A Model-Based Approach to Calculate Maintainability Task Lists of PLC Programs for Factory Automation

Kiana Busch, Jannis Rätz, Sandro Koch,
Robert Heinrich, Ralf Reussner
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{kiana.busch, sandro.koch, heinrich, reussner}@kit.edu,
jannis.raetz@student.kit.edu

Suhyun Cha, Matthias Seitz,
Birgit Vogel-Heuser
Technical University of Munich (TUM)
Garching, Germany
{suhyun.cha, matthias.seitz, vogel-heuser}@tum.de

***Abstract*—As long-living systems, automated Production Systems (aPS) have to be adapted due to optimization and inclusion of new features in their life cycle over decades. aPS consist of electrical, mechanical, and software components, which have a complex interaction and mutual dependencies. Consequently, these heterogeneous components have to be maintained together. Thus, the change propagation analysis in aPS is a challenging task. Existing approaches to change impact analysis lack tool-support and require expert knowledge in the aPS, as well as in the machine under study and its environment. This paper presents a tool-supported approach to change propagation analysis in aPS based on initial change requests. Our approach calculates a list of maintainability tasks to implement change requests in control programs deployed on Programmable Logic Controllers (PLC). To evaluate the quality and coverage of the generated task lists, we applied our approach to a community case study.**

***Index Terms*—Change Impact Analysis, PLC Program, IEC**

I. Introduction

Automated Production Systems (aPS) are subject to changes due to changing requirements and laws during their life cycle [2]. Therefore, expandability and maintainability are important characteristics of PLC-controlled aPS [3]. aPS consist of heterogeneous hardware and software components from different disciplines influencing each other during the operation. As these different disciplines co-evolve, they have to be considered together when analyzing the change propagation in aPS [3].

Modifications in one machine component from a discipline can lead to a high impact on other components of the machine, or even in other disciplines [3]. Thus, the adaptation of aPS to new requirements affects not only the hardware, but also the software and their communication. A change to the machine, such as adding a sensor to handle different types of work pieces, may cause a different behavior of the machine [4]. This change can lead to the adaptation of the PLC program [3]. Thus, planning the changes and estimating their costs are difficult tasks that require expertise in aPS and their environment [5].

This work was supported by the DFG under the Priority Programm SPP1593 (RE1674/12-1, VO937/29-1). The content of this paper has been developed in the context of the thesis of Jannis Rätz [1].

The approach of Prähofer et al. [6] uses feature models and feature to code mapping to analyze the change propagation in industrial automation software. Delta extraction approaches, such as [7], uses variants to analyze the change propagation. However, these approaches require reliable expert knowledge. The model-based approaches to change propagation (e.g., [8], [9]) calculate differences and inconsistencies between models. However, they also require users' expertise to determine relevant changes for adaptation. To version the different engineering results, the approach of Biffl et al. [10] extracts the data elements of AutomationML (AML) and links the results to an AML tree of the overall machine. However, most approaches lack tools to automatically analyze change propagation and extract task lists to implement change requests.

This paper addresses the change propagation analysis in PLC programs following International Electrotechnical Commission (IEC) 61131-3 – a standard for programming PLCs [11]. We propose Karlsruhe Architectural Maintainability Prediction for IEC (KAMP4IEC) – a model-based approach to analyze the effects of an initial change in PLC programs following IEC 61131-3. The analysis of KAMP4IEC results in a task list containing the potential tasks needed to implement the initial change requests. The generated task lists allow aPS experts to plan the change implementation process and estimate the change effort more precisely. Thus, KAMP4IEC supports the decision-making process. Our approach applies a methodology to change impact analysis [12], which is a generalization of Karlsruhe Architectural Maintainability Prediction (KAMP) [5]. KAMP is a model-based approach to change impact analysis for the component-based software architecture. An empirical study on the quality evaluation of the generated task lists by KAMP shows that a tool-supported approach speeds up the analysis of the change propagation even in smaller software models compared to manually creating task lists by a factor of 1.98 [5].

The contributions of this paper are: i) a model-based approach assisting the change propagation analysis in an IEC 61131-3 programmed PLC software, ii) metamodels for PLC

software according to IEC 61131-3 standard tailored to change propagation analysis of KAMP4IEC, and iii) the evaluation of KAMP4IEC to present its applicability and the quality of its task list. The evaluation presents the precision and the recall of the generated task list compared to manually created ones based on change scenarios for a lab-size community case study.

The paper is structured as follows: In Sec. II, we give an overview of our foundation. We present our approach in Sec. III. Sec. IV explains the structure of the evaluation. Related work is addressed in Sec. V, followed by the conclusion in Sec. VI.

II. Foundation

A. Domain-spanning Change Impact Analysis

aPS contain mechanical, electrical, and software components. The approach Karlsruhe Architectural Maintainability Prediction for aPS (KAMP4aPS) [3] applies the methodology for change impact analysis in mechanical and electrical components [12]. This paper proposes KAMP4IEC focusing on the change propagation in PLC software. The application of KAMP4aPS and KAMP4IEC in combination allows analyzing the change propagation from the hardware to the PLC software.

B. IEC 61131-3

The IEC 61131-3 [11] is a standard for programming the software deployed on PLC. The standard allows developing different dialects based on the specification. KAMP4IEC is based on the dialect used by the IEC development tool CodeSys V3.1 [13]. This dialect is very close to the original specification. Further, it offers a run-time environment for PLC simulation. However, in principle KAMP4IEC can be extended to other dialects of the IEC standard. Therefore, the metamodels and the corresponding change propagation rules must be adapted.

According to IEC 61131-3, a PLC software comprises three types of main elements: program, function, and function block. A *program* controls the whole machine, whereas *function blocks* represent parts of the program. *Functions* serve as stateless helper. Programs can contain global variables. *Global variables* provide access to inputs and outputs (e.g., sensors). Programs and function blocks are stateful by granting access to global variables.*Configuration*s manage programs and global variables [11], [13]. We consider the Object-Oriented Programming (OOP) extensions of IEC 61131-3 [14]. They include *Method* and *Property* as subelements of function block and *Abstract Method* and *Abstract Property* as subelements of *Interface*.

III. KAMP4IEC

KAMP4IEC calculates the change propagation in an IEC 61131-3 program based on dependencies between IEC model elements. Similar to the KAMP approach [5], it consists of three phases: i) In the **preparation phase**, the aPS experts model the PLC program based on the provided IEC metamodel, additional information (e.g., test cases), and the initial change requests (hereinafter called *seed modifications*). Once the IEC program is modeled, the model can be used to predict the impact of future change requests and compare different design decisions. ii) In the **impact phase**, KAMP4IEC automatically calculates the propagation of changes in the PLC program model triggered by seed modifications. The result of this phase is a temporary task list containing all potential changes in the model of the PLC software. iii) The **post-analyze phase** enriches the temporary task list with the modeled additional information. This results in a task list containing maintenance tasks needed to implement the initial change requests. In the following we discuss each phase of the approach in more detail.

Application Example

As running example, we show the propagation of a seed modification in the eXtended Pick and Place Unit (xPPU). The xPPU is developed at Technical University of Munich (TUM) as a demonstrator and a common case study for aPS evolution [4], to which several approaches in aPS can be applied. All relevant engineering artifacts of the xPPU are available [3]. The xPPU is an extensible machine, which in its most basic form consists of a stack for work pieces, a crane for transporting the work pieces, and a ramp as the end point for work pieces [4]. Based on the PLC program of xPPU we discuss in the following sections the analysis steps of KAMP4IEC.

A. Preparation Phase

In this phase, the aPS expert has to model the IEC program structure, additional information and the seed modifications according to the KAMP4IEC metamodel.

1) Modeling IEC Program Structure: To model the structure of the PLC program, we provide a metamodel describing the structure of an IEC software. This metamodel allows specifying the elements of an IEC software that are relevant for the maintainability analysis. It is split in two further metamodels: i) a repository metamodel and ii) a system metamodel.

The *repository metamodel* contains IEC elements that need to be specified once. These elements are then referenced in the program. This metamodel includes programs, global variables, function blocks, functions, interfaces, abstract methods, methods, abstract properties, and properties. Fig. 1 gives an excerpt of the repository metamodel. It illustrates program, global variable, function block, interface, and their relationship.

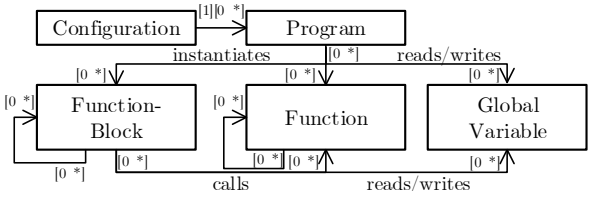

Fig. 1. An excerpt from IEC repository metamodel

Fig. 2 illustrates the relationships between methods, abstract methods, properties, and abstract properties. A function block can contain methods and properties, whereas an interface can contain abstract methods and abstract properties.

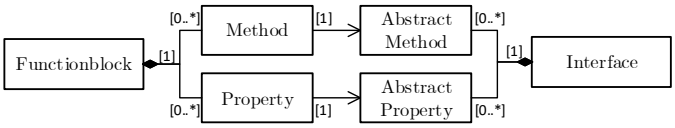

Fig. 2. An excerpt from IEC repository metamodel

The *system metamodel* contains an IEC configuration. The model elements specified in the IEC repository model can be referenced in the system model. Consequently, the system metamodel has a composite structure. The relations between configuration, program, and other metamodel elements are shown in Fig. 1. The modeling effort depends highly on the granularity of models. Although the effort for coarse-grained models could be lower than fine-grained ones, the fine-grained models can improve the precision of the analysis. Modeling leads to extra effort in early development stages. It is expected that program models reduce the work in subsequent stages [15].

2) Modeling Additional Information: A change affects not only the structure of a PLC program, but it may result in organizational and technical activities. These activities also account for the impact of implementing a change request [5], [3]. Providing additional information about the source code files, metadata files, tests, and the responsible staff improves the effort estimation. Using such information, the task list can be extended to include concrete working steps and annotations for responsible staff. This aims at automatically identifying efforts caused by additional information (e.g., rerunning the tests and adapting source code).

3) Change Request: We provide a further metamodel allowing modeling the modified elements. Each element in this metamodel references an element in the IEC metamodel. This way, the seed modifications can be modeled. Our approach supports change requests on a high abstraction level (e.g., changing the PLC program) and on a low abstraction level (e.g., changing a method). The modeled seed modifications should be minimal to avoid overestimation [16].

Application to the xPPU: We model the IEC program of the xPPU and additional information. The change request is exchanging the binary start button of the machine with a rotary one to switch between different production modes. The aPS expert selects as seed modification the global variable `IN_START_BUTTON` in the program.

B. Impact Phase

This phase automatically analyzes the change propagation based on a change propagation algorithm (cf. Algorithm 1). It calculates the impact of the seed modification on the PLC program model. A change propagation algorithm consists of a set of change propagation rules [12]. A change propagation rule describes from which metamodel element the change can propagate to which metamodel element [12]. In general, the rules need to be defined by the aPS expert.

Algorithm 1 iterates over all seed modifications. For each seed modification "Element A" all corresponding change propagation rules are applied. The change propagation rules are given in. Table I. Each row shows from which metamodel element (i.e., elements in first column) to which metamodel element (i.e., elements in the second column) a change can propagate [17]. After applying the change propagation rules, the algorithm adds the newly affected elements to the temporary task list. This task list does not contain any duplicates.

Algorithm 1 Algorithm for the Change Propagation Analysis

```
Require: Model of the program structure, list of seed modifications
Temporary task list = ∅
while There is at least one unprocessed seed modification referring metamodel element A do
    for all Elements B accessing the modified element A and element A is from the first column of Table I and element B is from the second column of the same row as element A do
        if Temporary task list does not contain element B then
            Add element B to temporary task list
            if Element A is a function block or an interface then
                Add Element B to the list of seed modification
            end if
        end if
    end for
end while
```

Application to the xPPU: Algorithm 1 iterates in the while loop over all unprocessed seed modifications (i.e., `IN_START_BUTTON`). Then, it iterates in the for loop over all model elements that access `IN_START_BUTTON`. If these model elements are from the first row of Table I (i.e., configurations, function blocks, methods, and programs), Algorithm 1 adds these model elements to the temporary task list. For example, there is a configuration model element, in which the global variable is defined and a program model element, which reads from the global variable. These model elements are added to the temporary task list.

Element A	Element B
Global Variable	Configuration, function block, Method, Program
Function	Function, Function Block, Method, Program
Function Block	Abstract Method, Abstract Property, Function, Function Block, Global Variable, Method, Program, Property
Interface	Abstract Method, Abstract Property, Function, Function Block, Global Variable, Interface, Method, Program, Property
Abstract Method	Function Block, Method, Program
Abstract Property	Function Block, Method, Program, Property
Method	Function Block, Method, Program
Property	function block, Method, Program

TABLE I
LIST OF SIMPLE PROPAGATION RULES IN KAMP4IEC [17]

If the seed modification is a function block or an interface, the change can propagate through the whole program. In this case, Algorithm 1 calculates the affected elements referencing the modified function block or interface and adds them to seed modifications [17]. The algorithm recursively calculates the change propagation. In all other cases, Algorithm 1 calculates the change propagation in one iteration to avoid false positives.

Application to the xPPU: As the seed modification is the global variable (i.e., neither a function block nor an interface), we can abort the change propagation after one iteration.

The algorithm terminates, if there are no unprocessed seed modifications. The result is a temporary task list containing all potential modification to implement the change request.

Application to the xPPU: The temporary task list contains `IN_START_BUTTON` and the affected model elements such as configuration, program, function block, and method.

C. Post-analyze Phase

The result of the impact phase is a temporary task list. Each task references a modified element in the PLC model. The post-

analyze phase enriches the task list with activities regarding additional information modeled in the preparation phase. These activities aims at estimating the change effort more precisely.

Application to the xPPU: If the aPS expert modeled additional information such as source code files for PLC program, corresponding metadata, or test cases for the IEC program in our example, KAMP4IEC extends the task list by changing the corresponding source code files, metadata files, or test cases.

IV. Evaluation

KAMP4IEC analyzes the impact of seed modifications on the model of an IEC 61131-3 program. In this section, we evaluate the quality and coverage of the generated task lists by applying KAMP4IEC to the community case study xPPU.

A. Study Design

We used a Goal Question Metric (GQM) plan to evaluate KAMP4IEC. The **Goal** is to evaluate the quality of the generated task lists compared to manually created reference task lists. After comparing both task lists, we categorize each task of the generated task list into the following groups: i) *True Positive* (t_p) tasks are contained in both task lists. ii) *False Positive* (f_p) tasks are included in the generated task list, but not in the reference task list. In other words, the generated task list contains elements, which are not actually modified. iii) *False Negative* (f_n) tasks are included in the reference task list, but not in the generated task list. KAMP4IEC aims at avoiding false negatives at the expense of an overestimation of affected model elements (i.e., false positives). iv) *True Negative* (t_n) tasks are not included in both task lists.

We define **Question 1** as: How precise and complete are the generated task lists in comparison to the reference task lists? To answer this question we use two metrics: **Metric 1.1**: $precision = \frac{t_p}{t_p+f_p}$ and **Metric 1.2**: $recall = \frac{t_p}{t_p+f_n}$. Metric 1.1. (i.e., precision) relates the number of true positives to the number of false positives. Metric 1.2. (i.e., recall) relates the number of true positives to the number of false negatives. Thus, both metrics neglect the number of all model elements [16].

To consider the number of all model elements we formulate **Question 2** as: Does the application of KAMP4IEC reduce the number of model elements, which the aPS expert needs to consider during the impact analysis? We define the following metrics: **Metric 2.1** is the ratio of the number of model elements referenced by the reference task list to the number of all model elements n: $r_t = \frac{t_p}{n}$. **Metric 2.2** is the ratio of the number of all model elements referenced by the generated task list l to the number of all model elements: $r_g = \frac{l}{n}$ [16].

B. Change Scenarios

To evaluate KAMP4IEC, we choose nine change scenarios, which represent equivalence classes of change requests that are relevant for change propagation. All scenarios are applied to a model of the IEC program of xPPU. There are several change scenarios for xPPU, describing the machine's evolution [4]. An aPS machine is composed of four element types: *Structure*, *Component*, *Module* and *Interface* [3]. The first three scenarios are partially based on change scenarios of [4] and present changes to structures, components, and modules. The other scenarios cover the functionalities of remaining rules and the relevant IEC model elements for the maintainability.

1) Change Scenario 1 (CS1) – Structure: We consider a basic version of xPPU and add a whole structure to it – the stamp. Thus, the seed modifications are the corresponding function blocks and global variables.

2) Change Scenario 2 (CS2) – Module: In this scenario three microswitches modules indicating the direction of the crane are replaced by one potentiometer. This results in replacing three global variables with a new one and adapting all elements accessing the global variables.

3) Change Scenario 3 (CS3) – Component: This scenario represents the running example.

4) Change Scenario 4 (CS4) – Function block: We delete a function block performing some simple arithmetic operations, as the program is restructured and its functionalities will be implemented in other elements. The function block accessed by several IEC model elements. The seed modification is the function block. All elements accessing it will be affected.

5) Change Scenario 5 (CS5) – Function: We analyze the impact of deleting the helper function `CheckGreaterEquals` checking whether one of the values is greater than or equal to the other values. The seed modification is the function. All elements accessing this function will be affected.

6) Change Scenario 6 (CS6) – Interface: During the refactoring we delete an interface. The seed modification is the interface. All elements accessing or implementing it or its abstract methods and properties are affected.

7) Change Scenario 7 (CS7) – Method: We change the return type of the abstract method `IUnitMode.Hold` from boolean to the more flexible enum. The seed modification is `IUnitMode.Hold`. All elements accessing it or methods implementing it will be affected.

8) Change Scenario 8 (CS8) – Property: We change the return type of the abstract property `IUnitmode.Active` from boolean to int. The seed modification is `IUnitmode.Active`. All elements accessing or properties implementing `IUnitmode.Active` will be affected.

9) Change Scenario 9 (CS9) – Program: We rename the program `MAIN` to `xPPU`, as the PLC runs different IEC programs. The seed modification is `MAIN`. A change to the program represents a coarse-grained change. This change can propagate to the elements containing or accessing the program. The configuration managing `MAIN` will be affected, as we have one configuration in our PLC program. In general, the seed modification should be kept as small as possible.

C. Results

Table II shows the results of the evaluation. The first two columns present the results regarding Metric 1.1 and 1.2 to answer Question 1, the last two columns show the results regarding Metric 2.1 and 2.2 to answer Question 2. Each row describes the evaluation results for a change scenario.

The precision of the generated task list in CS1 was 88.9% due to the overestimation, mentioned in Sec. III. To avoid false negatives (i.e., a recall of 100%), we accept some false positive modifications. The reason for the overestimation is general change propagation rules that generate some false positives to avoid missing tasks in the task list (i.e., false negative). The false positives in CS1 occurred for example due to dependencies between the methods of an affected function block to the methods of another function block. A further metric is the ratio of the number of model elements, which have to be changed, to the number of all model elements (i.e., r_t). r_t is in this scenario 0.138. KAMP4IEC marks 0.155 of all model elements as changed. Thus, our approach reduces the number of model elements that the aPS experts need to consider during the impact analysis. In other scenarios, the values of precision and recall are 100%. This means, that the generated task lists do not have any false positives. Further, no element from the reference task list is missing in the generated task list. Thus, the values of r_t and r_g are equal. These values show that only a minimal part of the model had to be considered during the change propagation analysis. Thus, the generated task list for CS2 to CS9 was equal to the reference task list.

	Precision	Recall	r_t	r_g
CS1	0.889	1.000	0.138	0.155
CS2	1.000	1.000	0.052	0.052
CS3	1.000	1.000	0.021	0.021
CS4	1.000	1.000	0.05	0.05
CS5	1.000	1.000	0.021	0.021
CS6	1.000	1.000	0.037	0.037
CS7	1.000	1.000	0.021	0.021
CS8	1.000	1.000	0.017	0.017
CS9	1.000	1.000	0.009	0.009

TABLE II
RESULTS OF THE EVALUATION

D. Threats to Validity

As described in [18], we consider four classes of validity:

1) Internal Validity: As KAMP4IEC is based on models, the accuracy of the model affects the accuracy of the change propagation analysis [16]. IEC 61131-3 supports different dialects. We chose the dialect of CodeSys V3.1 [13], as it is very close to the original standard. The dialect also allows the OOP extensions to IEC 61131-3 [14], which allows modularizing the PLC program and creating a detailed program model. However, programs written in other dialects may need other IEC metamodels. These programs could be transformed into IEC programs, which are based on the OOP extensions. Further, the provided IEC metamodels and change propagation rules can in principle be extended to other IEC dialects.

2) External Validity: We applied KAMP4IEC to a lab-size community case study, namely xPPU. The results of a case study might not be transferable to all other PLC programs (e.g., PLC programs that are not based on IEC 61131-3). However, the case study allows a better understanding of the evolution of a PLC, in comparison to an experiment.

3) Construct Validity: As KAMP4IEC is a model-based approach, a model of the analyzed program conform to the given IEC metamodel has to be created. However, other models could also be converted to an instance given IEC metamodel. Another option is changing the IEC metamodel. In this case, the change propagation algorithm needs to be changed based on the changed metamodel. The evaluation change scenarios represent equivalence classes of metamodel elements, which are relevant for the change propagation analysis.

4) Conclusion Validity: The evaluation results are based on statistical metrics (e.g., precision or recall). This way, we aim at avoiding different interpretation by other researchers.

E. Assumption and Limitation

KAMP4IEC derives a task list including IEC model elements that are potentially affected by an initial change request. KAMP4IEC overestimates the maintenance tasks (i.e., false positives) to prevent false negatives (e.g., in the case of function blocks or interfaces). If the task list would contain many false negatives, the aPS expert has to manually analyze the change propagation [16]. Nevertheless, the evaluation shows that the generated task list is precise in most cases. KAMP4IEC analyzes the change propagation using static references based on the proposed metamodel. The metamodel contains the elements relevant for the maintainability on an abstract level. Therefore, other elements (e.g., user defined data types) have to be mapped to the proposed metamodel. It is also possible to extend the metamodel and the corresponding change propagation rules to include new elements and dependencies.

V. RELATED WORK

This section proposes related work on metamodels and approaches to change impact analysis in aPS.

A. Metamodels for aPS

In the field of aPS, model-based system engineering has been regarded as an important methodology [19] to handle the complexity in aPS development and metamodels. It has been introduced for different engineering aspects in aPS. Especially for the control software in IEC 61131-3, Thramboulidis [20], Katzke et al. [21] (UML-PA for process automation), and Witsch et al. [22] (plcML) adapted UML from the software field to the automation field for PLC code design and generation, as well as the deployment of software to hardware as early approaches. They provided a metamodel for IEC 61131-3 and plcML as the UML profile and suggested a mapping methodology between the metamodels. Marcos et al. suggested a validation methodology for industrial control system in automated way [23], [24]. The authors presented metamodels of Industrial Process and Measurement and Control Systems (IPMCS) using 3+1 SysML view-model architecture of Thramboulidis suggested in [25]. In addition to these metamodels, Marcos et al. [26] presented a UML metamodel in order to describe modeling elements needed for reconfiguration of the control system. Estvez et al. [27] proposed a markup language of automatic generation of PLC automation projects from component-based models. The approach of [26] generates software for service-oriented systems (SOA) based on a metamodel description. Feldmann et al. [28] presented a

metamodel to capture the dependency of the software for software analysis framework. A recent approach [29] presents a change impact information acquirement as one step of system test prioritization. In this analysis, a change impact analysis algorithm is developed for indirect influences on the control software based on the extended dependency model from [28].

B. Change Propagation Analysis in aPS

Change impact analysis approaches in aPS can be divided into two categories. i) *Configuration-based* approaches: Prähofer et al. [6] propose a feature-oriented modeling environment. They use a feature to code mapping to estimate the change effort. The approach of [30] extracts variability information from related variants of a product line and map the features and their interactions to code. However, the change impact analysis and task list generation have to be done manually. The delta extraction approach of [7] transforms variants into each other by a delta. The delta has to be created semi-automatically, which is an error-prone task. ii) *Model-based change detection* approaches: [2] proposes an approach to automatically creation of a behavior model based on the input and output of a system. Other approaches such as [8], [9] use UML models to create a list of deltas between two system states and find inconsistencies in the design. The framework of [8] offers visualization tools for deltas. Thus, human experts can determine relevant changes, find duplicates, and resolve conflicts during merging the deltas. Dam et al. [9] present a general approach for difference calculation in UML models. The user can define rule violations for the UML model. In case of a modification a repair plan can be calculated to make the model valid again. As these approaches are domain independent, rules for IEC 61131-3 have to be defined. Biffl et al. propose a model-driven approach to version the different results and link them to a structure tree of the overall machine in [10]. Egyed et al. [31] analyze the change propagation from the perspective of transformation to achieve consistent model with the change. However, these approaches do not offer automatically change propagation or task list derivation.

VI. Conclusion and Outlook

This paper presented KAMP4IEC – a tool-supported approach to change propagation analysis for PLC programs following the IEC 61131-3 standard. We use models to represent the structure of the IEC program on an abstract level, additional information, and initial change requests. Based on these models, KAMP4IEC analyzes the propagation of the initial change requests in the program model. It calculates a task list referencing the changing elements in the model of the PLC program. The task list allows aPS experts planning the change implementation process. We evaluated the precision and coverage of the generated task lists based on the representative change scenarios of the common case study xPPU.

As future work, we plan to analyze the effects of changes to electrical design and extend the evaluation to include the adaptability metrics. We will integrate our approach to engineering tools for cost estimation and programming.

References

[1] J. Rätz, "Erweiterung eines Wartbarkeits-Frameworks für die Programmiersprache IEC 61131-3," 2017.

[2] J. Ladiges *et al.*, "Automated determining of manufacturing properties and their evolutionary changes from event traces," *Intelligent Industrial Systems*, 2016.

[3] B. Vogel-Heuser *et al.*, "Maintenance effort estimation with kamp4aps for cross-disciplinary automated production systems - a collaborative approach," in *IFAC*, 2017.

[4] B. Vogel-Heuser *et al.*, "Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the Pick and Place Unit," Tech. Rep., 2014.

[5] K. Rostami *et al.*, "Architecture-based assessment and planning of change requests," in *ACM QoSA*, 2015.

[6] H. Prähofer *et al.*, "Feature-oriented development in industrial automation software ecosystems: Development scenarios and tool support," in *IEEE INDIN*, 2016.

[7] M. Soto, "Delta-p: Model comparison using semantic web standards," *Softwaretechnik-Trends*, vol. 2, no. 27-30, p. 87, 2007.

[8] C. Pietsch *et al.*, "Sipl–a delta-based modeling framework for software product line engineering," in *IEEE/ACM ASE*, 2015.

[9] H. K. Dam and M. Winikoff, "Supporting change propagation in uml models," in *IEEE ICSM*, 2010.

[10] S. Biffl *et al.*, "Linking and versioning support for automationml: A model-driven engineering perspective," in *INDIN'15*. IEEE.

[11] IEC, "61131-3:programmable controllers–part 3:programming languages."

[12] R. Heinrich *et al.*, "A methodology for domain-spanning change impact analysis," in *Euromicro Conference on SEAA*. IEEE, 2018.

[13] J. T. AG. Codesys - entwicklungssoftware für industriesteuerungen. [Online]. Available: https://www.janztec.com/embedded-pc/codesys

[14] B. Werner, "Object-oriented extensions for iec 61131-3," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 36–39, 2009.

[15] H. Koziolek, "Parameter dependencies for reusable performance specifications of software components," Ph.D. dissertation, KIT.

[16] K. Rostami *et al.*, "Architecture-based change impact analysis in information systems and business processes," in *IEEE ICSA*, 2017.

[17] K. Busch *et al.*, "A cross-disciplinary language for change propagation rules," in *14th CASE*. IEEE, 2018.

[18] P. Runeson *et al.*, *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Publishing, 2012.

[19] B. Vogel-Heuser *et al.*, "Evolution of software in automated production systems: Challenges and research directions," *JSS*, vol. 110, 2015.

[20] K. C. Thramboulidis, "Using uml in control and automation: a model driven approach," in *IEEE INDIN*, 2004, pp. 587–593.

[21] U. Katzke and B. Vogel-Heuser, "Combining uml with iec 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems," *IFAC'07*, vol. 40, no. 16.

[22] D. Witsch and B. Vogel-Heuser, "Plc-statecharts: An approach to integrate uml-statecharts in open-loop control engineering aspects on behavioral semantics and model-checking," *IEEE IFAC*, vol. 44, no. 1, 2011.

[23] M. Marcos *et al.*, "Analysis and validation of iec 61131-3 applications using a mde approach," in *IEEE ETFA*, 2010, pp. 1–8.

[24] E. Estevez and M. Marcos, "Model-based validation of industrial control systems," *IEEE TII*, vol. 8, no. 2, pp. 302–310, 2012.

[25] K. Thramboulidis and A. Buda, "3+1 SysML view model for IEC61499 function block control systems," *IEEE INDIN*, pp. 175–180, 2010.

[26] R. Priego *et al.*, "On applying mde for generating reconfigurable automation systems," in *IEEE INDIN*, 2015, pp. 1233–1238.

[27] E. Estévez *et al.*, "Automatic generation of plc automation projects from component-based models," *The International Journal of Advanced Manufacturing Technology*, vol. 35, no. 5, pp. 527–540, 2007.

[28] S. Feldmann *et al.*, "Analysis framework for evaluating plc software: An application of semantic web technologies," in *IEEE ISIE*, 2016.

[29] S. Ulewicz and B. Vogel-Heuser, "Industrially applicable system regression test prioritization in production automation," *IEEE Transactions on Automation Science and Engineering*, pp. 1–13, 2018.

[30] L. Linsbauer *et al.*, "Variability extraction and modeling for product variants," *SoSyM*, vol. 16, no. 4, pp. 1179–1199, 2017.

[31] A. Egyed *et al.*, "Fine-tuning model transformation: Change propagation in context of consistency, completeness, and human guidance," in *ICMT*. Springer, 2011, pp. 1–14.