SMOKE: Simulink Model Obfuscator Keeping Structure

Alexander Boll
University of Bern
Bern, Switzerland
Gesellschaft für Informatik
Berlin, Germany

Timo Kehrer
University of Bern
Bern, Switzerland

Michael Goedicke
University of Duisburg-Essen
Essen, Germany

Abstract

Simulink is extensively used across various industries to model and simulate cyber-physical systems. Most industry-built models contain sensitive intellectual property, which prevents companies from sharing models with interested third parties, such as researchers. Initiatives to replace industry-built models with open-source alternatives exist, however they offer only limited remedy. In this work, we offer a novel approach: a Simulink obfuscation tool named SMOKE, designed to selectively protect intellectual property in models. This allows companies to share relevant parts of their models with researchers or other third parties, while safeguarding all sensitive information. We evaluated the tool on an extensive set of open-source models and found it successfully removes sensitive components, while preserving model structure. A video demonstration of SMOKE is available online at https://youtu.be/TFeFNKHSlAw.

CCS Concepts

• **Software and its engineering** → **Software maintenance tools**; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Computing methodologies** → **Model development and analysis**; • **Security and privacy** → **Software and application security**.

Keywords

Simulink, Obfuscation, Sanitization, Protection, Intellectual Property, FAIR Principles, Tool

ACM Reference Format:
Alexander Boll, Timo Kehrer, and Michael Goedicke. 2024. SMOKE: Simulink Model Obfuscator Keeping Structure. In *Proceedings of ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems – Tools and Demonstrations Track (MoDELS Demos).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Across various industries, Simulink is a widely-used tool to design, implement and simulate cyber-physical systems [4, 5]. The popularity of Simulink also gives rise to a considerable research interest, e.g., into better understanding Simulink models and their evolution [22]. However, within the research community, it is well known that companies mostly do not share their Simulink models to protect intellectual property (IP) [7]. In some cases, industry partners share their models under severe limitations, such as non-disclosure agreements, which strictly prohibit the publication of model files or any visual representations of the models. Often, companies control the composition of research groups, or restrict access to models to company computers and locations. This practice creates significant challenges for the FAIR principles [28] in Simulink research and often leaves researchers without useful study subjects [22]. The modeling research community has begun addressing this issue by developing corpora of open-source Simulink models [5, 22–24], which can serve as substitutes for proprietary models in research. However, open-source models in general are much smaller than industry models, and are often no adequate substitutes [5, 7].

In this work, we introduce SMOKE: **S**imulink **M**odel **O**bfuscator **K**eeping structur**E**, an extendable, open-source tool for protecting intellectual property through selective anonymization of Simulink models. Our overall goal is to enable anonymization by ensuring the removal of sensitive information while maintaining the models' usefulness for research. Thus, a model being anonymized using SMOKE is still a valid Simulink model whose logical structure is isomorphic to the original one, yet exposing a different visual appearance and functional behavior, depending on the desired degree of anonymization. Both kinds of modification are implemented through model transformations. The former class of transformations is based on layout obfuscations as partially realized in an early predecessor version of SMOKE[1], while the latter class adopts the idea of sanitization (i.e., removal of sensitive data or functionality) as originally presented in the context of relational databases [17]. The concrete transformations to be applied can be selected by the user, depending on the desired degree of anonymization. SMOKE supports both interactive and non-interactive selections of transformations, which are then composed to an executable transformation workflow.

SMOKE addresses two use cases which are of primary interest for researchers. First, it simplifies obtaining approval for publishing visual representations of Simulink models by selectively removing layout information. Second, companies may permit further study or use of sanitized models where sensitive data or functionality is removed. As opposed to traditional obfuscators concealing a program's [10] or model's [27] entire functionality within an uninspectable and immutable virtual black box [3], our white-box anonymization yields a native Simulink model open to further inspection. This means that the model can be opened with the standard MATLAB/Simulink editor or other tools working with Simulink models. A particular feature of SMOKE is that it preserves the logical structure of the original models. Even if highly

[1] https://github.com/McSCert/Obfuscate-Model

MoDELS Demos, September 22–27, 2024, Linz, Austria

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

anonymized, the obtained 'structure-only' models still remain valuable study subjects for many research interests. To better understand various aspects of a model or its evolution, empirical research on Simulink models often focuses on metrics related to model structure [1, 5, 9, 24], or relies on third-party analysis tools such as clone detectors, differencing tools, slicers, or variant and information flow analyzers that require only an intact model structure to function [15, 19–21, 26].

We give an overview of SMOKE's anonymization capabilities, and showcase the effect of interactively applying a subset of those on a realistic example. Moreover, we report about an experiment applying SMOKE's full anonymization capabilities on thousands of open-source models taken from SLNET [24], with a particular focus on evaluating its general applicability and functional correctness. While SMOKE focuses on Simulink models, we argue that other modeling ecosystems such as the UML/SysML family could also benefit from a similar tool by adopting its basic ideas [16].

Our tool SMOKE, written in MATLAB, along with its documentation, and all artifacts from our experimental evaluation are open-source and available at https://github.com/lanpirot/SMOKE. A brief video demonstration of SMOKE is available online at https://youtu.be/TFeFNKHSlAw.

2 Background

2.1 Simulink

Simulink [13] is a versatile modeling, implementation, and simulation tool and IDE. Simulink is integrated with and based on MATLAB, and offers toolboxes for various industry domains. Models created in Simulink are block diagrams, where Blocks are connected by Signal Lines. Blocks perform computations, transforming their inputs to outputs, while the output is transported by a Signal Line to other Blocks' inputs. Special Subsystem Blocks are employed to hierarchically structure Simulink models, allowing for the nesting of Blocks and Signal Lines. Subsystems enable *views* of a model, where only the currently selected Subsystem and its direct children are visible. Two views of a Simulink model are shown in Figure 1, representing the contents of Subsystems named 'Heat Sources' and 'Kelvin to Celsius'. These Subsystems are components of the model shown in Figure 3a, where their internal contents are hidden and they appear as opaque Subsystem Blocks.

While Simulink comes with an extensive set of different kinds of Blocks, users can design new Blocks or configure a multitude of Block parameters to customize Block behavior. The variety of Blocks and their parameters offer a wide range of design options for static and dynamic modeling.

2.2 Obfuscation and Sanitization

Obfuscations are transformations that increase the difficulty of understanding a program's (i.e. model's) purpose or logic [8]. This is achieved by changing some aspect of a program while preserving the functionality of the original program. Collberg [10] classifies obfuscations into three basic categories: layout obfuscation, data obfuscation, and control obfuscation. Layout obfuscations adapt documentation, naming, or formatting, usually by removing them or resetting them to default placeholders. Data obfuscation alters the storage, encoding, aggregation, or ordering of data. Control obfuscation changes the order or branching of the control flow.

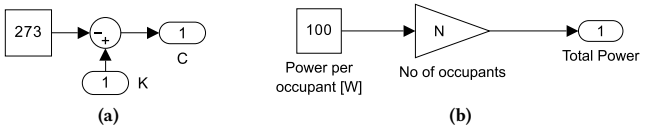


Figure 1: Two Simulink Subsystems of the model in Figure 3a: the Subsystem in Figure 1a transforms temperature from Kelvin to Celsius: $C = K - 273$, the Subsystem in Figure 1b computes the total heat generation of N occupants: $TotalPower = 100 \cdot N$.

Data sanitization [2, 17] is the protection of sensitive data in relational databases. This protection is enforced by selectively removing parts of the database, while valuable insights into the rest of the data remain possible. Data sanitization is performed, so that the protected database can be shared with others, without risking the sensitive data being leaked.

3 Tool Overview and Capabilities

3.1 Design Rationale

Inspired by the concept of data sanitization (cf. Section 2.2), a simple yet effective approach for structure-preserving sanitization of models is to remove or reset structurally irrelevant yet functionally critical data, such as:

- Block Parameters: Values and settings determining Block behavior.
- Constant Values: Fixed numerical values or other data.
- Block Callback Functions: Functions triggered by events.
- MATLAB scripts in Function Blocks: Complex MATLAB scripts may be part of these special blocks.
- Functionality of Stateflow charts: These charts hold representations of logic-based state machines and control systems.

To achieve obfuscation, we adopt a layout obfuscation approach (cf. Section 2.2), which is inherently structure-preserving. This method reduces the readability of the model by altering its visual layout, but it does not affect the model's functionality.

Users can choose from multiple options for both sanitization and obfuscation (cf. Section 3.3), which they can apply to selectively anonymize the model.

3.2 Transformation Workflow

Once the user initiates the anonymization, SMOKE executes the user-selected transformations sequentially. Each transformation iterates over all model elements, removing their names, or other properties, resetting parameters to default values, etc. For some transformations, SMOKE chooses a partial ordering. For example, library links should be resolved (i.e. library Blocks currently not directly included in model) before other transformations are applied, as changes on linked Blocks are not permitted. In a softer ordering rule SMOKE is resizing and reshaping Blocks in the penultimate

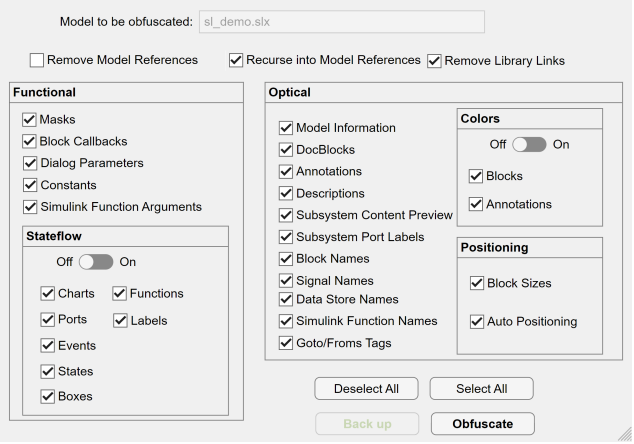


Figure 2: The GUI of SMOKE, displaying the various anonymization options.

step, to allow possible text within them to fit. Blocks are repositioned in the final step, to minimize overlap and create visually pleasing diagrams.

However, users can choose their own order of transformation execution by performing step-wise anonymization, activating only a single checkbox each time.

3.3 GUI Overview and User Interaction

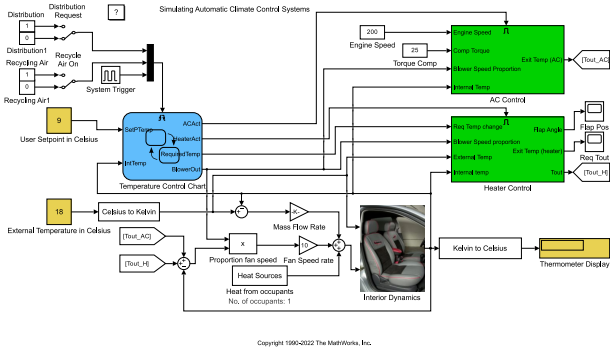
After starting SMOKE, its GUI presents the user with a variety of anonymization options that can be activated or deactivated, see Figure 2, and stating which model will be anonymized in the top line. Users can choose from *Functional* sanitization (left panel), which purposefully removes or breaks data or functionality of the model, and *Optical* obfuscation (right panel), which hampers the comprehensibility of the model. Furthermore, users may choose to anonymize imported models (above the panels), i.e. Model References or linked Blocks. Users can interactively anonymize a model, checking only a few boxes at first, then inspect the resulting model, and then either revert or do another anonymization step with other anonymizations.

Alternatively, users may choose to use SMOKE in batch mode, to anonymize a whole set of models.

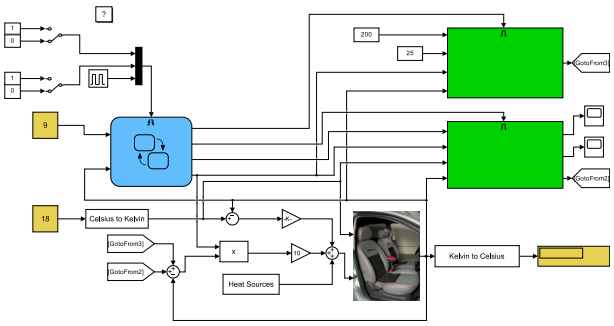
If a user used all anonymizations, only an 'empty shell' remains, while all data, functionality, and layout is gone. Such a shell can still offer value for structural analysis, for publishing screenshots of a model view, or be used as a base for a model with different content. Depending on the particular use-case, a partial sanitization may suffice, while leaving key aspects of the model intact.

4 SMOKE in Action

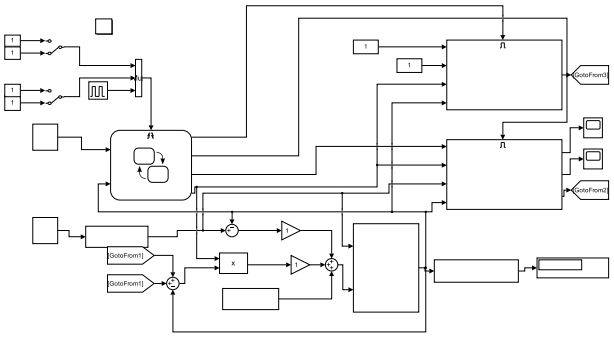
To get an understanding of SMOKE, we show a series of example obfuscation steps in Figure 3. All shown obfuscation steps are also taking place within the nested Subsystems, like the ones in Figure 1 or even deeper layers, but are not visible in the outside views of Figure 3. The sanitization steps are also invisible. Note that the user can decide which aspects of the model they want obfuscated and in which order.



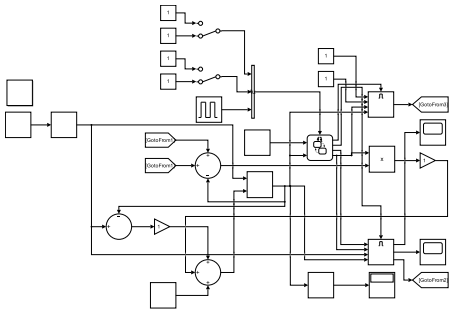
(a) A Simulink model before anonymization.



(b) Block names and Annotations are removed.



(c) Coloring and Masks are removed.



(d) Block position, sizes and shapes are reset.

Figure 3: SMOKE obfuscates an exemplary model further and further. Depending on the user-chosen anonymizations, additional invisible changes may also occur.

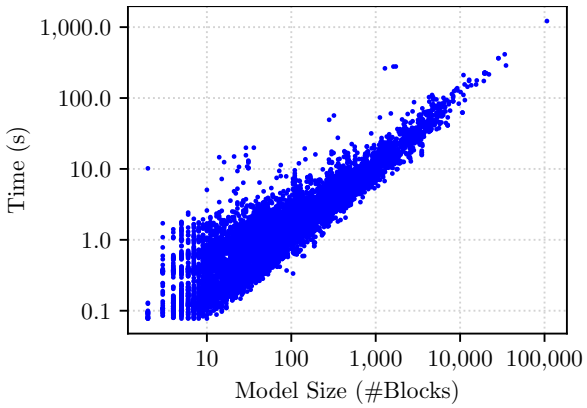


Figure 4: Log-log scatter plot of model size vs anonymization time for SMOKE execution on the SLNET model set.

(1) The original model in Figure 3a depicts various Block types, and a richness in layout like coloring and relevant names.
(2) In a first step, SMOKE removes Annotations (top and bottom text), resets names of Blocks, and Inputs/Outputs with placeholders and hides them, cf. Figure 3b.
(3) Next, coloring and Block Masks are removed in Figure 3c.
(4) Lastly in Figure 3d, Block sizes and shapes are reset, and Blocks and Signal Lines are repositioned. The diagram is now completely scrambled, considering Figure 3a.

5 Evaluation

In this section, we demonstrate SMOKE's applicability and validate its functionality. To evaluate SMOKE in a large-scale experimental setting, we used SLNET [24], a set of 9,105 open-source Simulink models from 2,837 repositories from GitHub[2] and the Mathworks FileExchange.[3] SLNET was previously used as a benchmark set in other empirical studies [6, 25] and represents a highly diverse spectrum of Simulink models, presenting numerous challenging corner cases for SMOKE. We evaluated SMOKE on a Windows laptop with an i9-13980HX processor and MATLAB R2024a.

To demonstrate SMOKE's applicability, we used it to anonymize the entire SLNET set within a reasonable timeframe. To this end, SMOKE anonymized every model of the SLNET set, with all anonymization options activated, as shown in Figure 2. SMOKE skipped broken models that are unloadable, and models that are locked: of the 9,105 models 9,040 were loadable, and 7,916 of these were unlocked and thus anonymized. SMOKE successfully anonymized the entire set in about 15 hours, processing about 100 blocks per second on average (see Figure 4 for a scatter plot of anonymization times for each model). Though there might be even larger industrial models than the open-source models comprised by SLNET, this demonstrates SMOKE's suitability for anonymizing entire industry projects.

To validate SMOKE's correctness, we checked the structural integrity of the anonymized models, and whether their functionality is removed. We verified the equality of block numbers in the original and anonymized models as a proxy for structural integrity. We used the block number metric, because it is commonly used in empirical research [5, 9, 24] and computationally inexpensive, considering the large dataset. SMOKE preserved the number of blocks for all models. To gauge functionality removal, we checked if models were compilable before and after anonymization, as shown in Table 1. More than half of the compilable models were no longer compilable after the anonymization. Interestingly, a few models became compilable (without true functionality), likely because inconsistencies, along with all other functionality of the original models, were removed. We did not perform model simulations due to the unknown settings for model input, output, and other simulation parameters. Nevertheless, the anonymization already has a significant impact on compilation.

	before compilable	before not compilable
after compilable	1060	253
after not compilable	1582	5021

Table 1: A cross-table of the ability to compile SLNET models, before vs. after anonymization.

6 Related Work

While obfuscation in traditional text-based programming languages is a well-established discipline with decades of research [8, 10], we found only limited prior work on the obfuscation of Simulink models. Most notably, Tevajärvi [27] recently conducted a literature review of existing model obfuscation techniques and then compared them in terms of a case study. Amongst others, they successfully used functional mock-up interfaces and obfuscating generated code to preserve functionality while hiding sensitive data or functionality. However, the resulting black boxes are then in a different, uninspectable format, rather than remaining as (Simulink) models.

Simulink comes with the Protected Model Creator,[4] which is a versatile tool that again transforms Simulink models into black boxes of another file format, or white boxes that are not editable. However, these immutable white boxes do not anonymize at all. Other built-in options to create black box versions are S-Functions, or static libraries.[5]

A subset of SMOKE's layout obfuscations could be found in a tool by Ohashi[6] and the Obfuscate-Model tool by Jaskolka et al.[7]. The former, however, is unmaintained and broken since at least 2017, according to the tool's reviews on MATLAB FileExchange. The latter is more mature and has been utilized for obfuscation in an industry-research partnership [12, 18]. Though it supports only layout obfuscations, we built SMOKE based on a fork of the Obfuscate-Model tool, extending it with new capabilities (all functional sanitization introduced in Section 3 as well as additional layout obfuscation for resizing and repositioning of Blocks, and a anonymization reset) refined prior capabilities (SMOKE recurses

[2] https://github.com
[3] https://www.mathworks.com/matlabcentral/fileexchange
[4] https://www.mathworks.com/help/rtw/ref/protectedmodelcreator.html
[5] https://www.mathworks.com/matlabcentral/answers/91537-how-do-i-protect-the-ip-of-my-simulink-model-when-sharing-it-with-others-who-may-include-it-in-thei
[6] https://www.mathworks.com/matlabcentral/fileexchange/54359-model-obfuscation-tool
[7] https://github.com/McSCert/Obfuscate-Model

into all the deep parts of models for its transformations: into descendants of masked, read-protected, or linked Blocks, and variants of Blocks), and fixed bugs (e.g., crashes on broken array indices or uncaught exceptions for some transformations). Moreover, we evaluated SMOKE's functionality on a large model set.

7 Conclusion and Future Work

With SMOKE, we offer a versatile tool to selectively protect IP from Simulink models, while their structure is preserved. Our hope is that SMOKE will facilitate the collaboration of researchers and industry partners, as it enables the selective protection of models and their sharing with the research community for many research areas.

We are currently integrating SMOKE as a filter step into a workflow of creating research data management containers[11, 14]. With SMOKE, we ensure sensitive model parts are excluded, before they become part of an immutable container. In the future, we hope that Simulink will integrate features of SMOKE natively to make model anonymization even more accessible.

8 Acknowledgements

This work was partially supported by the NFDIxCS project, funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project number 501930651.

References

[1] Tiago Amorim, Alexander Boll, Ferry Bachman, Timo Kehrer, Andreas Vogelsang, and Hartmut Pohlheim. 2023. Simulink bus usage in practice: an empirical study. *Journal of Object Technology* 22, 2 (July 2023), 2:1–14.

[2] Mike Atallah, Elisa Bertino, Ahmed Elmagarmid, Mohamed Ibrahim, and Vassilios Verykios. 1999. Disclosure limitation of sensitive rules. In *Proc. 1999 Workshop on Knowledge and Data Engineering Exchange (KDEX'99)*. IEEE, 45–52.

[3] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. 2001. On the (Im)possibility of Obfuscating Programs. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference (Lecture Notes in Computer Science, Vol. 2139)*, Joe Kilian (Ed.). Springer, 1–18.

[4] Vincent Bertram, Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe, and Michael von Wenckstern. 2017. Component and Connector Views in Practice: An Experience Report. In *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA, September 17-22, 2017*. IEEE Computer Society, 167–177.

[5] Alexander Boll, Florian Brokhausen, Tiago Amorim, Timo Kehrer, and Andreas Vogelsang. 2021. Characteristics, potentials, and limitations of open-source Simulink projects for empirical research. *Software and Systems Modeling* 20, 6 (2021), 2111–2130.

[6] Alexander Boll, Pooja Rani, Alexander Schultheiß, and Timo Kehrer. 2024. Beyond code: Is there a difference between comments in visual and textual languages? *Journal of Systems and Software* 215 (2024), 112087.

[7] Alexander Boll, Nicole Vieregg, and Timo Kehrer. 2022. Replicability of experimental tool evaluations in model-based software and systems engineering with MATLAB/Simulink. *Innovations in Systems and Software Engineering* (2022), 1–16.

[8] Mariano Ceccato, Massimiliano Di Penta, Paolo Falcarin, Filippo Ricca, Marco Torchiano, and Paolo Tonella. 2014. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering* 19 (2014), 1040–1074.

[9] Shafiul Azam Chowdhury, Lina Sera Varghese, Soumik Mohian, Taylor T. Johnson, and Christoph Csallner. 2018. A curated corpus of simulink models for model-based empirical studies. In *Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems, ICSE 2018, Gothenburg, Sweden, May 27, 2018*, Tomás Bures, John S. Fitzgerald, Bradley R. Schmerl, and Danny Weyns (Eds.). ACM, 45–48.

[10] Christian Collberg, Clark Thomborson, and Douglas Low. 1997. *A taxonomy of obfuscating transformations*. Technical Report. Department of Computer Science, The University of Auckland, New Zealand.

[11] Michael Goedicke and Ulrike Lucke. 2022. Research Data Management in Computer Science - NFDIxCS Approach. In *52. Jahrestagung der Gesellschaft für Informatik, INFORMATIK 2022, Informatik in den Naturwissenschaften, 26. - 30. September 2022, Hamburg (LNI, Vol. P-326)*, Daniel Demmler, Daniel Krupka, and Hannes Federrath (Eds.). Gesellschaft für Informatik, Bonn, 1317–1328.

[12] Monika Jaskolka, Vera Pantelic, Alan Wassyng, and Mark Lawford. 2020. Supporting Modularity in Simulink Models. arXiv:2007.10120 [cs.SE]

[13] Harold Klee and Randal Allen. 2018. *Simulation of dynamic systems with MATLAB® and Simulink®*. Crc Press.

[14] Firas Al Laban, Jan Bernoth, Michael Goedicke, Ulrike Lucke, Michael Striewe, Philipp Wieder, and Ramin Yahyapour. 2023. Establishing the Research Data Management Container in NFDIxCS. In *1st Conference on Research Data Infrastructure - Connecting Communities, CoRDI 2023, Karlsruhe, Germany, September 12-14, 2023*, York Sure-Vetter and Carole A. Goble (Eds.). TIB Open Publishing.

[15] Marcus Mikulcak, Paula Herber, Thomas Göthel, and Sabine Glesner. 2019. Information Flow Analysis of Combined Simulink/Stateflow Models. *Inf. Technol. Control.* 48, 2 (2019), 299–315.

[16] Peter Munk and Arne Nordmann. 2020. Model-based safety assessment with SysML and component fault trees: application and lessons learned. *Software and Systems Modeling* 19, 4 (2020), 889–910.

[17] Stanley R. M. Oliveira and Osmar R. Zaïane. 2003. Protecting Sensitive Knowledge By Data Sanitization. In *Proc. 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*. IEEE Computer Society, 613–616.

[18] Vera Pantelic, Steven Postma, Mark Lawford, Monika Jaskolka, Bennett Mackenzie, Alexandre Korobkine, Marc Bender, Jeff Ong, Gordon Marks, and Alan Wassyng. 2018. Software engineering practices and Simulink: bridging the gap. *International Journal on Software Tools for Technology Transfer* 20 (2018), 95–117.

[19] Robert Reicherdt and Sabine Glesner. 2012. Slicing MATLAB Simulink models. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, Martin Glinz, Gail C. Murphy, and Mauro Pezzè (Eds.). IEEE Computer Society, 551–561.

[20] Alexander Schlie, Sandro Schulze, and Ina Schaefer. 2018. Comparing Multiple MATLAB/Simulink Models Using Static Connectivity Matrix Analysis. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 160–171.

[21] Alexander Schlie, David Wille, Sandro Schulze, Loek Cleophas, and Ina Schaefer. 2017. Detecting Variability in MATLAB/Simulink Models: An Industry-Inspired Technique and its Evaluation. In *Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume A, Sevilla, Spain, September 25-29, 2017*, Myra B. Cohen, Mathieu Acher, Lidia Fuentes, Daniel Schall, Jan Bosch, Rafael Capilla, Ebrahim Bagheri, Yingfei Xiong, Javier Troya, Antonio Ruiz Cortés, and David Benavides (Eds.). ACM, 215–224.

[22] Sohil Lal Shrestha, Alexander Boll, Shafiul Azam Chowdhury, Timo Kehrer, and Christoph Csallner. 2023. EvoSL: A Large Open-Source Corpus of Changes in Simulink Models & Projects. In *26th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2023, Västerås, Sweden, October 1-6, 2023*. IEEE, 273–284.

[23] Sohil Lal Shrestha, Alexander Boll, Timo Kehrer, and Christoph Csallner. 2023. ScoutSL: An Open-Source Simulink Search Engine. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2023 Companion, Västerås, Sweden, October 1-6, 2023*. IEEE, 70–74.

[24] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2022. SLNET: A Redistributable Corpus of 3rd-party Simulink Models. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*. ACM, 1–5.

[25] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2023. Replicability Study: Corpora For Understanding Simulink Models & Projects. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2023, New Orleans, LA, USA, October 26-27, 2023*. IEEE, 1–12.

[26] Matthew Stephan and James R. Cordy. 2015. Identification of Simulink model antipattern instances using model clone detection. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, Timothy Lethbridge, Jordi Cabot, and Alexander Egyed (Eds.). IEEE Computer Society, 276–285.

[27] Juho Tevajärvi. 2023. *Protecting Intellectual Property in Multi-Supplier Ship Powertrain Co-Simulation*. Master's Thesis. Aalto University, Otaniemi. Advisor(s) Riku Ala-Laurinaho.

[28] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3, 1 (2016), 1–9.