2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) | 978-1-6654-6152-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SEAA56994.2022.00061

Bad Smells in Industrial Automation: Sniffing out Feature Envy

Lisa Sonnleithner, Rick Rabiser, Alois Zoitl, *Member, IEEE*
CDL VaSiCS, *LIT CPS Lab, Johannes Kepler University Linz*
Linz, Austria
{lisa.sonnleithner, rick.rabiser, alois.zoitl}@jku.at

***Abstract*—Bad Smells are sub-optimal software structures or patterns. They can obstruct the understandability of a software system and cause maintenance issues. As a result, it is critical to avoid Bad Smells. While the subject is well-researched in software engineering, it remains an unresolved issue in industrial automation, e.g., when developing control software in the context of a Cyber-Physical Production System (CPPS). In this short paper, we present possible detection methods for Feature Envy, a smell that indicates bad modularization of a software system. We explain how these methods can be applied to analyze control software developed in IEC 61499. We present first results as well as next steps.**

***Index Terms*—IEC 61499, distributed control system, CPS, CPPS, design smell, code smell, bad smell, anti-pattern, Feature Envy, metric, measure**

I. Introduction

So-called Bad Smells [1] are certain sub-optimal structures or patterns in software. They negatively impact software quality and can lead to poor understandability, maintenance issues, or poor expandability of a software system [2], [3]. Therefore, Bad Smells should be avoided.

The topic is already well-researched in software engineering in general. The tools of many programming languages are already capable of automatically detecting various smells. However, the domain of industrial automation is lagging behind. In our previous work [4], we took a first step towards closing that gap and defined a catalog of Bad Smells in IEC 61499 [5], a domain-specific modeling language that is used for developing distributed control software in CPPSs. Some of the identified smells can be automatically detected, e.g., by defining and checking rules. We implemented automatic detection for such smells in 4diac IDE, an IEC 61499 development tool that is part of the open-source project Eclipse 4diac [6]. Now, we focus on the detection of more complex smells. Particularly, in this short paper, we investigate how Feature Envy can be automatically detected. It "stinks" of Feature Envy when a software component is more interested in another component's parts than in its own [7]. Therefore, this smell indicates flaws in the modularization and design of a software system. Other research on Feature Envy in software engineering [8], [9] already provides an extensive overview of the topic. We will use this existing knowledge, adapt it, and bring it to the domain of CPPS.

We will first take a closer look at how Feature Envy is detected in other domains and whether research on Feature Envy exists in the domain of industrial automation in Section II. Then we describe possible scenarios where it could smell of Feature Envy in IEC 61499 in Section III. In Sections IV and V we apply two possible detection methods to IEC 61499 before we present first results of applying the detection methods in Section VI. We conclude the paper in Section VII.

II. Related Work

Fowler et al. [7] first identified the Bad Smell Feature Envy. As already mentioned, Feature Envy is a sign of bad modularization. Since then, the detection of this smell has received much attention from the research community. In software engineering, surveys and literature reviews on bad smells in general [8] and on smell detection techniques in particular exist [9], [10]. For Feature Envy, we can thus find many detection approaches in the literature.

Nongpong [11], [12] introduces a metric to detect Feature Envy that is based on the ratio of external and internal calls. They call this metric Feature Envy Factor. Another approach for Feature Envy detection is proposed by Simon et al. [13]. This approach is based on calculating the similarity and distance [14] between a set of entities with respect to certain properties. The goal is to identify methods of a class that are more similar to another class's methods than their own. The approach proposed by Tsantalis and Chatzigeorgiou [15] is implemented in the open-source tool JDeodorant [16] and is also based on a distance measure. Seng et al. [17] propose an approach that uses an evolutionary algorithm and simulated refactorings to improve the class structure of an object-oriented system. Liu et al. [18] present machine-learning-based Feature Envy detection. Kumar and Chhabra [19] describe a two-level technique that is based on dynamic analysis to sniff out Feature Envy. Bavota et al. [20] identify move method refactoring opportunities to remove Feature Envy based on relational topic models.

However, despite all the advances in Bad Smell detection in software engineering, Bad Smells in languages that are commonly used in industrial automation to develop control software for CPPSs (i.e., IEC 61131-3 and IEC 61499) are not researched well. Vogel-Heuser et al. [21] mention that the presence of Bad Smells negatively influences the extendability and maintainability of a software system and therefore hinders software evolution. Patil et al. [22] mention Bad Smells as the root cause of many bugs and defects and name five

978-1-6654-6152-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SEAA56994.2022.00061

mostly size-related smells. Our previous work [4], to the best of our knowledge, is the only one that provides a detailed overview of Bad Smells in IEC 61499. In this paper, we investigate the applicability of the metrics-based detection approaches proposed by Nongpong [11] and Simon et al. [13] to IEC 61499 in more detail (see Section IV and Section V).

III. IEC 61499 and Feature Envy

In this section, we will give a brief introduction to IEC 61499 and then describe scenarios of Feature Envy in IEC 61499.

A. IEC 61499

IEC 61499 is a domain-specific modeling language that is used for distributed control software in CPPSs. An IEC 61499 Application is a network of Function Blocks (FBs). There are several different kinds of FBs, e.g., a Basic Function Block (BFB) that contains a state machine or a Composite Function Block (CFB) that encapsulates a Function Block Network (FBN). Each FB has a defined interface, which can consist of several incoming and outgoing event and data connections. An incoming event triggers a certain procedure within the FB. On completion, an outgoing event may be triggered. Each event can be associated with data connections. If a data connection is associated with an event, the data gets updated when the event is triggered. Data inputs can be seen as the input parameters of an FB's method that can be called via an event. Similarly, output data associated with an output event can be seen as the return parameters of a method. However, without looking inside the FB or using behavioral models, it is impossible to know which input event will lead to which output event. An FB may have several input and output events. Therefore, an FB could be seen as a class that provides several methods in the form of events and associated data connections. For further details on IEC 61499, we refer the reader to the literature, e.g., [23].

B. Feature Envy in IEC 61499

Based on the definition of Feature Envy by Fowler et al. [7], we identified scenarios where Feature Envy can be found in IEC 61499 [4].

These scenarios are based on different encapsulation mechanisms of IEC 61499. On the one hand, there is an encapsulation of code and on the hand there is an encapsulation of FBNs. In each case, the encapsulated code or network should be highly cohesive as it represents a software module.

BFBs and Simple Function Blocks (SFBs) provide a means to encapsulate a state machine with several algorithms and a single algorithm, respectively. CFBs and typed Sub Applications (SubApps) provide an encapsulation mechanism for an FBN.

As a first step, we will focus on the detection of Feature Envy of encapsulated FBNs (i.e., Feature Envy in CFBs or typed SubApps) and will describe it in more detail. In future work we will also cover where Feature Envy could be sniffed out in other IEC 61499 FB kinds.

When looking at FBs in an FBN, on the one side, we can see event connections that determine the sequence in which FBs are called and data connections that visualize where data is coming from. In Figure 1, an FBN is shown that consists of two expanded SubApps. Event connections are shown in red and are connected to the upper part of an FB while data connections are shown in blue and are connected to the lower part. Instead of combining these different kinds of connections into one metric definition, we propose two metric definitions: one metric that measures Feature Envy on the control flow level and one metric that measures Feature Envy on the data flow level.

IV. Detection based on Feature Envy Factor

In this section, we describe the Feature Envy Factor metric that was defined by Nongpong [11] and then apply it to IEC 61499 in the form of a Control Flow Feature Envy Factor and a Data Flow Feature Envy Factor. For a given object obj and method mtd Nongpong [11] defines the Feature Envy Factor FEF as

$$FEF(obj, mtd) = w\frac{m}{n} + (1 - w)(1 - x^m) \qquad (1)$$

with the number of calls m within the method on the object, the total number of calls n in the method and the weights w and x. The first part of the equation calculates the ratio of method-to-object calls compared to all calls of the method, i.e., how interested the method is in the object's data. The second part of the equation leads to more attention on methods that are called more often than others. The weight w determines how much weight each part of the equation is given.

A. Feature Envy Factor in IEC 61499

To calculate the Feature Envy Factor in IEC 61499 for CFBs and typed SubApps, we define an object as a CFB or typed SubApps and a method as an FB that is encapsulated by an object, i.e., an FB within a CFB or typed SubApp. For the *Control Flow Feature Envy Factor*, m is the number of event connections from the method to the object and n is the total number of event connections of the method. In the same way, the *Data Flow Feature Envy Factor* is defined for data connections.

B. Sniffing out Feature Envy

To identify potential Feature Envy, the Control Flow and Data Flow Feature Envy Factor of all FBs within a CFB or SubApp and to connected CFBs or SubApps has to be calculated. The results can be visualized in a matrix or a heat map. Values should be high for FBs within their own CFB or SubApp and low to other CFBs or SubApps. When one FB shows significantly higher values for both factors to another CFB or SubApp, it strongly indicates Feature Envy.

V. Detection based on Distance

Similarly to the Control Flow Feature Envy Factor and the Data Flow Feature Envy Factor, we also propose to calculate a distance measure that is based on the control flow and one

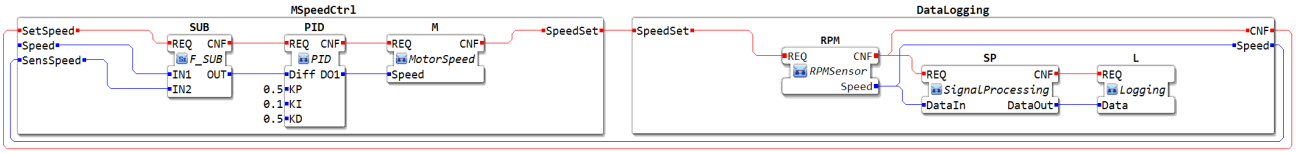

Fig. 1: Example of a motor speed control and a datalogging SubApp.

that is based on the data flow. Both metrics are only applicable to pairs of FBs within an FBN. We first describe the similarity and distance measures that were defined by Simon et al. [14] and used for Feature Envy detection [13]. We then apply this concept to IEC 61499.

Simon et al. [14] define the degree of similarity $sim(x, y)$ between two entities x and y relative to a finite subset B of all properties P_i as

$$sim(x, y) = \frac{|(p(x) \cap p(y)) \cap B|}{|(p(x) \cup p(y)) \cap B|} \tag{2}$$

with $p(x) = \{P_i | x \text{ possesses } Pi\}$. With the degree of similarity $s(x, y)$, the distance between two entities x and y is defined as

$$dist(x, y) = 1 - sim(x, y). \tag{3}$$

To apply this concept to IEC 61499, the definition of entities and properties is essential. As our goal is sniffing out Feature Envy on FB level, we define FBs as entities. The definition of which properties to use to detect Feature Envy in IEC 61499 is less straightforward.

A. Behavior Distance in IEC 61499

With the Behavior Distance, we want to measure the similarity or dissimilarity of FBs in relation to the control flow of the application. Therefore, the property to measure the Behavior Distance of an IEC 61499 FB is an event connection, as it defines the control flow. With this property definition and Equations 2 and 3, the behavior distance between two FBs, FB_1 and FB_2, can be calculated as follows:

$$dist_B(FB_1, FB_2) = 1 - \frac{\sum E_S}{\sum E_A}. \tag{4}$$

E_S, are the shared event connections that connect FB_1 and FB_2. E_A means all incoming or outgoing event connections of FB_1 and FB_2, regardless of their source or destination.

B. Information Distance in IEC 61499

With the Information Distance, we want to measure the similarity or dissimilarity of FBs in relation to where the data they are using is coming from. Therefore, the property to measure the Information Distance of an IEC 61499 FB is a data connection, as it defines the data flow. With this property definition and the Equations 2 and 3, the information distance between two FBs, FB_1 and FB_2, can be calculated as follows:

$$dist_I(FB_1, FB_2) = 1 - \frac{\sum D_S}{\sum D_A}. \tag{5}$$

D_S, are shared data connections that connect FB_1 and FB_2. D_A means all incoming or outgoing data connections of FB_1 and FB_2, regardless of their source or destination.

C. Sniffing out Feature Envy

To identify potential Feature Envy, the behavior and information distance between all FBs within a CFB or SubApp has to be calculated. The results can again be visualized in a matrix or a heat map. When one FB shows significantly higher values for both behavior and information distances to the other FB, it is a strong indication of Feature Envy. Additionally, the distance between this potentially problematic FB and other FBs it is connected to can be calculated. If the distance to FBs within another CFB or SubApp is lower, a refactoring operation to move the FB could be suggested.

VI. First Results

Figure 1 shows an example of two SubApps, $MSpeedCtrl$ and $Datalogging$. The task of these SubApps is to control the speed of a motor via a simple closed-loop-control and log the speed data after the raw data is processed. The FB RPM, which is reading the data of a rotational speed sensor, is misplaced and should instead belong to $MSpeedCtrl$. Also, by placing RPM in $MSpeedCtrl$, the number of interface elements of the two SubApps could be reduced from 7 to 6. As a first evaluation of our metrics, we now apply them to this small example to test whether the misplaced FB is detected.

Table Ia shows the Control and Data Flow Feature Envy Factor as proposed in Section IV. For this example, we chose a weight $w = 0$ to focus on the first part of the equation. With $w = 0$, a Feature Envy Factor above 0.5 means that more than half of the total connections of an FB are connected to the currently investigated SubApp. Therefore, FBs should have a value of 0.5 or above in their own SubApp and a value of 0.5 or below to other SubApps. In this example, the Control Flow Feature Envy Factor of the FB RPM and the Data Flow Feature Envy Factor of the FB SUB within their own SubApp are 0.33, which indicates a problem. Additionally, RPM shows a higher Control Flow Feature Envy Factor to $MSpeedCtrl$. Therefore, RPM is successfully identified as a misplaced FB.

Table Ib shows the Behavior Distance Matrix and Table Ic shows the Information Distance Matrix of the example based on the metrics proposed in Section V. As $dist(x, y) = dist(y, x)$, the matrix is symmetrical.

In the Behavior Distance Matrix, we can see that the FB RPM is connected to two FBs of $MSpeedCtrl$, while it is only connected to one other FB of its current SubApp. The Information Distance Matrix does not give a clear indication about a possibly misplaced FB. As the Behavior Distance Matrix suggests that RPM should better be placed in $MSpeedCtrl$ and the Information Distance Matrix does not provide contradicting information, also in this case RPM is successfully identified as a misplaced FB. This becomes

TABLE I: Results of the proposed Feature Envy metrics applied to the example.

(a) Feature Envy Factor of example

	MSpeedCtrl		Datalogging	
	CF	DF	CF	DF
SUB	0.50	0.33	0.50	0.33
PID	1	1	0	0
M	0.50	1	0.50	0
RPM	0.67	0.50	0.33	0.50
SP	0	0	1	1
L	0	0	1	1

(b) Behavior Distance Matrix

	SUB	PID	M	RPM	SP	L
SUB	X	0.67	1	0.75	1	1
PID	0.67	X	0.67	1	1	1
M	1	0.67	X	0.75	1	1
RPM	0.75	1	0.75	X	0.75	1
SP	1	1	1	0.75	X	0.50
L	1	1	1	1	0.50	X

(c) Information Distance matrix

	SUB	PID	M	RPM	SP	L
SUB	X	0.75	1	0.75	1	1
PID	0.75	X	0.50	1	1	1
M	1	0.50	X	1	1	1
RPM	0.75	1	1	X	0.67	1
SP	1	1	1	0.67	X	0.50
L	1	1	1	1	0.50	X

particularly clear when we compare the average distance values of FBs within their own SubApp before and after moving RPM to $MSpeedCtrl$. The average Behavior Distance for $MSpeedCtrl$ slightly increases from 0.78 to 0.81 and the average Information Distance also slightly increases from 0.75 to 0.83 when RPM is added. The average Behavior Distance for $Datalogging$ decreases from 0.75 to 0.5 and the average Information Distance also decreases from 0.72 to 0.5 when RPM is removed.

VII. Conclusion and Next Steps

In this work, we identified methods that are used in other domains to sniff out Feature Envy and applied them to the domain of industrial automation, i.e., IEC 61999-based control software in CPPSs. We proposed two sets of metrics: (i) the Control Flow and Data Flow Feature Envy Factor based on the Feature Envy Factor from Nongpong [11] and (ii) the Behavior and Information Distance based on Simon et al. [14] for the identification of Feature Envy on FB level in IEC 61499. We applied these metrics to a small example and evaluated the results. Both types of metrics successfully sniffed out Feature Envy in the example. In particular, the metrics seem promising as they can not only identify Feature Envy, but also provide information on where the problematic FB could be moved to.

This would allow us to combine the detection of Feature Envy with refactoring proposals, such as the one by Oberlehner et al. [24], to automatically resolve the detected problem. As a next step, we will implement both sets of metrics in the open-source tool 4diac IDE. We will then apply the metrics to real-world examples, provided by industry partners, to evaluate them further. Additionally, we will investigate (i) other detection approaches and (ii) other scenarios where it could smell of Feature Envy in IEC 61499.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged.

References

[1] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[2] M. Abbes, F. Khomh, Y.-G. Guéhéneuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in *2011 15th European Conf. on Software Maintenance and Reengineering*, 2011, pp. 181–190.

[3] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" in *2012 28th IEEE Int. Conf. on Software Maintenance*, 2012, pp. 306–315.

[4] L. Sonnleithner, M. Oberlehner, E. Kutsia, A. Zoitl, and S. Bácsi, "Do you smell it too? towards bad smells in iec 61499 applications," in *2021 26th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2021, pp. 1–4.

[5] IEC TC65/WG6, "IEC 61499-1, function blocks - part 1: architecture: Edition 2.0," Geneva, 2012. [Online]. Available: www.iec.ch

[6] Eclipse 4diac, "Eclipse 4diac - the open source environment for distributed industrial automation and control systems," 2019. [Online]. Available: https://www.eclipse.org/4diac

[7] M. Fowler, K. Beck, J. Brant, and W. Opdyke, "Refactoring: Improving the design of existing code," 1999.

[8] T. Sharma and D. Spinellis, "A survey on software smells," *J. of Systems and Software*, vol. 138, pp. 158–173, 2018.

[9] G. Rasool and Z. Arshad, "A review of code smell mining techniques," *J. of Software: Evolution and Process*, vol. 27, no. 11, pp. 867–895, 2015.

[10] A. AbuHassan, M. Alshayeb, and L. Ghouti, "Software smell detection techniques: A systematic literature review," *J. of Software: Evolution and Process*, vol. 33, no. 3, p. e2320, 2021.

[11] K. Nongpong, "Feature envy factor: A metric for automatic feature envy detection," in *2015 7th Int. Conf. on Knowledge and Smart Technology*, 2015, pp. 7–12.

[12] ——, "Integrating" code smells" detection with refactoring tool support," Ph.D. dissertation, The University of Wisconsin-Milwaukee, 2012.

[13] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," in *Proceedings Fifth European Conf. on Software Maintenance and Reengineering*, 2001, pp. 30–38.

[14] F. Simon, S. Löffler, and C. Lewerentz, "Distance based cohesion measuring," 08 1999.

[15] N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities," *IEEE Trans. on Soft. Eng.*, vol. 35, no. 3, pp. 347–367, 2009.

[16] M. Fokaefs, N. Tsantalis, and A. Chatzigeorgiou, "Jdeodorant: Identification and removal of feature envy bad smells," in *2007 IEEE Int. Conf. on Software Maintenance*, 2007, pp. 519–520.

[17] O. Seng, J. Stammel, and D. Burkhart, "Search-based determination of refactorings for improving the class structure of object-oriented systems," in *Proc. of the 8th Annual Conf. on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2006, p. 1909–1916.

[18] H. Liu, Z. Xu, and Y. Zou, *Deep Learning Based Feature Envy Detection*. New York, NY, USA: ACM, 2018, p. 385–396.

[19] S. Kumar and J. K. Chhabra, "Two level dynamic approach for feature envy detection," in *2014 Int. Conf. on Computer and Communication Technology (ICCCT)*, 2014, pp. 41–46.

[20] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "Methodbook: Recommending move method refactorings via relational topic models," *IEEE Trans. on Soft. Eng.*, vol. 40, no. 7, pp. 671–694, 2014.

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

[22] S. Patil, D. Drozdov, G. Zhabelova, and V. Vyatkin, "Refactoring of IEC 61499 function block application — A case study," in *IEEE Industrial Cyber-Physical Systems*. IEEE, 2018, pp. 726–733.

[23] A. Zoitl and R. W. Lewis, *Modelling control systems using IEC 61499*, 2nd ed., ser. IET Control engineering series. London: IET, 2014, vol. 95.

[24] M. Oberlehner, L. Sonnleithner, B. Wiesmayr, and A. Zoitl, "Catalog of refactoring operations for iec 61499," in *2021 26th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2021, pp. 01–04.