Towards Model-driven Verification of Robot Control Code using Abstract Syntax Trees in Production Systems Engineering

Kristof Meixner[1], Dietmar Winkler[1], Petr Novák[2] and Stefan Biffl[3]
[1]*Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle, TU Wien, Vienna, Austria*
[2]*Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Czech Republic*
[3]*Information & Software Engineering Group, Institute of Information Systems Engineering, TU Wien, Vienna, Austria*

Keywords: Production Systems Engineering, Industrial Robots, Verification and Validation, Engineering Models, Abstract Syntax Tree.

Abstract: *Context.* In *Production Systems*, software components are often tightly connected to defined hardware device types like robots. Different types of robots, even from the same vendor, often use vendor-specific programming languages. Therefore, the exchange of devices or device types, e.g., during system evolution, is challenging and needs new or adapted control software and repeated verification and validation process steps, even if the software behavior remains unchanged. Models aim at supporting these verification and validation tasks during system evolution. *Objective.* This position paper aims at providing a verification and validation process approach with models for supporting automation systems maintenance and evolution processes. For evaluation purposes, we report on a feasibility study with a focus on two selected robot types in the context of *Production Systems Engineering (PSE)*. *Method.* We use the *Abstract Syntax Tree (AST)* concept as a foundation for generating models as the basis for human-based verification and validation. Based on two generated *AST* variants, related to old and new software control code, human experts can compare the behavior of the expected system to verify and validate the code. *Results.* First results showed the feasibility of the *AST* concept to support human-based verification and validation in the context of *PSE* maintenance projects on a structural level. *Conclusion.* Although the human-based verification and validation process is feasible and promising on a structural level, the complexity of *AST* for large-scale models needs to be addressed by tool support to overcome complexity levels of the production system and limitations of human-based verification and validation.

1 INTRODUCTION

The increasing share of software code in *PSE* projects require appropriate testing approaches (Schafer and Wehrheim, 2007) to evaluate the expected behavior of the production system. A production system typically includes production resources, like robots, shuttles, conveyors, or specified tools to produce a product (Chan and Spedding, 2003). Fig. 1 shows a schematic overview of a production system with a rail conveyor and four robots, located at the Czech Technical University in Prague[1]. During the engineering of production systems typical engineering processes, such as waterfall-like or V-model like process approaches are applicable (Wasson, 2015). Common process steps include requirements elicitation and systems design, system construction, implementation, test and commissioning as well as operation and maintenance (Winkler et al., 2017). Maintenance engineers execute tasks like regular services, repair, or evolution of the system in the system maintenance phase. In *PSE*, individual devices often use vendor-specific programming languages to implement the required logical behavior of the device, e.g., of a robot. In practice, we observed various programming languages even for robots manufactured by the same vendor. Thus, during the evolution of the system, e.g., exchanging a robot or upgrading it from one version to another, control code must be manually rewritten, verified and validated. Therefore, these human-based activities are time-consuming and error-prone and could lead to significant downtimes of the system. Nevertheless, a structured human inspection task based on models can help to improve this process (Aurum et al., 2002).

[1]Testbed 4.0: www.ciirc.cvut.cz/testbed

Meixner, K., Winkler, D., Novák, P. and Biffl, S.
Towards Model-driven Verification of Robot Control Code using Abstract Syntax Trees in Production Systems Engineering.
DOI: 10.5220/0007484104020409
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, pages 402-409
ISBN: 978-989-758-358-2

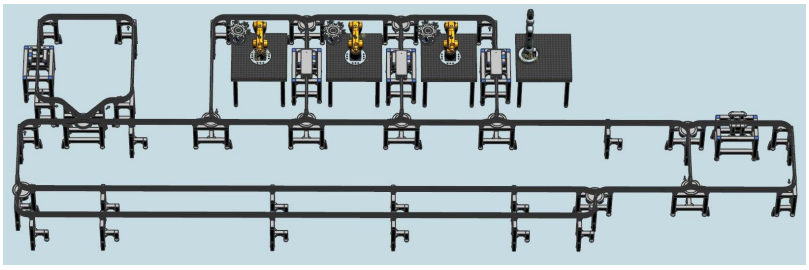

Figure 1: Industry 4.0 Testbed used as a use-case.

In this position paper, we address the key question, **how to verify and validate the behavior of the system of different robot control code implementations** to improve system maintenance and evolution and to support quality assurance experts and engineers in their verification and validation process. Therefore, we propose a model-driven verification and validation approach that makes the evaluation of software control code more effective and efficient. Furthermore, we conceptually evaluate the proposed approach in a feasibility study in a real-world industry setting, i.e., exchanging a robot in an existing production system within a system evolution project.

The key idea is to abstract from the software control codes and evaluate generated abstract representations in a human-based review process step: (a) automatically generate *AST* models based on existing software control code, i.e., an *AST* model based on the previous version of the robot control code and an *AST* model based on the newly written control code; and (b) compare both *AST* variants for deviations in the code structure. As both robot control programs aim to address similar requirements and system behavior, both *AST* variants have to follow a similar structure, depending on the programming language. Deviations could represent defects that need to be addressed before the deployment of the modified system. Note that in the first step of this initial prototype, the comparison task solely relies on human experts, i.e., a quality assurance engineer, who actually perform the verification and validation process step as part of a human inspection process. In the future of this research, this process step is planned to be supported by tools. Furthermore, we need to mention that the *AST* model should be based on a shared model that uses generic resp. common concepts (Winkler et al., 2017), to support a unified representation that allows implementing tools that can be used for a variety of control languages.

To demonstrate the feasibility of the proposed approach, we use the *"Industry 4.0 Testbed"*, a testing facility for evaluating new approaches and methods in the context of the 4th industrial revolution (Biffl et al., 2016). The most conspicuous part of the Industry 4.0 Testbed is a monorail transportation system *Montrac*, see Fig. 1. On this rail system, several shuttles transport small pallets with material, semi-products and final goods from and to industrial robots. The system is equipped with three *KUKA KR Agilus* robots and one *KUKA LBR iiwa* robot (www.kuka.com), performing production operations according to a given production plan. Both types of robots use different programming languages. Even if two robots of different types execute a similar operation from the physical point of view (similar systems behavior), their programs are different concerning the programming language and the program structure. If the production system configuration changes, i.e., by exchanging one robot type by another robot type, it is necessary to re-program the robotic program. It requires significant human effort not only for re-programming but also for testing.

This position paper represents a starting point for supporting engineers to make *PSE* projects more efficient and effective. In this paper, we focus on the verification and validation process of robot software code in the context of a system evolution process.

2 RELATED WORK

This section presents related work on *PSE*, models used in the *PSE* domain, and *ASTs* as an representation of source code.

2.1 Production Systems Engineering

Production systems typically consist of mechatronic objects and incorporate multiple engineering disciplines within the product life cycle (Moser et al., 2012). Engineers of different disciplines use various programming languages and various models for problem description and problem-solving. Synchronization of related subsystems is typically executed by applying a bunch of interfaces that overcome technical and semantic heterogeneity of tools and data models

(Winkler et al., 2017).

In context of a production system, the control logic is typically implemented by a *Programmable Logic Controller (PLC)*. A family of *PLC* programming languages is standardized as IEC 61131. Based on this standard, *PLC* are normally programmed with *Structured Text*, a programming language comparable with *PASCAL*, *ladder diagrams* based on relay logic, or *function block diagrams* (Collins, 2007). However, there are numerous extensions and variations that are vendor-dependent. Another standardization effort in the frame of a standard PLCopen is trying to bridge the gap between various *PLC* programming implementations[2]. Higher levels than PLCs are frequently implemented with *Python*, *C#*, or *Java*, data acquisition can be implemented in the *C* language.

Industrial robotics typically utilizes proprietary languages. For example, the robot vendor KUKA (www.kuka.com) uses the language KRL for the traditional industrial robots, which represents the most significant part of KUKA portfolio, whereas the new advanced type of cooperative robots uses the Java language. Due to the use of a large set of different programming languages in *PSE*, the verification and validation of correct operations and correct systems behavior are challenging. This is also an issue during maintenance and evolution, and even minor updates or changes of the system during production system life-cycle implies high testing effort and costs.

Therefore, new approaches are needed for verification of robot control code in *PSE* to be able to verify that corresponding code snippets in different languages are equivalent from the behavior point of view.

2.2 Models and Data Integration in *PSE*

Models for production systems can be represented in various formats, such as XML-based representations, one of the most commonly used data formats.

AutomationML[3] (Drath et al., 2008) is an XML-based and standardized data format (IEC 62714) for representing engineering knowledge in the area of process automation and control. *AutomationML* aims at integrating a set of growing standardized data representations such as CAEX for plant topology information (IEC 62424), COLLADA[4] for geometry and kinematic information and PLCopen XML for logic information.

The process of data integration in industrial automation is standardized in ISA-95 (Unver, 2012). ISA-95 (resp. IEC 62264) and focuses on vertical integration of automation tools and data models within the automation pyramid. However, ISA-95 does not provide a communication language but rather a methodology to define data models for the integration of MES and ERP systems.

Semantic models for industrial systems can be represented in an *OWL*[5] ontology according to the standard ISO 15926 (Kim et al., 2011). Although the standard originally addressed issues in the oil and gas industry, the concepts are applicable also in other types of production systems. The original version of the standard utilized *EXPRESS* language, but due to its limited tool support, the *OWL* system description was added, see online at the POSC Caesar Association[6] for details.

2.3 Abstract Syntax Tree

Control software is written in various programming languages including general-purpose ones such as *Python*, *C#* , or *Java*; or particular specialized languages for robot, *PLC*, and other device programming. As each of such programming languages uses its specific syntax, the engineers implementing the software need to understand these characteristics.

An *Abstract Syntax Tree (AST)* abstractly represents the essential formal structure of a software artifact, by leaving out parts of the code which are specific to the programming language, like indention or parentheses (OMG, 2011) (Jones, 2003).

As example, Fig. 3 shows on the middle left-hand side parts of an *AST* of a `switch-case` statement that switches a variable `GI_PROG_NR`. This abstract representation of source code can support developers, who are not familiar with a particular programming language, to understand the behavior of a program.

To generate an *AST* out of source code, the code first needs to be parsed with a parser for the specific language, like *ANTLR* (Parr and Quong, 1995), and, afterward, be translated to a particular *AST* format. This format can either be a visual representation such as *GraphML* (Brandes et al., 2001) or a model representation format such as *EMF* (Steinberg et al., 2008) for further processing of the *AST*. A model representation, which has also be proven to be successfully used for program analysis in the industrial context (Grimmer et al., 2016), is the *Abstract Syntax Tree Model (ASTM)* of the *Object Management Group (OMG)* (OMG, 2011). Other approaches were to parse the source code directly to an ontology (Atzeni and Atzori, 2017) aiming at the generation of

[2]PLCOpen: www.plcopen.org/

[3]AutomationML: www.automationml.org

[4]COLLADA: www.khronos.org/collada

[5]OWL: www.w3.org/OWL

[6]PCA: www.posccaesar.org/wiki/ISO15926inOWL

linked data from source code and querying these data structures.

Using such a representation of a *AST* supported by proper tools can help developers and engineers to efficiently analyze code from different programming languages or even different domains. Therefore, this approach seems to be promising in *PSE* for effective and efficient verification and validation of different robot control code variants.

3 RESEARCH ISSUES

Based on the need for verification and validation support, discussions with industry experts, and related work, we identified the following research questions.

RQ.1 *How can a maintenance process support the verification and validation of software control code for different programming languages?* After engineering the production system, the second big phase in the *PSE* life-cycle is the *Operation & Maintenance* phase that includes control and change activities of the production system during its run-time and aims at optimizing the production processes (Lüder et al., 2017). In this phase, for example, worn off parts of a production system such as robots need to be replaced in a maintenance, modernization, or evolution project. In the context of this paper we focus on exchanging a robot type within a production system (using different programming languages) without changing the behavior of the system. Thus, *RQ.1* focuses on identifying processes that enable the verification of control software that is implemented in different programming languages.

RQ.2 *To what extent is the AST capable of supporting PSE developers during the verification and maintenance process?* An *AST* represents the formal structure of a program without the specifics of a programming language, like indentation or punctuation rules. The *AST* itself can be contained in a particular model or represented in different ways such as tree structure, a graph-like visualization or an ontology. However, programs in different languages can also have different visualizations due to the particular keywords, method calls or the program structure. Thus, *RQ.2* focuses on characteristics that allow evaluating how an *AST* is capable of supporting engineers during the verification process within a system maintenance project.

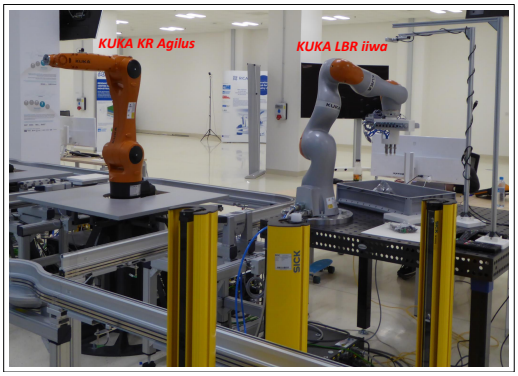

Figure 2: Two types of robots in Industry 4.0 Testbed.

4 ILLUSTRATIVE USE CASE

In this section, we introduce an illustrative use case, i.e., the “Industry 4.0 Testbed” that serves as a foundation for illustrating the solution approach in the context of a maintenance project. Fig. 2 illustrates the use case including two different robot types within a production system.

These two robot types consist of a *KUKA KR Agilus* and a *KUKA LBR iiwa* robot (see www.kuka.com for more details). The key characteristics of the involved robot types are: The *KUKA KR Agilus* robot type is a fast industry robot frequently used in high-performance environments such as car assembling in the automotive industry sector. For safety and security reasons, barriers, such as a cage or optical sensors, are needed to prevent physical damage or human injuries. *KUKA KR Agilus* uses the vendor-specific *KUKA WorkVisual IDE* and the programming language *KRL*. Fig. 3 shows a source code example of *KRL* on the bottom left hand side.

The second robot type is a *KUKA LBR iiwa*, which is able to cooperate with humans and its environment and, thus, should not do any harm. Therefore, there is no need for a safety zone, like a cage or optical sensors. For programming, this robot type uses the *KUKA Sunrise Workbench IDE* that is build on top of the *Eclipse Framework* and *Java* as a programming language.

The underlying use case focuses on a maintenance and evolution project with the goal to exchange the *KUKA KR Agilus* robot with the modern *KUKA LBR iiwa* robot. The benefits of this evolution project include: (a) that humans can cooperate with robot, e.g., put raw material into the manufacturing line or hand-over the final products without considering robot safety zones; and (b) that humans can continuously check the quality of semi-products and the manufacturing process as they can inspect the robot from

a small distance (even within the safety zone).

Because both robot types should have similar system behavior but use different programming languages, engineers have to rewrite, verify and validate the robot control code from *KRL (KUKA KR Agilus)* environment to Java *(KUKA LBR iiwa)* environment. To support engineers in doing these cost-intensive tasks, this position paper focuses on the verification and validation activity of engineers to ensure similar system behavior for both robot types, i.e., whether or not the original software control code in KRL behaves similarly to Java implementation.

5 SOLUTION APPROACH

This section explains our solution approach for a human-based maintenance process in a *PSE* environment that is based on an abstract model of the control logic source code.

In the *Maintenance & Operation* phase of an *PSEs*, engineers, among other activities, need to renew and modernize engineering units (Lüder et al., 2017) such as robot arms, which also means that the type of the unit can change. In this case, a challenge during the maintenance phase is to verify that the functional behavior of the unit to be replaced equals that of the newly installed unit. We, therefore, propose a structured maintenance process that enables a model-based validation and verification of the control code that replaces the old control code.

Fig. 3 shows an overview and the main parts of our solution approach and is divided into two sections. The upper part of the figure explains our proposal for a human-centered maintenance process for *PSE* including the relevant activities, labeled in green from *A* to *E*, that have to be executed by different stakeholders. The lower part of the figure shows, labeled in orange from *1* to *3*, the steps that we implemented to derive an *AST* model, which can be used for validation and verification, from snippets of the control logic source code of the two robot arms introduced in Section 4 and their *ASTs* which are used in the maintenance process.

In (Lüder et al., 2017) the authors sketched some general activities and corresponding artifacts that are relevant during the *Maintenance* phase of a production system, however, a process for maintenance activities is only scarcely described. To address this issue, we developed a structured process for the maintenance of software and control code in the *PSE* domain, which is based on a general software maintenance process (Yau et al., 1988) and a process for testing software (ISO 29119, 2013). The process is shown as activity diagram in the upper part of Fig. 3 and described in the following.

- First, the process is started with a *Maintenance Planning* activity, labeled with *A* in the figure, where a *Planner* defines the objectives of the phase and identifies necessary tasks to be done. This can be, for example, the modernization of a worn off robot arm that also requires a rescheduling of the production plan due to temporarily missing resources. The next three activities (*B* to *D*) are part of a subprocess called *Maintenance Execution*.
- Second, the tasks identified in the planning activity need to be prepared and executed in the *Maintenance Implementation* activity (*B*) by *Developers* of different domains. This phase includes, for example, the implementation of the robot behavior for the new robot type.
- The third phase (*C*) is divided into two tasks. On the one hand, the *Validation & Verification* task, is executed by a *Quality Assurance Engineer*, which aims at checking the validity and quality of the implementation. On the other hand, this phase contains, in our case, a *Human Inspection* of the *AST* model to check the correctness of the behavior. If this *Human Inspection* fails, it is fed back to the *Validation & Verification* task which reports the issue back to the developers and starts a new *Maintenance Implementation* cycle.
- In the *Deployment* activity (*D*), after the *Validation & Verification* task was completed successfully, the code artifacts are deployed to the control nodes, and the replacement units are installed in the production system.
- The final activity (*E*) of our *Maintenance Process* is the *Operation* task that includes the startup of the production system with the new configuration and checks whether everything works as expected after maintenance and is executed by the *Operations* team.

To support the *Validation & Verification, the* task of the *Maintenance Process* we chose a model-based approach that exploits the *AST* of the implemented control codes. The goal was to leave out irrelevant parts of the control code and concentrate on the structural parts that represent the behavior of the programs. This abstract representation of code as models should enable engineers, without the specific knowledge of the particular programming languages, to easier inspect whether the behavior of the programs is similar. For our approach, we decided to utilize the language-independent *ASTM* (OMG, 2011) provided

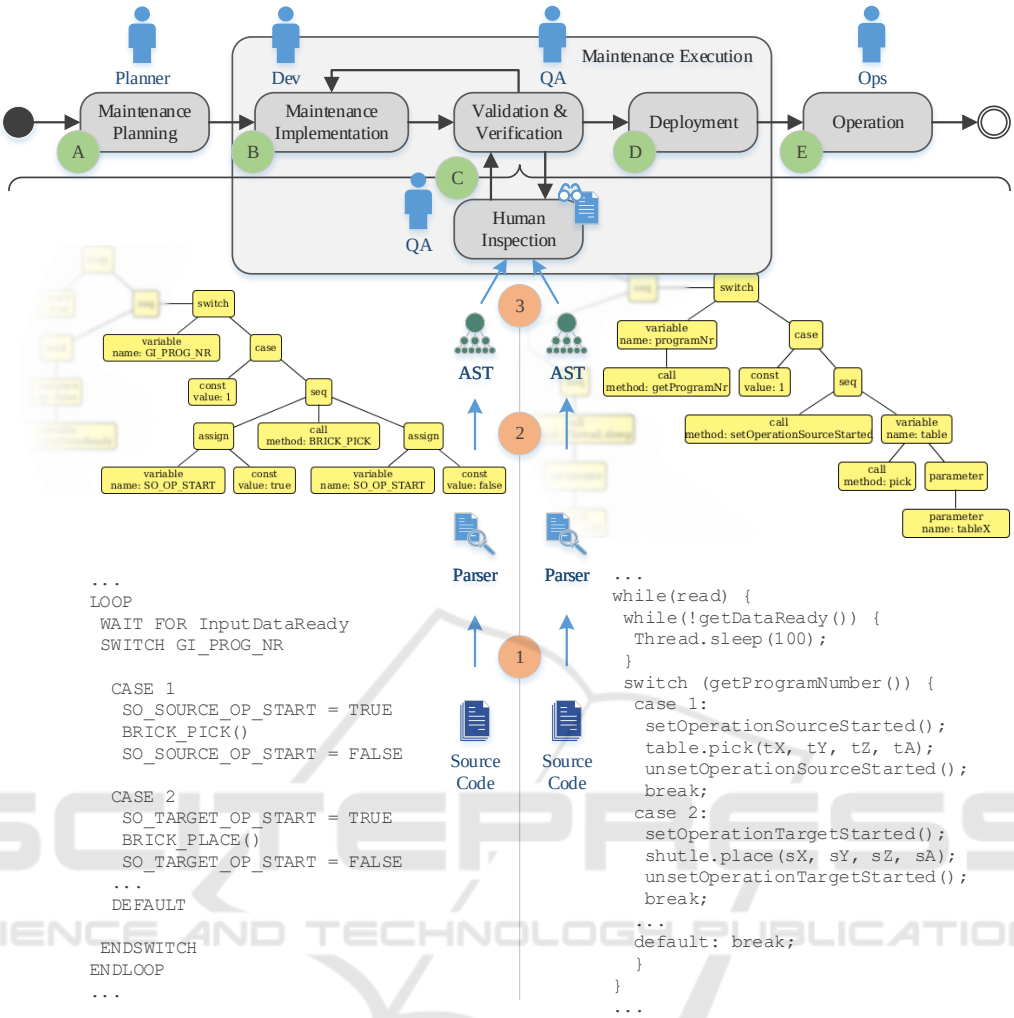

Figure 3: Maintenance process for *PSE* and creation of *AST* model from robot control code.

by the *OMG*. This model establishes a *Generic Abstract Syntax Tree* that provides common concepts for modeling programming languages. Furthermore, it introduces a *Specific Abstract Syntax Tree (SAST)* for particular languages. However, to this point in time, the *SAST* needs to be implemented individually or at least a mapping needs to be provided for a particular programming language. This model allows a flexible creation of models from source code and additionally includes an *EMF* description that easily allows using the model in software applications. To create the model-based *AST* visualizations of control codes the following steps are necessary.

- In a first step, labeled *1* in Fig. 3, the *Source Code* is parsed by a *Parser* specific to the particular programming language of the control code. For this step either a parser exists, or it has to be created with the help of a parser framework.
- The second step, labeled *2*, translates resp. maps the internal model of the parser to the common *AST* model, which is in our case the *ASTM*.
- The last step, labeled *3*, includes the visualization of the language-independent *ASTM* in a proper format, like *GraphML*, and the provision to the *Quality Assurance* team that executes the *Human Inspection*.

6 EVALUATION

To show the feasibility and evaluate our approach we selected a prototypical case for the exchange of the two robot arms introduced in Section 4 during a production system maintenance phase.

We, therefore, first parsed the control code for each of the robot arms, that is shown at the bottom of Fig. 3. While the code on the left hand is a snippet of the *KUKA KR Agilus* robot, which is programmed

using the *KRL* language, the snippet on the right hand is a similar part of the *KUKA LBR iiwa* robot code, which is implemented in Java. The parsing models can then mapped to a Java implementation of the *ASTM*, which we created from the provided *EMF* description with the help of the Eclipse IDE. Using the *ASTM* allowed us not only to run several checks on the model to compare the *ASTs*, but also to visualize the models with the same methods and frameworks. The visualization of the *ASTs* displayed in Fig. 3 at the label *2* was done using *GraphML* and the freely available editor *yEd*[7]. We were then able to visually compare the *ASTs* of the two robot control codes that were implemented in the two different programming languages.

Table 1: Identifying strengths and weaknesses of different approaches. ++ = high, - - = low, × = not supported, ✓ = supported, Code = Effort for Code Implementation, V & V = Validation and Validation, AST = Abstract Syntax Tree.

		As Is	AST Based	AST Tool Supported
Effort	Code	++	++	++
	V & V	++	+	-
Engineering Knowledge	Code	++	-	- -
	V & V	++	-	- -
Separation of Roles	Code	×	✓	✓
	V & V	×	✓	✓

After the implementation, we compared the traditional approach of comparing the control code with our *AST*-based approach. The results of our findings are summarized in Table 1 and act as a basis for a conceptual evaluation that also includes a future version of our approach that includes tool support for the comparison. The table groups the traditional approach *As Is*, our *AST*-based approach and our *AST*-based with improved tool support, as well as the three categories (a) *Effort* that is invested for writing as well as validating and verifying the control code, (b) *Engineering Knowledge*, which is needed for the comparison of the code and the validation and verification, and (c) *Separation of Roles*, which means how well the different identified roles involved are separated during the maintenance process.

The *Effort* to write the control code for the robot arms remains high, resp. does not change, for all three approaches. However, the *Effort* to validate and verify the control code is slightly better in our approach than in the actual approach of code comparison. We expect the *Effort* to be even lower when the *AST*-based approach is supported by tools. The *Engineering Knowledge* in the actual approach needs to be high to compare the old with the new behavior of the robot arm. Using our approach the details of the programming languages are omitted which simplifies the comparison and lowers the effort. Once again, with proper tool support, we expect the effort to be even lower. The *Separation of Roles* is given in our *AST*-based and the tool-supported *AST*-based approach, however, in the current approach the developer of the code and the quality assurance engineer are strongly intertwined.

[7]yEd Graph Editor: www.yworks.com/products/yed

7 DISCUSSION AND CONCLUSION

This section discusses the findings of the solution approach and the evaluation in the context of the research questions and presents the limitations. The main goal of this position paper is to support engineers in the verification and validation process within a maintenance process of a *PSE* project.

With the focus on **RQ.1**, we explained that although robots are constructed as universal and flexible devices, changing from one type to another can mean to re-program the entire control code of the robot. Therefore, a maintenance process in the *PSE* domain have to include software code construction and verification and validation. The adapted maintenance process builds on a *AST* approach to compare different abstract representations of the related software control code. The verification and validation of software control code for different programming languages can significantly reduce time and costs for testing and commissioning of the new robot.

Addressing the research question **RQ.2**, we found out that an *AST* is a suitable formalism for verification and maintenance process. However, it is not enough to compare the corresponding *ASTs* structurally (i.e., from the syntax point of view). To conduct a robust comparison, the corresponding *ASTs* have to be rather analyzed and compared in a complex way including knowledge about the semantics of the languages behind. This fact poses a main limitation of the proposed method. Furthermore, more complex software control code will result in more complex *AST* models which could hardly be handled by humans, also because of the possibility of a differing structure of the models. Therefore, tool support is needed to support human inspection in the verification and validation task.

Limitations of the approach includes semantic meaning of *AST* models, derived from software con-

trol code and the complexity of result *AST* representations which require tool support for searching, filtering, and similarity detection. Further limitations focus on the evaluation where we present some preliminary investigations on in a small use case by using small code snippets. In the context of this position paper, the further investigation remains for future work.

Therefore, as *future work*, we plan to investigate in more detail the benefits and limitations of the proposed approach, including automated tool support. Furthermore, we plan to address such a semantic comparison of corresponding *ASTs*. Finally, we will focus on considering languages of other robot vendors such as ABB or FANUC.

ACKNOWLEDGEMENTS

The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. The research done by Petr Novák has been supported by the DAMiAS project funded by the Technology Agency of the Czech Republic.

REFERENCES

Atzeni, M. and Atzori, M. (2017). Codeontology: Rdfization of source code. In *International Semantic Web Conference*, pages 20–28. Springer.

Aurum, A., Petersson, H., and Wohlin, C. (2002). State-of-the-art: software inspections after 25 years. *Software Testing, Verification and Reliability*, 12(3):133–154.

Biffl, S., Lüder, A., and Winkler, D. (2016). *Multi-Disciplinary Engineering for Industrie 4.0: Semantic Challenges and Needs*, pages 17–51.

Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., and Marshall, M. S. (2001). Graphml progress report structural layer proposal. In *International Symposium on Graph Drawing*, pages 501–512. Springer.

Chan, K. and Spedding, T. (2003). An integrated multidimensional process improvement methodology for manufacturing systems. *Computers & Industrial Engineering*, 44(4):673 – 693.

Collins, K. (2007). *PLC programming for industrial automation*. Exposure.

Drath, R., Luder, A., Peschke, J., and Hundt, L. (2008). Automationml-the glue for seamless automation engineering. In *ETFA 2008.*, pages 616–623. IEEE.

Grimmer, A., Angerer, F., Prahofer, H., and Grunbacher, P. (2016). Supporting program analysis for non-mainstream languages: Experiences and lessons learned. In *2016 IEEE 23rd Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER)*, pages 460–469. IEEE.

ISO 29119 (2013). INTERNATIONAL STANDARD ISO / IEC / IEEE Software and systems engineering — Software testing — Part 2: Test processes.

Jones, J. (2003). Abstract syntax tree implementation idioms. In *Proceedings of the 10th conference on pattern languages of programs (plop2003)*, pages 1–10.

Kim, B. C., Teijgeler, H., Munc, D., and Han, S. (2011). Integration of distributed plant lifecycle data using ISO 15926 and Web services. *Annals of Nuclear Energy*, 38:2309–2318.

Lüder, A., Schmidt, N., Hell, K., Röpke, H., and Zawisza, J. (2017). *Identification of Artifacts in Life Cycle Phases of CPPS*, pages 139–167. Springer International Publishing, Cham.

Moser, T., Mordinyi, R., and Winkler, D. (2012). Extending mechatronic objects for automation systems engineering in heterogeneous engineering environments. In *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conf. on*, pages 1–8. IEEE.

OMG (2011). Architecture-driven Modernization : Abstract Syntax Tree Metamodel. [Online; 2018-11-12].

Parr, T. J. and Quong, R. W. (1995). Antlr: A predicated-ll (k) parser generator. *Software: Practice and Experience*, 25(7):789–810.

Schafer, W. and Wehrheim, H. (2007). The challenges of building advanced mechatronic systems. In *Future of Software Engineering (FOSE '07)*, pages 72–84.

Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson.

Unver, H. O. (2012). An isa-95-based manufacturing intelligence system in support of lean initiatives. *The International Journal of Advanced Manufacturing Technology*, pages 1–14.

Wasson, C. S. (2015). *System engineering analysis, design, and development: Concepts, principles, and practices*. John Wiley & Sons.

Winkler, D., Sabou, M., and Biffl, S. (2017). Improving quality assurance in multidisciplinary engineering environments with semantic technologies. In Kounis, L. D., editor, *Quality Control and Assurance*, chapter 8. IntechOpen, Rijeka.

Yau, S. S., Nicholl, R. A., Tsai, J.-P., and Liu, S.-S. (1988). An integrated life-cycle model for software maintenance. *IEEE Transactions on Software Engineering*, 14(8):1128–1144.