process, as well as limit a number of errors existing in the application running on a plant.

Modeling, automatic code generation, standardized implementation, as well as testing may be beneficial and increase the overall control software quality. However, in case of complex systems, there is often a necessity of making modifications in various project parts, even after generating a part of code from the model. For instance, a developer could need to adjust code of a few units after their generation, but it is not convenient to introduce the same changes in the model manually. Without proper mechanisms, the system can be moved into the unsynchronized state and its further usage can be really complicated. Introducing the reverse engineering [7] into the development process can solve some problems by generating the model from the current implementation code. However, a much better solution is the round-trip engineering (RTE) [8], which tracks changes made in the project and propagates them to leave the project in the synchronized state.

In this paper, the authors propose the agile hierarchical round-trip engineering approach to the IEC 61131-3 control software development process. It supports tracking changes and propagating them into the model, configuration, implementation, and tests. The synchronization can be performed either automatically or semi-automatically. The concept makes it also possible to adjust the order of development stages according to developer's preferences and the project specificity.

* Corresponding author.
E-mail addresses: marcin@tituto.com (M. Jamro), drzonca@kia.prz.edu.pl (D. Rzonca).

https://doi.org/10.1016/j.compind.2018.01.004
0166-3615/

To make understanding of the proposed RTE concept easier, the running example is presented and explained while describing the approach. It is a very simple alarm system consisting of three motion detectors, implemented as state machines. By default, the system is stopped. After engaging by the user, all sensors are activated and monitor suitable areas. If any motion is detected, the alarm is started. It can be stopped by switching off the system.

The paper is organized as follows. In the next part, the supported development process is briefly presented. In Section 3, related work is described. Section 4 shows the proposed RTE concept to the IEC 61131-3 control software. The following part (Section 5) presents a simple example of the RTE process with explanation of automatically performed operations in the project. The next four parts specify stages of the concept related to propagating changes from model (Section 6), configuration (Section 7), implementation (Section 8), and tests (Section 9). The proposed approach has been introduced into the CPDev engineering environment, which is presented in Section 10.

2. Supported development process

The proposed RTE concept supports the software development process that consists of modeling, standardized implementation, and precise testing. Such operations could allow to significantly increase the product quality and are shortly described in this section.

2.1. Modeling with code generation

The modeling can be applied using the Model-Driven Development (MDD) [9] or Model-Driven Architecture (MDA)[1] [10] approaches. A model can be created in a suitable modeling language, such as Unified Modeling Language (UML) [11] or Systems Modeling Language (SysML) [12]. The latter is based on UML and provides developers with a possibility of modeling a wider range of system parts – not only software, but also hardware and requirements [13]. Such benefits are important, because allow modeling various parts of the project in the same language, using a known modeling environment. With the usage of model, the project can be presented with less or more details. What is more, views can be easily changed by using a hierarchy of diagrams.

The MDD approach can also allow generating an implementation of the whole system or its particular parts, based on the model. This approach could be applied to the control software area, such as presented by Thramboulidis et al. in [14]. It is worth mentioning that generating implementation code can be addressed variously, such as using SysML [15], Timed-Message Based Part State Graph [16], or Petri Nets [17]. If necessary, UML model of the software part of the industrial automation system can be extracted from the SysML system model and refined to get the implementation code [18]. The extensions of UML as specialized UML profiles for the automatic generation of some software layers, e.g., machine-to-machine communications [19], are also possible.

2.2. Standardized implementation

Apart from modeling, the standardized implementation is another factor that could improve the software quality. One of very popular standards in the control software area is IEC 61131-3 [20]. It is a worldwide standard, which defines a software structure and five languages (textual, graphical, and mixed) for programming industrial controllers, including PLCs. The control system is composed of Program Organization Units (POUs), namely programs, function blocks, functions, and classes. Programs are assigned to particular tasks, which are executed using resources, such as controllers.

The IEC 61131-3 standard is popular and well-known in industry. According to the survey conducted by Colla et al., the IEC 61131-3 languages are used by about 40% of the MEDEIA project partners [21]. Other ways of programming controllers involve the IEC 61499 standard [22], as well as C, C++, Java, domain-specific languages, and their combinations [23]. The software architecture integrating IEC 61131-3 and IEC 61499 modules in DCS solutions has been proposed in [24]. Its key benefit is reusing of the IEC 61131-3 software in the IEC 61499 projects. In such a case, the round-trip engineering concept, described in the current paper, could be utilized for automatic creation of interconnect elements, such as Service Interface Function Block templates for IEC 61499 code, basing on the IEC 61131-3 part.

2.3. Precise testing

Implementation allows preparing the code related to control algorithms. However, it is crucial to check its correctness, which could be done by the precise testing. As stated by Hametner et al. [25], the agile, systematic, and automation-supported testing is necessary in case of control software. A proper testing process can significantly increase a project quality by many ways, such as by performing the regression testing that can simplify finding errors introduced in existing project parts, while developing new features. There are several methods of software testing, especially related to the business one.

Unfortunately, the process of control software testing is significantly less systematic. Currently, it is frequently performed by manual testing [26]. What is more, developers need to verify some specific aspects, characteristic to this kind of software, such as hardware- and communication-related issues. Various methods can be used to increase testing possibilities dedicated to control software, including unit tests oriented to POUs [27,28], performance tests measuring times of communication between devices in real distributed control systems [29] or reliably simulated in the Timed Colored Petri Nets formalism [30], as well as measuring execution times of POUs [31]. Among other solutions, the deterministic replay debugging approach exists that allows tracing running programs on a controller and later analyzing their execution off-line using the development environment [32]. Software implemented fault injection testing approach, tailored for IEC 61131-3 programs, for automatic generation of test cases has been also proposed [33].

3. Related work

The proposed RTE concept involves many research areas, including modeling, configuration, testing, implementation, as well as the round-trip engineering and synchronization between various views of the project. What is more, these subjects can be described for several domains of the software engineering, including development of embedded and control systems. In this section, only work related to RTE is presented, however, it is not limited to the control software to present a wider view of the current research in this area. More information about related work regarding modeling and testing is shown in [15,28], while subjects of performance testing are presented in [29,31].

Hettel et al. [34] present the round-trip engineering formal definition and semantics of changes made in the target model. They focus on partial and non-injective transformations. The authors formally define the model, synchronization, round-trip transformation, as well as atomic and complex change. Some of the presented

[1] http://www.omg.org/mda/specs.htm.