Catalog of Refactoring Operations for IEC 61499

Michael Oberlehner*, Lisa Sonnleithner*†, Bianca Wiesmayr*, Alois Zoitl*†, *Member, IEEE*
†*CDL VaSiCS*
**LIT CPS Lab, Johannes Kepler University Linz*
Linz, Austria
{michael.oberlehner, lisa.sonnleithner, bianca.wiesmayr, alois.zoitl}@jku.at

Abstract—**Refactoring is a key technology for improving the quality of a software system. A higher quality will lead to a clearer structure, which facilitates sharing work across multiple teams. This is important especially in the domain of cyber-physical production systems, where different engineering disciplines meet at their cutting point. IEC 61499 tries to abstract the increasing complexity of such systems. This work gives an overview of existing refactoring techniques from software engineering and provides a first insight on how to apply refactoring to IEC 61499 systems.**

Index Terms—**IEC 61499, Cyber-Physical Production System, Refactoring, Model-Driven Software Engineering**

I. Introduction

Complexity increases during software evolution unless work for maintenance or restructuring is done, states Lehman's law number 2 of software evolution. [1]. An essential part during evolution is to refactor the structure of the system to reduce complexity.

"Refactoring is the process of changing a software system in a way that does not alter the external behavior of the code, yet improves its internal structure" (Fowler, 1999 [2]) The main goal of refactoring is to improve the design and readability of a software system. To reduce that effort, tools that support refactoring features are needed. If refactoring is applied correctly, bugs in a software can be found and resolved more easily, because of the clearer program structure after refactoring. On the other hand, refactoring existing systems might also introduce new bugs. Therefore, a well-defined set of refactoring operations is needed. One issue is that refactoring requires a lot of effort, especially when executing the necessary steps manually. An Integrated Development Environment (IDE) can use the defined operations as a basis to perform the necessary steps automatically, which leads to a reduced amount of errors during refactoring.

Also within the domain of cyber-physical systems (CPSs), the amount of software is increasing. Hence, Lehman's law is also valid in that domain [3]–[5]. To handle the growth of software within an CPS, refactoring is unavoidable [6]. The industrial standard IEC 61499 describes how to model CPS as distributed applications. Refactoring within IEC 61499 is important to keep the developed applications resilient against evolution.

The literature provides first approaches for refactoring operations within the domain of IEC 61499 applications. Patil et. al. [7] are describing methods for refactoring IEC 61499 applications, like extracting states from the Execution Control Chart to separate function blocks, which leads to measurable improvements for the application. The work of [8] describes how to detect possible deadlock states in a basic function block's Execution Control Chart and how to resolve them with graph transformations. Another reason why refactoring is an important topic, is that it can be used to resolve Bad Smells in software. Bad Smells are code structures or design flaws that have a negative influence on software quality and that can lead to problems. The term was introduced by Fowler et al. [2], who proposed a set of Bad Smells and also suggested refactoring methods that can be used to resolve these Bad Smells. We want to investigate refactoring operations that are capable of resolving specific IEC 61499 Bad Smells. This results in a need for an IEC 61499 refactoring catalog that can be used by IEC 61499 development tools.

This paper addresses our ongoing work of inspecting existing contributions and gathering potential refactoring operations for IEC 61499. Common practices from textual languages should be reused or adapted. In his book, Fowler described more than 20 different refactoring operations [2] and collected them in a catalog. To the best of the author's knowledge, there is no refactoring catalog for IEC 61499.

II. Background: Refactoring

An IEC 61499 system is built with several software engineering technologies. Each of them has its own refactoring approaches. IEC 61499 is a Domain Specific Modeling Language (DSML) [9] that describes how to model distributed applications within cyber-physical systems. Hence, for restructuring IEC 61499 systems, refactoring operations for models are required. As the DSML allows textual algorithms, also the refactoring of source code needs to be considered. Besides textual programming, IEC 61499 systems are mainly developed graphically, so we need to take a closer look at refactoring operations for visual programming languages.

A. Refactoring in Textual Programming languages

Refactoring or restructuring source code is performed to enhance readability and maintainability of applications [2] [10]. A clearer structure of the source code increases its maintainability, e.g., simplifies adding new features. In textual programming languages such as Java, refactoring operations are widely researched and essential for software developers. The most common IDEs for Java development (Eclipse, IDEA, JBuilder,..) offer numerous features for performing automatic refactoring operations [11]. Changing the name of a function is a typical refactoring operation. After the name has changed at the function declaration, the next step is to search all occurrences within the text file, and to replace them with the new name. This can be time-consuming and error-prone if the function is widely used across the application. Such operations need to be automated by the development tool.

B. Refactoring in Model Driven Software Engineering

Model-Driven Software Engineering (MDSE) describes the separation between domain engineering and application engineering [12]. A model is an abstract representation of real world artifacts that should hide as much underlying source code as possible to reduce the complexity of a system [13]. As models represent evolving systems such as cyber-physical production systems, they also need to be restructured after major changes. Therefore, refactoring techniques are also required in MDSE. The most common way to refactor models is to apply graph transformations [14].

One example for MDSE is the Eclipse Modeling Framework (EMF). The framework provides the modeling language Ecore to model classes, for which Java Source Code can be generated. The works of [15] and [16] have shown how to apply model refactoring and they also described a tool set that is capable of performing the operations with graph transformation algorithms. UML also needs to be inspected, as this is the most researched topic in Model-Driven Software Refactoring [17].

Another example for refactoring a modeling language is the Business Process Model Notation (BPMN). Like models in systems engineering, BPMN process models evolve. First refactoring approaches have been discussed in the literature and feasible refactoring techniques for large amount of process instances have been introduced [18].

C. Refactoring in Visual Programming Languages

A visual programming language is a programming language that uses graphical elements for developing an application [19] [20]. MDSE can also be used to build visual programming languages. Examples of visual programming languages are MATLAB® / Simulink®, SysML, LabVIEW®, IEC 61131, or IEC 61499.

Sui et al. [19] are defining the term Dataflow Visual Programming Languages (DFVPLs). They provide a first categorization of refactoring compared to object-oriented languages. They also describe a refactoring process that might be automated. A concrete example of a DFVPLS is MATLAB® / Simulink®, which is used to represent software functionality as data flow diagrams. First approaches to refactor MATLAB® / Simulink® models have been discussed in [21], for example, the feature ”Merge Subsystems”. A subsystem is a group of blocks. If one wants to merge two subsystems, it is necessary to extract the blocks from the existing subsystems, move them to the new subsystem, and reconnect the signals. With the refactoring feature, it is possible to just select both subsystems and perform the refactoring operation “Merge Subsystem” and the tool takes care of reconnecting signals [21]. They also introduced a catalog of refactoring operations for MATLAB® / Simulink®.

III. Bringing refactoring to IEC 61499

This section categorizes the properties of IEC 61499 into software engineering disciplines. We want to point out that the proposed refactoring operations are for IEC 61499 application developers. This means that it is not the goal to refactor the IEC 61499 meta-model, as changes at this level are rare.

A. IEC 61499 and textual programming languages

Textual refactoring features are useful in algorithm development for certain kinds of function blocks. Simple Function Block (SFB) or Basic Function Blocks (BFBs) may contain algorithms written in a textual programming language. Therefore, IEC 61499 IDEs need to provide a text editor that offers refactoring features such as “Rename Variable”.

B. IEC 61499 and Model Driven Software Engineering

IEC 61499 is a DSML [9] which has 6 different models, namely the **system model**, **distribution model**, **device model**, **resource model**, **function block model**, and **application model**. This work will deal with the application model and the function block model, but the operations should be as generic as possible to extend them to the other models. As IEC 61499 is a DSML, existing refactoring operations from Model-Driven Software Refactoring will be used in the catalog. Existing tools for graph transformations should be reused. The main challenge is to bring an IEC 61499 application in a graph representation that fits the model of a tool.

C. IEC 61499 and visual programming languages

IEC 61499 is a visual programming language that has several unique characteristics compared to other visual languages. It is event-controlled and it has a data flow. IEC 61499 also has a type system, that allows developers to create their custom type and use it in the application. Like textual languages, we can say that it is statically typed, as every type is known before compiling. As every visual programming language, also IEC 61499 has nodes, wires, and containers, which fulfils the formal definition of a visual programming language [19].

IV. Catalog of Refactoring Operations for IEC 61499

The catalog is the result of collecting information from the literature throughout different software engineering disciplines. We also collected possible refactoring features during our contributions to Eclipse 4diac IDE [22]. The identified catalog is illustrated in Table I. The catalog is divided into refactoring groups and operations. The groups with their corresponding operations should provide a listing of operations, which can be used by IEC 61499 development tools, such as Eclipse 4diac IDE [22]. A chain of operations might be performed to alter the structure of an IEC 61499 application. The operations are grouped depending on where they can be applied. For instance, Execution Control Chart (ECC) refactoring can only be applied within FBs that contain an ECC. The refactoring operations are meant to be performed after a Bad Smell has been detected from [23].

A. *Application Refactoring*

An IEC 61499 application contains function blocks, event connections, and data connections. If one wants to restructure function blocks, also their corresponding connections (event and data) must be restructured. Application refactoring might be driven by certain quality metrics. So it is feasible to perform refactoring operations for restructuring applications to improve these metrics, which lead to a clearer structure of the program. For example if a group of function blocks is replaced by a subapp, it would be efficient to have a tool that detects all occurrences of that group and replaces them with a corresponding subapp type. This operations might be fully or partly automated. A possibility for partly automating would be to guide the developer through every occurrence and to ask them whether the operation should be performed.

B. *Function Block Refactoring*

IEC 61499 provides different types of function blocks, namely Service Interface Function Block (SIFB), BFB, SFB, adapter, Composite Function Block (CFB) or subapps. For different function block types, different refactoring operations are required. Basic FBs need refactoring operations for their execution control chart and their textually written algorithms. SFBs mainly need textual refactoring methods. Also adapters are an interesting topic for refactoring. A developer may want to split or merge adapters. Service sequence diagrams are describing the order of the incoming and outgoing events of a function block. So if the events are altered, this should also be synchronized with the sequence diagram.

C. *Subapp and CFB*

Subapps or CFBs require refactoring techniques to adapt their interfaces or to manipulate their inner network. They are used to group multiple function blocks into one unit, which itself is again a function block. Since subapps and CFBs only differ during execution on the runtime, the same refactoring operations can be used. Common operations that are time consuming for the developer are restructuring elements that are nested in subapp applications. When moving a block across subapp borders, all the connections need to be reconnected manually. As subapps can be nested into other subapps, it is possible to build multiple hierarchy levels across IEC 61499 applications. Refactoring within such nested applications would no longer be feasible without tool support. One example for subapp refactoring is to move multiple connected elements that are nested within a subapp one or more levels up. During the movement, the connections should be kept and the subapp interface needs to be adapted.

D. *Type Management Refactoring*

Every IEC 61499 application uses types from its custom or shared type library. Therefore, refactoring operations for managing the type library are essential. As the types are developed outside of the application, we need special mechanisms to refactor types and keep them in sync with the application. The IDE needs to keep track of all the instances of every type, so that it is capable of updating them if the type has been changed. The type update mechanism needs to be adapted to the developer's needs, as an automatic type update may destroy the application, especially when interfaces are changing. For this problem a sophisticated error handling is needed.

E. *Error-Based Refactoring*

As Version Control Systems (VCS) are used to share source files with different versions among developers, it is also necessary to think about the handling of conflicting files. If an application is broken, for example it uses a type that has been deleted from the type library, we try to repair that type as much as possible. That means that we infer type information at the interface from the opposite pin of the connection, or we can reuse the type name from the source file. It also happens that an interface pin of a type no longer exists. In this case, the connection is broken. Instead of just dropping the connection, the user should be able to handle that broken connection. There are two possibilities to resolve that problem. Either the user creates a new pin for the type, or the connection can be reconnected to another pin.

F. *Visual Refactoring*

In contrast to the previous groups, this refactoring group is not intended for restructuring applications. The goal of visual refactoring is to reorder the positioning of the graphical elements (FBs, pins, and connections) within an IDE to provide a better overview to the developer. Graph layouting algorithms can be used for this refactoring group. To get better results of the graph layouting algorithms, refactoring operations such as reordering the pins or stretching the margin between pins can be performed. This means that a connection between two pins has a shorter way if they are close to each other on the x-axis.

V. Conclusion and Future Work

The catalog that we proposed in the previous section should provide a set of refactoring operations for IEC 61499 IDEs. With the help of these operations, it will be easier to refactor IEC 61499 applications. This means that the quality of the developed software systems will rise and the applications will get a clearer structure which makes maintenance easier. It is not intended to provide a full catalog within this paper. It should outline our ongoing work that will be extended step-by-step.

In future work, we also want to combine our proposed refactoring catalog with the Bad Smell catalog that we introduced in [23] to make recommendations on which refactoring methods can be used to resolve a certain Bad Smell. Other important topics in refactoring are verification and validation. For that we want to introduce an error checking mechanism that performs a verification of the IEC 61499 application before and after

TABLE I
IEC 61499 CATALOG OF REFACTORING

Refactoring Group	Operations	Comment
Application	Substitute Function Blocks with a CFB	Discussed in [8]
	Substitute multiple connections by adapter	Discussed in [8]
	Search all occurrences of pattern and replace it with type	adapted from Replace Command with Function [2]
	Update type instances	
	Search duplicate patterns and replace them	
Function Block	Resolve a Dead Lock in the ECC	Discussed in [8]
	Extract algorithm to a separate state	Discussed in [24]
	Consolidate Similar ECC transitions into a separate FB	Discussed in [24]
	Create adapter	
	Merge/Split adapter	
	Synchronize Events with Service Sequence	
	Textual Refactoring within algorithms	Source code refactoring [2]
Subapp/CFB	Merge Subapp	Adapted from Matlab subsystem [21]
	Split Subapp	Adapted from Matlab subsystem [21]
	Move FB to parent and keep connections	Adapted from Matlab subsystem [21]
Type Management	Update type file when renaming file	
	Convert multiple pins to a structured type and replace the pins with this new type	
	Update all instances of a type after the type has been edited	
Error Based	Repair a broken connection	
	Repair a broken type	
Visual	Layout the function block diagram	
	Reorder pins	Adapted from Matlab block [25]
	Increase margin between pins	

applying a refactoring operation. This can be done with formal verification [26], testing, or OCL constraints. To point these errors out we want to provide a graphical representation with error markers, which will also offer first approaches that will repair the conflicts. This approach should be the base for error-based refactoring.

REFERENCES

[1] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski, "Metrics and laws of software evolution-the nineties view," in *Proceedings Fourth International Software Metrics Symposium*, 1997, pp. 20–32.

[2] M. Fowler, *Refactoring: Improving the Design of Existing Code (Object Technology Series)*, illustrated edition ed. Addison-Wesley Longman, Amsterdam, 1999.

[3] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

[4] B. Vogel-Heuser, C. Diedrich, A. Fay, S. Jeschke, S. Kowalewski, M. Wollschlaeger, and P. Göhner, "Challenges for software engineering in automation," *Journal of Software Engineering and Applications*, vol. 07, no. 05, pp. 440–451, 2014.

[5] M. Törngren and P. Grogan, "How to deal with the complexity of future cyber-physical systems?" *Designs*, vol. 2, no. 4, p. 40, 2018.

[6] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[7] S. Patil, D. Drozdov, G. Zhabelova, and V. Vyatkin, "Refactoring of iec 61499 function block application — a case study," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 15.05.2018 - 18.05.2018, pp. 726–733.

[8] V. Vyatkin and V. Dubinin, "Refactoring of execution control charts in basic function blocks of the iec 61499 standard," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 2, pp. 155–165, 2010.

[9] "Iec 61499 architecture for distributed automation: The 'glass half full'view," *IEEE industrial electronics magazine.*, vol. 3, no. 4, p. 7, 2009.

[10] T. Mens and T. Tourwe, "A survey of software refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[11] E. Mealy and P. Strooper, "Evaluating software refactoring tool support," in *Australian Software Engineering Conference (ASWEC'06)*. IEEE, 2006, pp. 10 pp–340.

[12] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 09 2012, vol. 1.

[13] Brooks, "No silver bullet essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[14] T. Mens, G. Taentzer, and D. Müller, "Model-driven software refactoring," in *Model-Driven Software Development*, J. Rech and C. Bunse, Eds. IGI Global, 2009, pp. 170–203.

[15] T. Arendt and G. Taentzer, "A tool environment for quality assurance based on the eclipse modeling framework," *Automated Software Engineering*, vol. 20, no. 2, pp. 141–184, 2013.

[16] Enrico Biermann, Karsten Ehrig, Christian Köhler, Günter Kuhns, Gabriele Taentzer, Eduard Weiss, "Emf model refactoring based on graph transformation concepts."

[17] M. Misbhauddin and M. Alshayeb, "Uml model refactoring: a systematic literature review," *Empirical Software Engineering*, vol. 20, no. 1, pp. 206–251, 2015.

[18] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers, "Refactoring large process model repositories," *Computers in Industry*, vol. 62, no. 5, pp. 467–486, 2011.

[19] Y. Y. Sui, J. Lin, and X. T. Zhang, "An automated refactoring tool for dataflow visual programming language," *ACM SIGPLAN Notices*, vol. 43, no. 4, pp. 21–28, 2008.

[20] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, "Graphical programming environments for educational robots: Open roberta - yet another one?" in *2014 IEEE Int. Symp. on Multimedia*, 2014, pp. 381–386.

[21] Q. M. Tran, B. Wilmes, and C. Dziobek, "Refactoring of simulink diagrams via composition of transformation steps," in *ICSEA 2013*, 2013.

[22] "Eclipse 4diac - the open source environment for distributed industrial automation and control systems," 2021. [Online]. Available: https://www.eclipse.org/4diac/

[23] L. Sonnleithner, S. Bácsi, M. Oberlehner, E. Kutsia, and A. Zoitl, "Do you smell it too? towards IEC 61499 bad smells," 2021, Manuscript submitted for publication.

[24] S. Patil, D. Drozdov, and V. Vyatkin, "Adapting software design patterns to develop reusable iec 61499 function block applications," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE, 18.07.2018 - 20.07.2018, pp. 725–732.

[25] T. Gerlitz, Q. M. Tran, and C. Dziobek, "Detection and handling of model smells for matlab/simulink models," in *MASE@MoDELS*, 2015.

[26] V. Dubinin, V. Vyatkin, and H.-M. Hanisch, "Modelling and verification of iec 61499 applications using prolog," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*, 2006, pp. 774–781.