SMOKE2.0 Whitebox Anonymizing Intellectual Property in Models While Preserving Structure

Alexander Boll, Manuel Ohrndorf and Timo Kehrer

Software Engineering Group, University of Bern, Switzerland.

*Corresponding author(s). E-mail(s): alexander.boll@unibe.ch;
Contributing authors: manuel.ohrndorf@unibe.ch;
timo.kehrer@unibe.ch;

Abstract

Simulink is widely 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 or collaborating companies. However, advancing model-based engineering research requires access to such models – either to derive empirical insights or to evaluate new tools. While initiatives to replace industry-built models with open-source alternatives exist, they offer only a limited remedy. In this work, we introduce a novel approach with SMOKE, a Simulink obfuscation tool designed to selectively protect intellectual property within models. This allows companies to share relevant parts of their models with researchers or other third parties while safeguarding all sensitive information. SMOKE's white-box design preserves the model's original format and structure, ensuring that meaningful insights remain accessible. 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 youtu.be/0i42BzgJAUA.

Keywords: Data Masking, Sanitization, Obfuscation, Anonymization, Abstraction, Filtering, Open Science

1 Introduction

Across various industries, Simulink is a widely-used tool to design, implement and simulate cyber-physical systems [1, 2]. The popularity of Simulink also gives rise to

a considerable research interest, e.g., into better understanding Simulink models and their evolution [3]. However, within the research community, it is well known that companies often refrain from sharing their Simulink models to protect intellectual property (IP) [4]. 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. Companies may also restrict access to models to company computers and locations. This practice creates significant challenges for the FAIR[1] principles [5] in Simulink research and often leaves researchers without useful study subjects [3]. The modeling research community has begun addressing this issue by developing corpora of open-source Simulink models [1, 3, 6, 7], which can serve as substitutes for proprietary models in research. In general, however, open-source models are much smaller than industry models, and are often no adequate substitutes [1, 4].

In this work, we introduce **SMOKE**: **S**imulink **M**odel **O**bfuscator **K**eeping struc-tur**E**, an extensible, open-source tool for protecting intellectual property through selective anonymization of Simulink models. Our goal is to anonymize models by removing sensitive information while preserving their general usefulness, especially 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 unidirectional model transformations [8], i.e., they are non-reversible without logging or domain knowledge. The first class of modifications are layout obfuscations, which preserve functionality but reduce understandability. The second class comprises sanitization techniques, which remove sensitive data or functionality. This is a concept we adapt from database sanitization, originally developed for relational databases [9]. 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. A precursor to SMOKE was already previously used to obfuscate industry models, enabling the publication of screenshots in scientific articles [10, 11]. 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 [12] or model's [13] entire functionality within an uninspectable and immutable virtual black box [14], 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 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

[1] Findability, Accessibility, Interoperability, and Reuse of digital assets.

Simulink models often focuses on metrics related to model structure [1, 6, 15, 16], or relies on third-party analysis tools such as clone detectors, differencing tools, slicers, or variant and information flow analyzers that require an intact model structure to function [17–21].

In addition to addressing researchers' interest, SMOKE can also be employed by industry partners themselves. In software value chains, models are developed in co-engineering fashion, and such collaborations only work if the parties have access to the models [22]. However, companies may still hesitate to grant full access to others or may need to comply with various competition laws [23]. Moreover, (partially) anonymized models can be indexed by model search engines, as they can search models by basic metric structure [7, 24, 25]. This way, interested parties may find models according to basic information, and can get into contact with the owners to agree on the terms of full access.

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 our evaluation applying SMOKE's full anonymization capabilities on thousands of open-source models taken from SLNET [6], 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 UML, SysML, Modelica, etc., could also benefit from a layout or functionality anonymizer [26].

Our tool SMOKE, written in MATLAB, along with its documentation, and all artifacts from our experimental evaluation, are open-source and available at github.com/lanpirot/SMOKE. A brief video demonstration of SMOKE is available online at youtu.be/0i42BzgJAUA.

This paper is a substantial extension of the short paper "SMOKE: Simulink Model Obfuscator Keeping Structure" by Boll et al. [27], demonstrated at the MODELS'24 demo track. We have built upon our prior work through the following improvements:

- We updated SMOKE, extending it with new anonymization functionality. Notably, it is now applicable to library models, and anonymizations can be performed on user-selected scopes of the model.
- We repeated our entire evaluation, incorporating the aforementioned new functionality and now including library models. We greatly improved the evaluation of the behavior (non-)preservation of SMOKE.
- We further added discussions on SMOKE's coverage and threats to validity.
- Leveraging the expanded space available compared to the original short paper, we have meticulously rewritten and significantly enhanced all sections, notably Sections 2 to 5, from scratch. This revision includes extensive additional detail and thorough explanations that were not feasible to incorporate previously.

2 Background on Simulink and Obfuscation Techniques

In this section, we first lay the foundational background knowledge of Simulink that is needed for this paper. We then elaborate on obfuscation and sanitization techniques, and how they can be applied in Simulink. Note that we do not discuss a completely

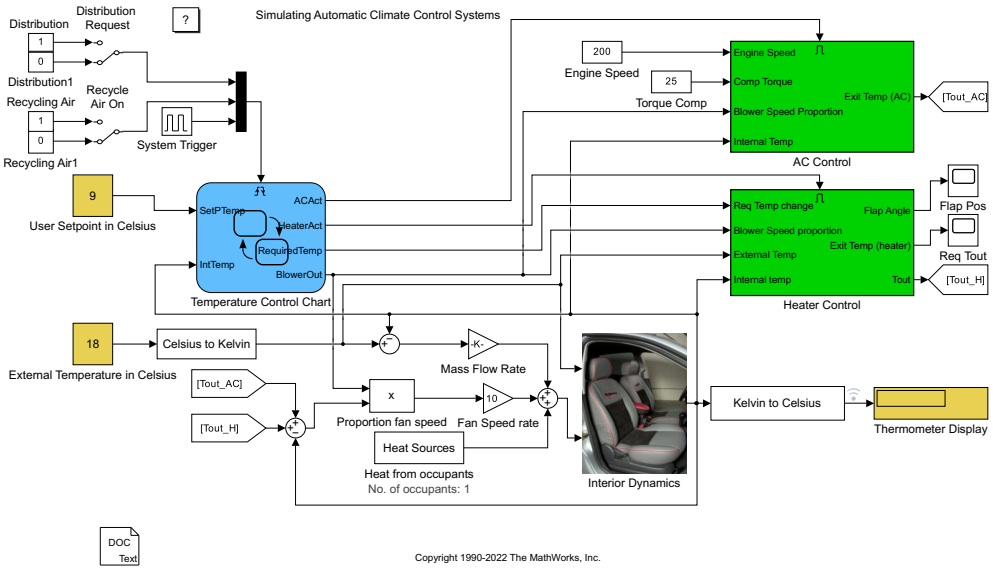


Figure 1: An exemplary Simulink model (with the name sldemo_auto_climate) showing various kinds of Blocks, connected by Lines.

exhaustive list of obfuscation techniques, but focus on those that are relevant to our context, i.e., the ones that are applicable to Simulink models, while keeping structural integrity.

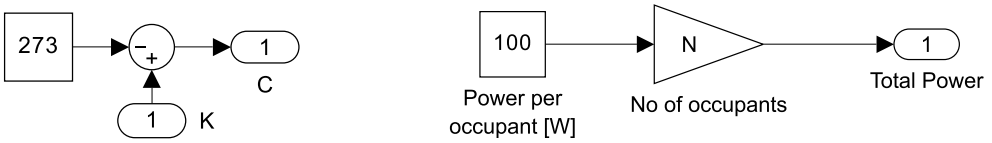
2.1 Simulink

Simulink [28] is a modeling language and versatile integrated development environment developed by Mathworks. It is based on and integrated into the MATLAB IDE and can be used for abstract modeling, implementation of functionality, simulation, and code generation. As its graphical modeling language is intuitive to use and understand, it is often used as a low-code development platform [29] in non-programmer domains like automotive, aerospace, medical and other technical industry domains [2, 30, 31].

A Simulink model consists of *Blocks*, which can be connected by *Lines* – both placed on a modeling canvas, cf. Figure 1.[2] Lines transport values from Block outport(s) to Block inport(s). Blocks transform their input from their inport(s) to an output which they emit via their outport(s). As a model grows, it can be partitioned using special *Subsystem* Blocks that nest Blocks and Signals, allowing developers to hierarchically structure their models. Subsystem Blocks can also nest other Subsystem Blocks and with this, developers can hierarchically structure their models.

The Simulink IDE offers different model *views* via the Subsystem Blocks. In a view, only the direct content of the currently selected Subsystem is visible, while Subsystems hide their nested implementation details from the outer view. While Figure 1 depicts

[2] Example Simulink model from mathworks.com/help/simulink/slref/simulating-automatic-climate-control-systems.html



(a) (b)

Figure 2: The inner view of two Simulink Subsystems of the model in Figure 1: the Subsystem in Figure 2a transforms temperature from Kelvin to Celsius: $C = K - 273$, the Subsystem in Figure 2b computes the total heat generation of N occupants: $TotalPower = 100 \cdot N$.

the root view of a model, the Subsystems' inner views of the Subsystems 'Kelvin to Celsius' (lower right of Figure 1) and 'Heat Sources' (bottom middle of Figure 1) is given in other views, shown in Figure 2. The root Subsystem, i.e., the model itself, may also have inports and outports, which act as the input and output of the model. In many models, inports are driven by physical sensors and outputs drive actuators. Additionally, there are also models without input or output, e.g., library models that contain useful sets of custom Blocks, or functionality used in other models where they can be imported and reused. While Simulink already offers an extensive set of different Block types, users can use this mechanism to define their own (complex) Block behaviors. In addition, each Simulink Block comes with its own set of *Parameters*[3] that can be adjusted individually for each instance of the Block, cf. Figure 7a. This variety of Block types and their individual Parameters offer a comprehensive range of design options supporting both static modeling (memory-less systems), and dynamic modeling (time-evolving systems with states).

In this work, we define the model's *structure* as a typed graph of Blocks connected by Lines (cf. Figure 3). When we speak of *structure-preserving* anonymizations, or transformations, we are thus looking for obfuscations and sanitizations that do not alter the underlying graph of Blocks and Lines.

2.2 Obfuscation Techniques

Obfuscations are transformations that change aspects of a program, while preserving its functional behavior. Obfuscating a program – in our case 'a program' is to be understood as 'a model' – increases the difficulty of understanding, or accessing its purpose or logic [33]. Using obfuscations, one can thus protect the intellectual property of a program and make it harder to reuse or repurpose it.

Collberg et al. [12] identify three basic classes of obfuscation: (1) *layout obfuscation*, (2) *data obfuscation*, and (3) *control obfuscation*. (1) Layout obfuscations are simple transformations that adapt documentation, names, or formatting, usually by removing them, resetting them to default values, or replacing them by arbitrary values. As

[3] In EMF models, the counterpart of Parameters are called Attributes [32]. The Parameters of a Block define how it exactly behaves during simulation, and are not external output/input like parameters in textual programming languages.

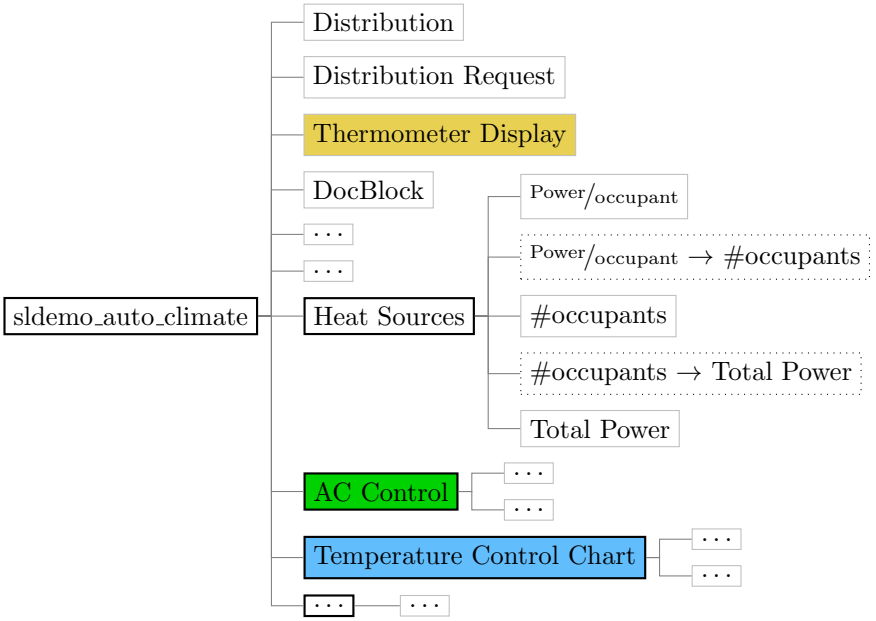


Figure 3: The partial structure of the model of Figure 1. All structure elements are part of this tree, hierarchically organized by the Subsystems. The root Subsystem is on the very left. Each Subsystem's child is a Block or Line within it. Only the content of the Subsystem 'Heat Sources' (cf. Figure 2b) is fully shown, all other content is only hinted at.

Simulink is a graphical language, it offers many more layout obfuscations than classical textual languages, e.g., Blocks and Lines can be repositioned, resized, or recolored, fonts can be changed, and media elements can be used. (2) Data obfuscations alter the storage, encoding, or ordering of data. A different encoding, or even encryption, makes it difficult to interpret the data that is being processed in the program. Additional data abstraction transformations can be splitting up related data, or bunching up data that is unrelated. Simulink offers multiple data types, like `int8, double, boolean`,[4] with which one could obfuscate data encoding. Additionally, transformations on data abstraction are possible by using BUS-Blocks, that bundle up data from multiple Lines into one. (3) Control obfuscation manipulates the abstraction layers, adds useless conditions, or dead code. Obfuscating the abstraction layers is similar to the obfuscation of data abstraction: the removal of meaningful hierarchy layers or the addition of extraneous ones. In Simulink this can be achieved, e.g., by resolving Subsystems or adding new arbitrary ones into the model.

[4] It is further possible to create new, custom data types.

2.3 Sanitization Techniques

While obfuscation does not change the behavior of a program, *sanitization* does. In this work, we generalize the concept of *data sanitization* [9, 34] to our purpose of *model sanitization*: data sanitization is applied on sensitive data in relational databases by selectively removing parts of the database, while valuable insights into the rest of the data still remain possible. Data sanitization is performed so that the protected database can be shared with others, without risking the sensitive data being leaked.

In our case, we want to preserve the structure of a model, so that valuable insights into the model are preserved, while the model behavior may be removed, changed or even completely broken. This means, the model should still be syntactically correct after all sanitization transformations, and still be loadable and inspectable in the IDE. On the other hand, the sanitized model should not show the same behavior, which means it either does not compile anymore, the simulation crashes or stops prematurely, or, given the same input, the output of the model changes.

As the structure of the model shall remain stable, the only way to alter the model functionality is to change Block or model Parameters. Such changes either result in altered behavior of Block calculations and thus overall model behavior, or model misconfigurations that lead to non-compilability or simulation crashes.

3 Related Work

3.1 Simulink-specific Obfuscation

While obfuscation in traditional text-based programming languages is a well-established discipline with decades of research [12, 33], we found only limited prior work on the obfuscation of Simulink models.

Simulink itself features the Protected Model Creator,[5] a versatile tool that 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 so-called S-Functions or static libraries.[6]

A subset of SMOKE's layout obfuscations could be found in a tool by Ohashi[7] and the Obfuscate-Model tool by Jaskolka et al.[8] 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 [10, 11]. Though it supports only layout obfuscations, we built SMOKE based on a fork of the Obfuscate-Model tool, extending it with new capabilities, refined prior capabilities, and fixed bugs. For a detailed list of all changes and improvements, see Section 4.2.1. Moreover, we evaluated SMOKE's functionality on a large model set.

[5] https://www.mathworks.com/help/rtw/ref/protectedmodelcreator.html
[6] 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
[7] https://www.mathworks.com/matlabcentral/fileexchange/54359-model-obfuscation-tool
[8] https://github.com/McSCert/Obfuscate-Model

3.2 Obfuscating Other Model Types

Obfuscation approaches are also suggested in various other domains, such as conceptual models, business process models, or ML models. An overview is given by Tevajärvi [13] who recently conducted a literature review of existing model obfuscation techniques and then compared them in terms of a case study. Among 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, opaque format.

Fill [35] discusses obfuscating conceptual models with the intent of making them shareable, but his transformation breaks the structure by splitting models into multiple parts. Other transformations he suggested break the model's syntax. Nacer et al. [36] also explore business process model obfuscation by fragmenting models.

A further concept of model sharing in a co-engineering scenario with multiple roles (see Martinez et al. [37]) is explored in the work of Weber et al. [38]. In a first step, the model is split into sub-modules and their interfaces. Each sub-module is encrypted and only decryptable by authorized parties – effectively fragmenting the model. In our tool, selective aspects of the model (e.g., whole sub-modules or only sensitive properties within the model or some sub-modules) are removed.

Sihler et al. [22] build an obfuscation tool for IRIS, a graphical modeling tool for technology road maps (TRM). The tool can perform various one-way transformations with the intent of preserving model behavior, in addition to the model still being in the same file format. During their transformations, the model structure becomes completely removed, though. Their transformations provably uphold criteria, such as self-containment of the model parts that are left; no inference possible on the obfuscated model parts; and preserving behavior. This makes their tool similar to SMOKE's use case of obfuscation, while preserving behavior. However, SMOKE also serves the use case of selectively sanitizing behavior and preserving model structure.

Less related are the ideas of Gupta et al. [39], who protects CAD models by inserting sabotaging elements into the model that hamper reproducing physical instances of the model. One may interpret this as the model structure largely being preserved, while its behavior is changed.

Finally, the *ModelObfuscator* by Zhou et al. [40] is intended for Deep Learning models and uses various obfuscation techniques, some of them structure-preserving. However, many of them intentionally alter the model's structure, such as extra layer injection.

4 Approach and Tool

4.1 Approach

The main goal of SMOKE is to support users with the selective removal of sensitive intellectual property from their models via obfuscation, or sanitization, while keeping the model structurally intact and loadable. We thus create one-way and resilient [41] (aka. unidirectional [8]) obfuscation and sanitization transformations in Simulink that fulfill these criteria for SMOKE.

To come up with a list of possible transformations, we first went over the metamodel approximation of Simulink from the Massif group (referenced in [32]) and identified model elements, that could be altered to obfuscate or sanitize. In a second step, we consulted the Simulink Documentation for model elements that are not listed in this metamodel (e.g., 'Model Information'). In a last step (see Section 5.5), we inspected the raw model files for possibly sensitive user information in elements, that were still present in models after we applied transformations.

As Simulink is a vast ecosystem,[9] we can't offer an obfuscation or sanitization option for every model element. Many model elements interact with each other in complex ways, which is impossible to deal with in a research prototype. Instead, we chose to offer options for those transformations that we deem the most intuitive ones: transformations on elements which are used often, are prominently displayed to users, or are used to hold sensitive or model-wide information. In Section 4.2 we will see though, that SMOKE can be easily extended to obfuscate or sanitize other model elements as users may seem fit.

Our extensive list of selectable elements to be obfuscated and/or sanitized can be seen in the main menu of SMOKE in Figure 4: in the lower right block 'Optical' is the list of obfuscations, in the lower left block 'Functional' is the list of sanitizations. In the following, we will shortly elaborate on a selection of these shown transformations, and our rationale behind them.

4.1.1 Obfuscations

In Simulink, a model's primary Parameters, such as `CreatorName, CreationDate, and ModifiedBy`, are stored in its 'Model Information' Parameters. SMOKE can remove these Parameters, i.e., reset them to default values, and thus disrupt the model's traceability. Next, Simulink offers various ways of commenting and documenting within models, such as 'DocBlocks,' 'Annotations,' and 'Descriptions'. Note that these documentation options are directly embedded into the model file, while additional model 'Notes' exist [42]. By removing such documentation SMOKE can impair the model's understandability. Additionally, Simulink offers a feature called 'Subsystem Content Preview,' which allows users to preview the contents of a Subsystem, such as the blue Subsystem shown on the left of Figure 1. Disabling this preview ensures that no external information is visible in the current model view, meaning screenshots will only display information from the currently opened Subsystem. Furthermore, all other obfuscation options shown in Figure 4 either remove custom names or custom design elements from the model, both of which negatively impact the model's understandability.

4.1.2 Sanitizations

In Simulink 'Masks' are used to hide and protect the internals of a Subsystem. This can be a simple picture icon visually (e.g., the car seat picture on the Subsystem in the

[9] In addition to the hundreds of blocks Simulink already ships with, various toolboxes offer additional Blocks. Each Block type has its own specific Parameters in addition to the common ones. We observed more than 100 Block types and more than 10,000 Parameters. Finally, Simulink comes with its own sub-languages: StateFlow, and SimScape. This results in a multitude of corner cases, that need to be dealt with, if one strives for an exhaustive obfuscator.

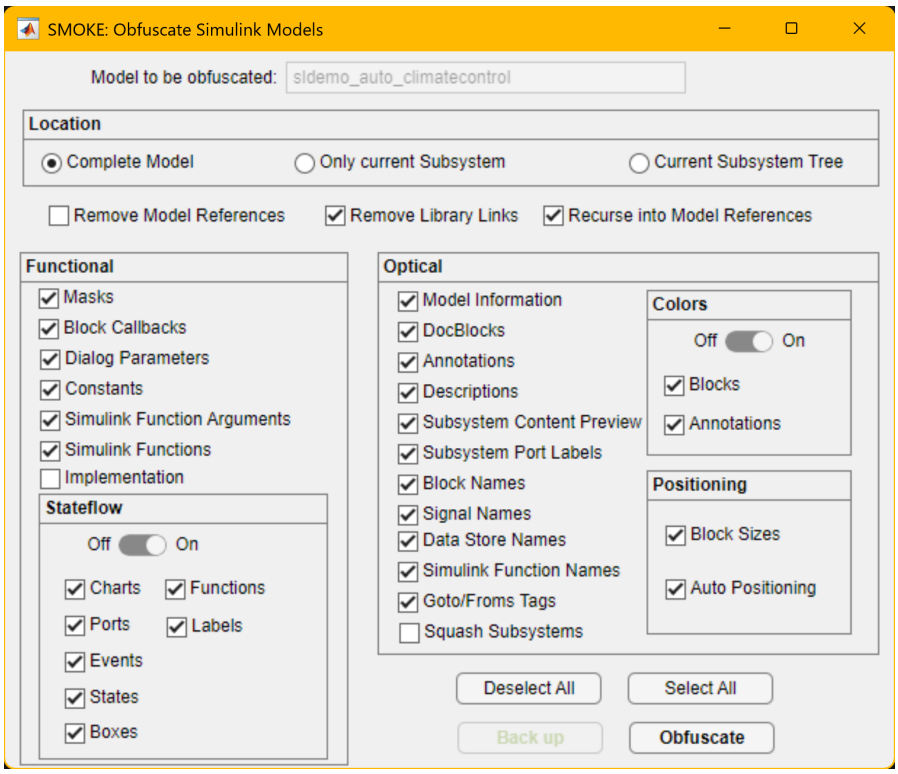


Figure 4: The main menu of SMOKE. The menu's elements are described in Section 4.3.1.

lower bottom of Figure 1), but can also offer complex custom Parameters. SMOKE removes such Masks, and the bare Subsystems remain under them. 'Block Callbacks' are custom MATLAB scripts that are executed on certain events, like loading the model or changing a view. They are powerful and can, e.g., check for the presence of certain model elements, and depending on the result of the check, prevent the saving of the model. Apart from a few undeletable Callbacks, SMOKE can remove all others. 'Dialog Parameters' are probably the most important sanitizer, as all kinds of Block functionality can be changed here for each Block individually (cf. Figure 7). SMOKE will reset Parameters to Block default values, whenever possible. 'Constants' are special Blocks, that drive a constant input into the model. Constant Blocks' values are rest to either default number or string values, depending on their type. More interestingly, users can embed a whole MATLAB script into a Function Block, which are sanitized, i.e., their body is removed by activating 'Simulink Functions'. We also address various parts of the Simulink sub-language 'Stateflow' which offers to embed state machines or stateflows into Blocks, with more fine-grained options.

4.1.3 Structural Transformations

The options 'Squash Subsystems' and 'Implementation' (the only unchecked options in the bottom of the panels 'Functional' and 'Optical' in Figure 4) do in fact change the model structure. We still incorporated them into SMOKE, as they were recommended by users. 'Squash Subsystems' removes all Subsystems, i.e., they get resolved and a completely flattened model hierarchy emerges. When checking 'Implementation' all Blocks and Lines apart from Inports and Outports of the current Subsystem get removed. This option is useful when certain parts of the model can be shared, while other parts are so sensitive that even their structure needs to be removed.

4.1.4 Reversing Transformations

In general, obfuscation and sanitization transformations should be hard or impossible to reverse; otherwise, they need not to be applied. We thus implement (see Section 4.2) all our transformations to actually remove or reset, and not to just hide model information. In particular, we remove model elements, whenever possible. Elements, whose removal would result in a changed structure, or broken model, are reset to default values. As such custom information is removed, it can only be reconstructed using additional domain knowledge or by extrapolating other model information that was not selected to be removed by the user.

Additionally, we made sure, that model information is actually permanently removed, once a transformation was applied by the user. There appear to be only two possibilities for reversing any deletion: either the deleted information, while not visible to the user in the Simulink IDE, may still be part of the raw model file (1), or the Simulink IDE may offer an "undo action" functionality to restore the information (2).
Regarding (1): we compared pairs of models before and after transformations, manually and using scripts, to make sure the deleted information is indeed removed from the raw model file. We found our transformations to completely remove the selected information from the raw file. As the model information is saved into and recreated from this file, this path of transformation reversal is thus impossible.
Regarding (2): Simulink does offer to undo IDE actions from its current session. Once a model is saved and closed in the IDE, the edit history dissipates and is not recoverable from the saved file.[10] Furthermore, the transformations SMOKE calls are not reversible in the first place, as they are not IDE edits, but edits executed by scripted MATLAB transformations, as shown in Section 4.2. MATLAB can currently only undo IDE edits, though.
In conclusion: we are positive, that sharing a SMOKE-transformed file will remove all the user-selected information permanently and irreversibly.

4.1.5 SMOKEing other Modeling Languages

Most ideas we present in this paper, are similarly applicable to obfuscate or sanitize models created in other modeling languages. One simply has to choose what kind of information shall be preserved in the model (in our case, the model structure and ability to parse it with the IDE), and what types of information can or should be

[10] This applies to MATLAB versions through R2025a and is expected to remain valid in future versions.

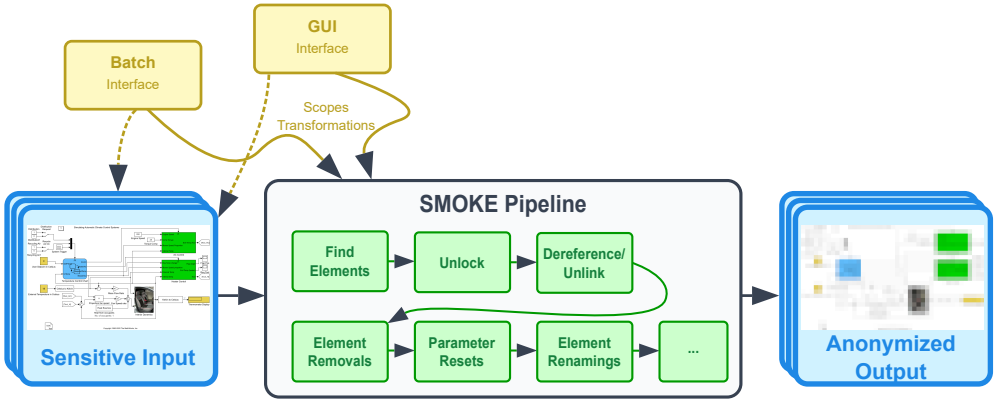


Figure 5: SMOKE's architecture: its pipeline of model transformations can be invoked from the GUI or in batch mode.

removed. For modeling languages with a known metamodel and fewer Block types and Parametrical corner cases than Simulink, it may also be possible to automatically derive obfuscation/sanitization transformations directly from the metamodel. Mathworks, however, never published Simulink's metamodel, and only the unofficial approximation by the Massif group [32] exists.

4.2 Implementation and Extensibility

We implemented SMOKE[11] in the MATLAB scripting language as a standalone application. Every transformation uses the Simulink model API of MATLAB, to alter the model via functions such as `delete(<modelElement>)`, or `set_parameter(<blockID>,<Parameter>,<newValue>)`. We did not directly alter the model's raw XML files, as this risks breaking the model, i.e., making it impossible to load or edit the model.

After starting the SMOKE app, the user chooses a Simulink model as input, and then selects transformations (cf. Section 4.3.1) to be applied on the model. SMOKE transforms a model through a pipeline architecture, cf. Figure 5. If the user chooses multiple transformations, they will be applied sequentially in a pipeline architecture. Before any transformation takes place, the model and all its Blocks are unlocked so they are editable in the further process. Next, the model references are resolved and library links are removed, as edits to them are not allowed by Simulink, otherwise. Finally, all selected transformations are applied sequentially.

Each transformation iterates over all the elements it pertains to and transforms them one by one, e.g., the DocBlock removal iterates over all DocBlocks. To ensure that SMOKE preserves the model structure, every transformation that is called removes exactly one model element or resets exactly one Parameter at a time. We only ever remove Annotations and DocBlocks (they are not structurally relevant) and check whether a Parameter change affects the model structure before attempting to change

[11] Available at https://github.com/lanpirot/SMOKE.

it. We thus guarantee structure-preservation by construction. For most transformations, our implementation is trivial, with just a few necessary interventions to keep the model intact, e.g., names of Blocks have to be unique in each view, or references to names have to be updated appropriately. For the resetting of names, we choose a trivial naming scheme, e.g., every Block will get the name `<block type of Block><Number>`.

One of the more interesting transformations is our sub-script `removeDialogParameters.m`. Here, all Blocks' Parameters with custom values shall be reset. Simulink does not offer a direct functionality for such a reset, and also does not give default values to which to reset them to. We thus create and insert a dummy Block of the same type into the model, and read out, and copy its values for Parameters. Sometimes dummy Blocks do not have all Dialog Parameters of the actual Block (c.f. also Figure 7, where the Parameter list changes after transformation). In this case, we reset Parameters that are numeric to `0` and string type to the empty word `''`. Further, we preserve some Parameters of Blocks, that would alter the model structure: we found, e.g., `7` Parameters that change the number of ports of a Block. Resetting them could remove the incoming or outgoing Signal Lines of the Block and thus break the model structure. After the Block's Parameters are reset, we clean up the model by removing the temporary dummy Block again. To reset a Block's size, we use the same technique and copy a dummy Block's default dimensions.

Our transformations' implementation is *optimistic* [43], i.e., our transformations may try to change or remove model elements that could break the model. Breaking transformations are caught by Simulink's error detection though and we simply skip the transformation for elements causing such errors, thus leaving the model syntactically correct. Some examples that get caught are: certain Blocks are not resizable; some model elements cannot be removed as Callback Functions are dependent on them, etc.

As each transformation takes and leaves a structurally intact model, most transformation orderings are interchangeable. For best performance, SMOKE still performs according to a partial ordering: It is best to start with transformations that remove model elements, like Blocks. This ensures that no transformation is performed on elements that are later removed, anyway. The removal of sizes and positions should be performed last. Resetting the shapes and sizes of Blocks, in the penultimate step, ensures that potential text can still be completely displayed. The removal of positions should be the last transformation: here, all Blocks and their sizes are taken into account to achieve visually pleasing diagrams without any overlap of model elements.

For debugging purposes, we found it best to move renaming transformations to the back, as it is much easier to quickly match the original model and its transformed counterpart this way. For an easier comparison of the model pair, it's also best to deselect size and position removal, as then all elements 'stay in their place' and can be matched visually at a glance.

To extend SMOKE with new transformations, users can either: (1) add a transformation to their suitable transformation list within 'element removals', 'parameter resets', and 'element renamings' (the lower middle boxes in Figure 5); (2) append it in the '...' box. The sole requirement for any new transformation is that it must preserve

syntactic correctness – that is, it should accept a valid Simulink model as input and produce a valid Simulink model as output, consistent with all prior transformations.

4.2.1 Comparison of SMOKE with Prior Versions

Our application is based on a predecessor app called `Obfuscate Model`[12] by Jaskolka et al. Here, we first compare `Obfuscate Model` to SMOKE in the state of the publication of [27], i.e., SMOKE1.0. Next, we compare SMOKE1.0 to SMOKE in this paper's state, i.e., SMOKE2.0.

Jaskolka et al. built a Simulink model obfuscation tool written in MATLAB. Besides some bug fixes (e.g., crashes on broken array indices), adding exception handling, and a slight redesign of the UI, in SMOKE1.0 we:

- added various functional transformations. `Obfuscate Model` was restricted to graphical obfuscations only.
- added the graphical obfuscations for resetting block position and block sizes.
- enabled deep obfuscations, where SMOKE1.0 also descends into masked, protected, variant or linked Blocks. These were not transformed before.
- added a "back up" option to quickly reset the transformation process and recover the original model.

Since [27], we fixed some further bugs and sped up SMOKE2.0's execution, as well as:

- The user can choose the location or scope where to apply transformations. This way, some parts of the model can be kept in their original state or be transformed differently than others.
- SMOKE2.0 can be applied to locked (library) models, which makes it applicable to all types of Simulink models, now.
- We added the most radical transformation: removing the complete implementation of a Subsystem. The whole local Subsystem tree is thus pruned.
- We added an obfuscation that removes the Subsystem hierarchy, and thus "flattens" the model, as all model elements are put into the same hierarchy level.
- We added a sanitization that removes Simulink Function bodies.

4.3 SMOKE in Action

4.3.1 Menu and User Interaction

The menu screen of SMOKE is presented to the user right at startup. At startup, the boxes are pre-checked as shown in Figure 4. In the upper line, the model slated for transformation is shown. For this, SMOKE automatically chooses the last model that is opened by the user. All chosen transformations will later be performed on this model. Next, in the second line, the user can select the scope of all chosen transformations. Either the complete model gets transformed, only the currently selected Subsystem and its direct elements, or a whole Subsystem Tree, i.e., the current Subsystem and its descendants. After that, users choose how to handle model references and library links.

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

In the lower left box, all options that remove model functionality, i.e., sanitizations, are listed. In the lower right box, all options pertaining to obfuscations are listed, which leave model functionality intact. On the bottom right are the action buttons. Two buttons for convenience that check all or none of the boxes. The 'Obfuscate' button triggers the obfuscations and sanitizations the user chose. The 'Back up' button reverts the model to its original, i.e., last saved state.

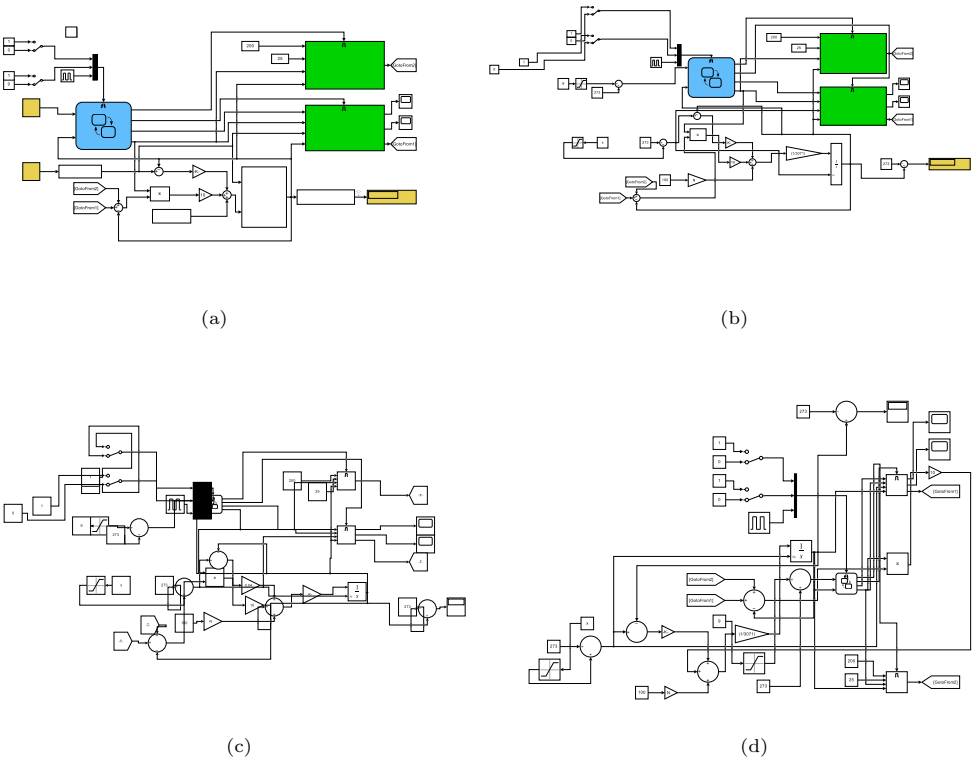
We envision two main user journeys using the GUI: (1) apply all pre-selected transformations as shown in Figure 4, which removes everything but the main model structure, (2) start with few transformations and selectively apply more and more of them, perhaps in different scopes, until all sensitive information is removed. When the user is satisfied with the model's state, they can save it, and it is ready to be shared.

In addition to SMOKE's interactive mode, it can also be called via MATLAB's scripting API. In this way, a whole project's models can be anonymized in batch mode, using pre-selected transformations. We used this mode in our evaluation in Section 5 to handle thousands of models at a time.

4.3.2 Exemplary Obfuscation

To give an intuitive overview of the capabilities and impact of SMOKE, we give an exemplary user journey of a model's obfuscation (Figure 6) and sanitization (Figure 7) of the model from Figure 1. In a first step, a user chooses to remove all documentation elements, labels and names, which leaves a completely anonymous model in Figure 6a. The model's layout otherwise is completely unchanged, and a visual mapping from the original is trivial. Next, the user decides to further flatten the model's hierarchy and all Subsystems in the model get resolved. This affects some Subsystems on the left and lower side of the model, which get replaced by their inner Blocks in Figure 6b. To further obfuscate the model, the user removes the Blocks' colors and resets their sizes in Figure 6c. In a final step, the user obtains a clean model layout by using SMOKE's autopositioning feature, with which one can cycle through semi-random layout arrangements. The final version is now completely obfuscated, while preserving the model structure and functionality. The original model from Figure 1 is hardly recognizable in its final form in Figure 6c.

If the user decides to (perhaps additionally to the obfuscation) sanitize the model, they can choose to apply various transformations affecting functionality (cf. lower left options in Figure 4). Most of these effects are not immediately visible in the IDE view, in contrast to the obvious obfuscation transformations. However, a Block's Parameters and their values can be inspected in popup menus like the one given in Figure 7a. If the user decides to reset the 'Dialog Parameters' some of the Block's Parameters are reset to their default values. For this Block's reset, the conditional Parameter 'Sample time' is revealed in Figure 7b. Changing a Block's Parameters can have a dramatic functional impact, as can be seen for the Pulse Generator Block from Figure 7, whose behavior changed from the one shown in Figure 8a to the one in Figure 8b, after the Parameter reset shown in Figure 7.



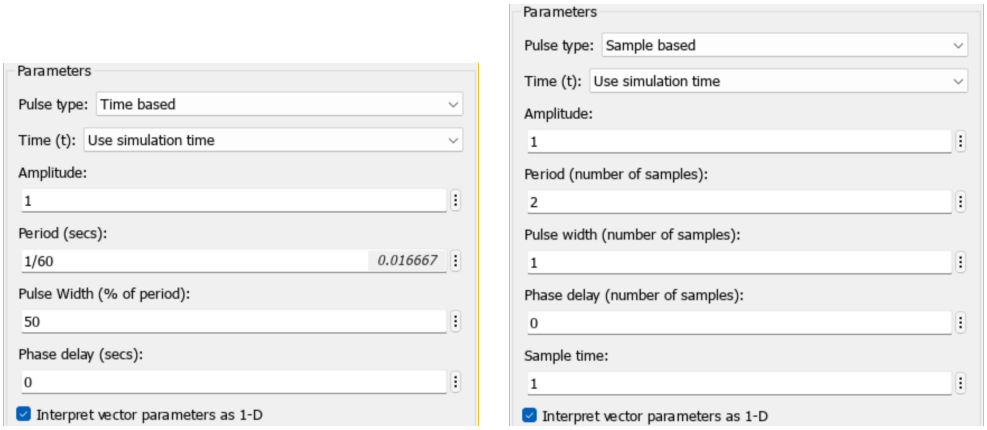
(a) (b)

(c) (d)

Figure 6: Step-wise obfuscation of the model from Figure 1. First, Annotations, Docblocks, labels and names are removed in Figure 6a. The Subsystem hierarchy is flattened in Figure 6b. Then, the Block colors and sizes are reset in Figure 6c. Finally, in Figure 6d the Block and Line positions are reset.

5 Evaluation

To ensure that SMOKE works as intended and designed, we test its behavior in multiple ways: we test, whether SMOKE's obfuscation preserves structural integrity for all its transformations in Section 5.2, whether SMOKE's obfuscation does not alter a model's behavior in Section 5.3, and whether SMOKE's sanitization *does alter* model behavior in Section 5.4. We further check SMOKE's coverage in Section 5.5. For all our experiments, we use a diverse set of models, which we briefly introduce in Section 5.1.



(a) (b)

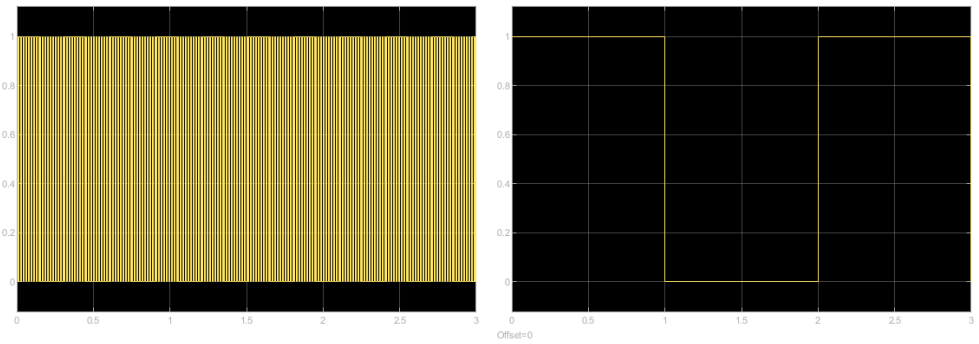
Figure 7: Figure 7a gives a snippet from the popup menu of the Parameters of the Pulse Generator Block (located in the upper left in Figure 1, called 'System Trigger'). Users can alter various Block specific Parameters in this menu. Figure 7b gives the Block's Parameters after sanitization: the Parameters 'Pulse type', 'Period', and 'Pulse Width' are affected by the resetting of Parameters. Due to the sanitization, other default Block Parameters may become accessible, like 'Sample time' in this case.

5.1 Experimental Design

5.1.1 Subjects and Setup

As described earlier, Simulink is a vast and diverse ecosystem. To ensure that the various kinds of models and use cases are covered by SMOKE, we apply SMOKE on the model collection SLNET [6]. This is a comprehensive set of 9,105 open-source models covering multiple domains like electronics, aerospace, robotics, or medicine. The collection encompasses small and big models for various purposes like toy projects, industrial application, etc. SLNET was previously used as a benchmark set in other empirical studies on Simulink models [42, 44]. 8,814 of the models were loadable error-free in our setup of Simulink with MATLAB R2024b on our laptop with Windows11, 96GB RAM, and Intel i9-13980HX processor. Models that were not loadable were not necessarily broken, but typically had some libraries missing that were needed in callbacks of Blocks. Five more models were excluded from our evaluation, as they caused MATLAB crashes during model simulation or model backup – both of these functions are called in our evaluation pipeline. This left 8,809 models for our evaluation. In terms our analysis of these models presented in the remainder of this section, we found more than 160 unique Block types and more than 10,000 unique types of Block Parameters that interact in various, complex ways. Thus, SLNET presents a large set of numerous challenging corner cases for SMOKE.

In our evaluation, we applied SMOKE sequentially on each SLNET model to obtain a pair of an original model and a transformed model. We then analyzed each pair further as described in the next sections.



(a) (b)

Figure 8: The exemplary effect of resetting Dialog Parameters: the original Block (Figure 7a) produces rapid pulses (Figure 8a), while the sanitized version (Figure 7b) has a much longer cycle length (Figure 8b).

5.1.2 Measuring Robustness, Performance, and Structural Integrity

In a first step, we measured the robustness of SMOKE, i.e., how often SMOKE was applicable to our models without error. We also recorded SMOKE's runtime performance, e.g., transformed Blocks per second, when applying the complete list of obfuscations and sanitizations that are shown as checked in Figure 4.

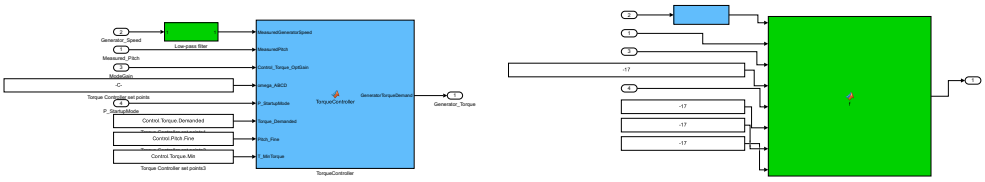
One of our goals of SMOKE is that it does not alter a model's structure, for both sanitization and obfuscation. Our first approach of validating the structural integrity was to employ an existing model comparison tool. However, the built-in tool of Simulink[13] was not able to accurately match pairs of original and transformed models. This was surprising, as it even failed to match easy cases like the one shown in Figure 9. There, the Model Comparison tool erroneously matched the green colored and the blue colored blocks of this small Subsystem. In our second attempt, we tried the clone detection tools Conqat and SIMONE. However, we did not get Conqat to run, and SIMONE is already outdated, as it is only able to handle `.mdl` models.[14]

We thus employed a signature-based model comparison [45], using model metrics for which we developed evaluation scripts specifically for this assessment. It uses simple model metrics like the number of Blocks, Lines, Subsystems, unique Block types of a model, and its cyclomatic complexity. All these metrics were either used previously in the literature [1, 3], or are given as relevant by Simulink researchers themselves [7]. We classify a model to be structurally integral if its signature remained unchanged.

5.1.3 Behavioral (Non-)Integrity of Transformations

In regard to the preservation of model behavior, our goal with SMOKE is two-fold: all obfuscation transformations shall *preserve* the original model's behavior, while

[13] mathworks.com/help/simulink/model-comparison.html
[14] Simulink uses `.slx` models by default since 2012.



(a) (b)

Figure 9: The Simulink-internal Model Comparison tool struggles to match even easy cases: here the green and blue Blocks from the original (Figure 9a) and the anonymized model (Figure 9b) are erroneously matched.

the sanitization transformations shall *alter* a model's behavior. To test whether a model's behavior is preserved after transformation, we treat models as black boxes into which we feed the same inputs and observe their outputs. If a transformed model shows a different output, given the same input, we classify it as altered behavior. We optimistically classify all obfuscated models, that show the same output as behavior-preserving. This is optimistic in the sense of our setup failing to try a 'deciding' input that produces diverging outputs. All sanitized models that have the same output are inspected, and classified manually, in a second step.

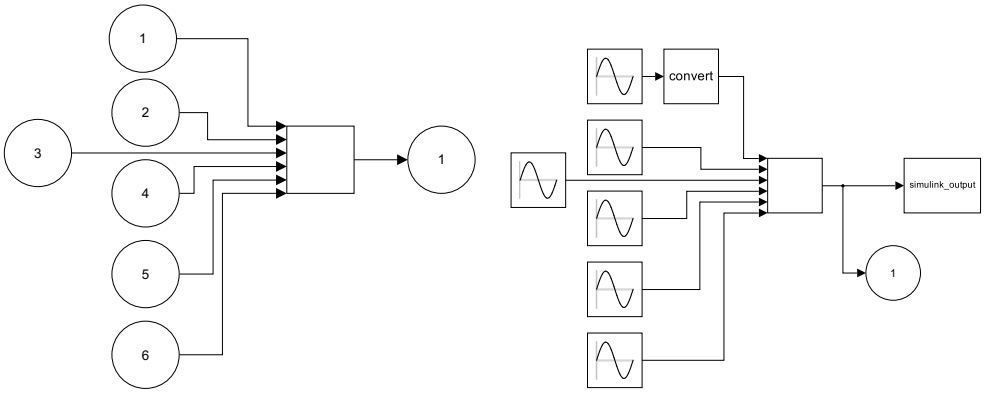
To record the input and output behavior, our evaluation scripts construct a wrapper for each model. The wrapper replaces the model inports with signal generators, and if necessary type converters, and additionally records the values of the outgoing signals, as shown in Figure 9. We then simulate both models of a pair and compare their outputs.

Note that we only construct wrappers for models that are actually compilable, and thus could demonstrate any behavior. However, we observed that often (see Table 1) the status of compilability is changed from sanitizing it. Each compilability status change is also counted as breaking the model's behavioral integrity, as one of the models can demonstrate behavior, while the other cannot.

5.2 Results: Robustness, Performance, Structural Integrity

SMOKE was able to apply all obfuscations and all sanitizations on all 8,809 models successfully. SMOKE worked at a speed of 25.4 Blocks per second, 33.8 Signals per second, 3.5 Subsystems per second, and 0.62 cyclomatic complexity units per second, when all transformations shown in Figure 4 are applied. The number of Subsystems showed the highest correlation to SMOKE execution time of +0.521, while cyclomatic complexity had the lowest correlation of +0.140.

In all models, the structural integrity was preserved, i.e., no Blocks, Lines, Subsystems were ever added or removed or types of Blocks changed. We found that the cyclomatic complexity changed in 1,954 sanitized models (never in the obfuscated models). This, however stemmed from Simulink's method of calculating cyclomatic complexity: only a compilable model's cyclomatic complexity can be calculated. The compilability of sanitized models, however, switched in many model pairs, as we can



(a) (b)

Figure 10: Our black box wrapping setup: all model inports in Figure 10a are replaced by signal generators in Figure 10b. The first port of this example model is of type boolean and a boolean converter is thus added. Similarly, all output signals are tapped into with a To Workspace Block to record the model output.

see in Table 1. We thus ignored cyclomatic complexity for our analysis, and find SMOKE to preserve structural integrity for all transformations, as desired.

5.3 Behavioral Integrity Results: Obfuscation

As expected, we found all models' behavior to be preserved by SMOKE's obfuscation transformations.

5.4 Behavioral (Non-)Integrity Results: Sanitization

Table 1: Compilation status of models before and after sanitization.

		After Sanitization	
		Compilation Fails	**Compilable**
Before Sanitization	**Compilation Fails**	5,933	199
	Compilable	1,638	1,039

In a first step of behavioral analysis, we compile all pairs of original and sanitized models – a necessary precursor to their simulation in the next step. Table 1 gives a result of the compilation. We can see that most original models are not compilable, already before the sanitization, i.e., $6,132 = 5,933 + 199$ models fail to compile from the start. This is because many of them serve some other purpose, like library models, and are not intended (or impossible) to be compiled, or run. We view non-compilable models as not showing any behavior that could be altered or evaluated automatically, and thus ignore them from the further process. The sanitization breaks the compilation

of 1,638 models, and interestingly, 199 models become compilable *after* sanitization. This is due to models being in some kind of broken state (in respect to compilation), which sometimes gets fixed by resetting Parameters with SMOKE. We view a change in compilability in a model pair as a behavior alteration, as only one of the two models can demonstrate any behavior, while the other cannot. Only 1,039 models are compilable before and after transformation, and these are the models we simulated next.

Table 2: Output status of models before and after sanitization..

		After Sanitization	
		Crash/no output	**Output**
Before Sanitization	**Crash/no output**	823	27
	Output	31	158

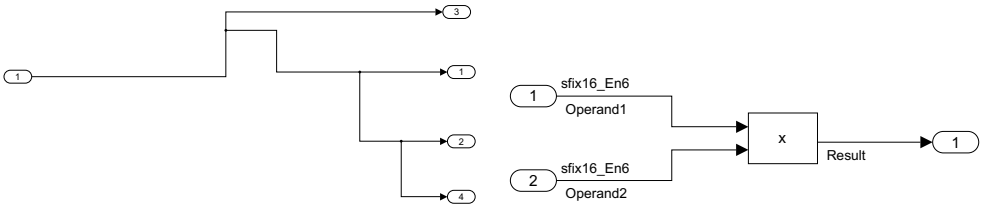
We give the results of the simulation in Table 2. Most models produce no output, i.e., they crash in their execution right away, or there was no output to harness in our test setup (e.g., library models). Similar to the compilation, we see 31 models producing no output after the sanitization, and more notably, 27 sanitized models to start producing output. Only 158 models ran their execution crash-free for both model versions. Of these, 16 models produced the exact same output. From our initial set of 8,809 models, we thus automatically classified $8,809 - 16 = 8,793$ models as behaviorally changed, or without behavior.

The models that became compilable (199), or executable (27) via the sanitization deserve a closer look. We observed that the sanitization removed custom (but broken) model parts, such as configurations, or it reset data types of Parameter values, removed faulty callbacks, etc. The other way around of sanitized models becoming broken (for compiling 1,638, or running 31) is more obvious: after the sanitization, needed variables, intra-model dependencies, or data types are missing, or Block Parameters become inconsistent with each other.

Table 3: Metric comparison of SLNET models vs. their subset of behaviorally stable models.

		Blocks	**Block Types**	**Signal Lines**	**Subsystems**
mean	**SLNET models**	133.5	10.8	177.8	18.6
	stable models ⊂ SLNET	7.6	3.4	8.7	0.3
median	**SLNET models**	30	9	36	4
	stable models ⊂ SLNET	5	3	3	0

We further inspected the last 16 stable models to determine why the sanitizing process did not alter their behavior. Our first observation is easily recognizable from



(a) (b)

Figure 11: Two exemplary models unaffected by sanitizing. They are too small, and or too simple, and SMOKE did not change any behavior-affecting Parameters in each.

Table 3: the stable models are much smaller for the various size metrics than their counterparts of the complete SLNET set. Most of their implementations are also completely flat, i.e., they are devoid of any Subsystems. This is intuitive because, on average, smaller models exhibit less complex behavior that could be affected by our sanitizations.

A manual inspection further showed that these models are also simple, probably toy projects. We give two example models in Figure 11, where one can see them to be too simple and too small for the sanitization to have any effect. We argue they show no real functionality that needs to be sanitized, i.e., is sensitive for users, in the first place.

In conclusion, SMOKE's sanitization altered the behavior of most models, with only a few small and trivial models remaining stable. As we believe models that contain sensitive information are also complex and or big, i.e., containing user-changed Parameters or Functions, we view SMOKE's sanitization as effective.

5.5 Coverage

In a final step to enhance SMOKE's capabilities, we examined the raw model files to improve its coverage, i.e., we checked if SMOKE leaves potentially sensitive model elements untouched. In the SLNET model files, we found 160 different Block types and 10,731 different Parameters. These Parameters are either model-wide Parameters or for its various elements, such as Lines, Blocks, Annotation, etc. Many of these Parameters are for internal Simulink use, and users are not supposed to edit them.

SMOKE does not support *all* of these Blocks and Parameters, as many of them are not supposed to be changed, or would need individual handling in our implementation, which is not feasible. When designing SMOKE, we first came up with a number of obfuscation and sanitization candidates (compare Section 4.1). After anonymizing them, we next ran a script that finds model elements that SMOKE has thus far left out, but could potentially hold sensitive information. These sensitive parts were then included into transformations of SMOKE, and we started the process again. To identify possibly sensitive parts, our script gathered all unique values in all models for each Parameter. We manually went over this list of values and inspected the first, or the first couple of them, to see whether they might hold valuable information. If they did, we updated SMOKE so that they are also anonymized. Our heuristic here was to give a closer look at the values of text or numeric type – especially if

there were more than 10 different values for a Parameter. Our argument here is that Parameters holding few different values, or even always the same value, are less likely to be sensitive to disclosure. We further inspected a handful of raw transformed model files visually, to see whether any more possibly sensitive information might still be left. Our investigation process does not guarantee that *every possible* sensitive part of models will be removed by SMOKE, but it does remove all those that we found.

5.6 Threats to Validity

5.6.1 Internal Threats

The biggest threat to our internal validity is in our test setup for running the models. Our harnessing to capture outputs (cf. Figure 10) is based on the heuristic that the model inputs and outputs are not hidden somewhere in the model but are on the outer layer. Additionally, some models may need inputs of certain types or values, which our test setup did not try. However, the size of our model pool made a non-model-specific and fast heuristic necessary. Furthermore, our test setup captured outputs for more than a fourth of the models that were compilable (cf. Table 2), while many of the models were never intended to be run in the first place.
Another aspect of our model output analysis in Section 5.4 is that we only measured whether the output of a sanitized model differed *in any way* to the original model's output. We cannot sufficiently decide whether "enough" of a model or a model's behavior is altered or removed. Users have to make sure themselves whether a transformed model satisfies their need for information protection.

Regarding the ability to reverse-engineer information or SMOKE's coverage (Section 5.5): we did not investigate, whether users (or Simulink) hides possibly valuable data in an encrypted way in the model (or raw files). During our investigation of the raw files, we found a few parts that were not intelligible, but we suspect them to be for internal Simulink purposes and thus ignored them. We otherwise completely remove model elements or map Parameter values to the same value. Mathematically, this means that reversing our transformations would require reconstructing the original elements or Parameter values, via a reverse transformation, that would have to 'guess' correctly [8]. Such guessing might be helped, if enough contextual information is still present in the model. We thus caution users of SMOKE to anonymize enough, so that the anonymized parts, cannot be inferred from the rest of the model using domain knowledge.

5.6.2 External Threats

Although the SLNET evaluation set is extensive and diverse, we have not yet tested SMOKE with actual corporate models, which are its intended target. However, prior studies have shown that a subset of models from SLNET is 'industry-like' [1]. While potential industry partners may have additional requests of obfuscating or deleting elements that SMOKE currently ignores, such features are easily integrated, as we already demonstrated in SMOKE's evolution (compare Sections 4.2.1 and 5.5).

6 Conclusion

With SMOKE, we provide a versatile tool that allows users to share Simulink models while protecting their intellectual property. The tool enables selective and fine-grained obfuscation or sanitization of models, all while preserving their structure. Our hope is that SMOKE will enhance collaboration among researchers and industry partners by facilitating the selective protection of models. This will enable secure sharing within the research community and between companies.

We are currently integrating SMOKE as a filter step into a workflow of creating research data management containers [46, 47]. 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. SMOKE is open-source[15] and easily extendable, e.g., with additional transformations. We welcome any suggestions for additional features the community would like to see integrated.

References

[1] Boll, A., Brokhausen, F., Amorim, T., Kehrer, T., Vogelsang, A.: Characteristics, potentials, and limitations of open-source simulink projects for empirical research. Software and Systems Modeling **20**(6), 2111–2130 (2021)

[2] Bertram, V., Maoz, S., Ringert, J.O., Rumpe, B., Wenckstern, M.: Component and connector views in practice: An experience report. In: 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS, pp. 167–177. IEEE Computer Society, Austin, TX, USA (2017)

[3] Shrestha, S.L., Boll, A., Chowdhury, S.A., Kehrer, T., Csallner, C.: Evosl: A large open-source corpus of changes in simulink models & projects. In: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 273–284 (2023)

[4] Boll, A., Vieregg, N., Kehrer, T.: Replicability of experimental tool evaluations in model-based software and systems engineering with matlab/simulink. Innovations in Systems and Software Engineering, 1–16 (2022)

[5] Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L.B., Bourne, P.E., *et al.*: The fair guiding principles for scientific data management and stewardship. Scientific data **3**(1), 1–9 (2016)

[6] Shrestha, S.L., Chowdhury, S.A., Csallner, C.: Slnet: a redistributable corpus of 3rd-party simulink models. In: Proceedings of the 19th International Conference on Mining Software Repositories. MSR '22, pp. 237–241. Association for Computing Machinery, New York, NY, USA (2022)

[15] https://github.com/lanpirot/SMOKE

[7] Shrestha, S.L., Boll, A., Kehrer, T., Csallner, C.: Scoutsl: An open-source simulink search engine. In: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 70–74 (2023)

[8] Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM systems journal **45**(3), 621–645 (2006)

[9] Oliveira, S.R.M., Zaiane, O.R.: Protecting sensitive knowledge by data sanitization. In: Third IEEE International Conference on Data Mining, pp. 613–616 (2003)

[10] Pantelic, V., Postma, S., Lawford, M., Jaskolka, M., Mackenzie, B., Korobkine, A., Bender, M., Ong, J., Marks, G., Wassyng, A.: Software engineering practices and simulink: bridging the gap. International Journal on Software Tools for Technology Transfer **20**, 95–117 (2018)

[11] Jaskolka, M., Pantelic, V., Wassyng, A., Lawford, M.: A comparison of componentization constructs for supporting modularity in simulink. SAE Technical Paper (2020-01-1290) (2020)

[12] Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical report, Department of Computer Science, The University of Auckland, New Zealand (1997)

[13] Tevajärvi, J.: Protecting intellectual property in multi-supplier ship powertrain co-simulation. Master's thesis, Aalto University, Otaniemi (December 2 2023)

[14] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference. Lecture Notes in Computer Science, vol. 2139, pp. 1–18. Springer, Berlin/Heidelberg, Germany (2001)

[15] Chowdhury, S.A., Varghese, L.S., Mohian, S., Johnson, T.T., Csallner, C.: A curated corpus of simulink models for model-based empirical studies. In: Bures, T., Fitzgerald, J.S., Schmerl, B.R., Weyns, D. (eds.) Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems, ICSE 2018, Gothenburg, Sweden, May 27, 2018, pp. 45–48. ACM, New York City, NY, USA (2018)

[16] Amorim, T., Boll, A., Bachman, F., Kehrer, T., Vogelsang, A., Pohlheim, H.: Simulink bus usage in practice: an empirical study. Journal of Object Technology **22**(2), 2–114 (2023)

[17] Stephan, M., Cordy, J.R.: Identification of simulink model antipattern instances using model clone detection. In: Lethbridge, T., Cabot, J., Egyed, A. (eds.) 18th ACM/IEEE International Conference on Model Driven Engineering Languages

and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015, pp. 276–285. IEEE Computer Society, New York City, NY, USA (2015)

[18] Schlie, A., Wille, D., Schulze, S., Cleophas, L., Schaefer, I.: Detecting variability in matlab/simulink models: An industry-inspired technique and its evaluation. In: Cohen, M.B., Acher, M., Fuentes, L., Schall, D., Bosch, J., Capilla, R., Bagheri, E., Xiong, Y., Troya, J., Cortés, A.R., Benavides, D. (eds.) Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume A, Sevilla, Spain, September 25-29, 2017, pp. 215–224. ACM, New York City, NY, USA (2017)

[19] Reicherdt, R., Glesner, S.: Slicing MATLAB simulink models. In: Glinz, M., Murphy, G.C., Pezzè, M. (eds.) 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland, pp. 551–561. IEEE Computer Society, New York City, NY, USA (2012)

[20] Schlie, A., Schulze, S., Schaefer, I.: 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, pp. 160–171. IEEE Computer Society, New York City, NY, USA (2018)

[21] Mikulcak, M., Herber, P., Göthel, T., Glesner, S.: Information flow analysis of combined simulink/stateflow models. Inf. Technol. Control. **48**(2), 299–315 (2019)

[22] Sihler, F., Tichy, M., Pietron, J.: One-way model transformations in the context of the technology-roadmapping tool iris. Journal of Object Technology **22**(2), 2–114 (2023). The 19th European Conference on Modelling Foundations and Applications (ECMFA 2023)

[23] Lobrano, G.: Making Sense of Competition Law Compliance For, A Practical Guide for SMEs. Business Europe, Brussels, Belgium (2017)

[24] López, J.A.H., Cuadrado, J.S.: An efficient and scalable search engine for models. Software and Systems Modeling **21**(5), 1715–1737 (2022)

[25] Reza, S.M., Badreddin, O., Rahad, K.: Modelmine: a tool to facilitate mining models from open source repositories. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. MODELS '20. Association for Computing Machinery, New York, NY, USA (2020)

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

[27] Boll, A., Kehrer, T., Goedicke, M.: Smoke: Simulink model obfuscator keeping structure. In: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. MODELS Companion '24, pp. 41–45. Association for Computing Machinery, New York, NY, USA (2024)

[28] Klee, H., Allen, R.: Simulation of Dynamic Systems with MATLAB® and Simulink®, 3rd edn. Crc Press, Boca Raton, FL, USA (2018)

[29] Di Ruscio, D., Kolovos, D., Lara, J., Pierantonio, A., Tisi, M., Wimmer, M.: Low-code development and model-driven engineering: Two sides of the same coin? Software and Systems Modeling **21**(2), 437–446 (2022)

[30] Liggesmeyer, P., Trapp, M.: Trends in embedded software engineering. IEEE Software **26**(3), 19–25 (2009)

[31] Pajic, M., Jiang, Z., Lee, I., Sokolsky, O., Mangharam, R.: From verification to implementation: A model translation tool and a pacemaker case study. In: 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium, pp. 173–184 (2012)

[32] Sánchez, B., Zolotas, A., Rodriguez, H.H., Kolovos, D., Paige, R.: On-the-fly translation and execution of ocl-like queries on simulink models. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 205–215 (2019). IEEE

[33] Ceccato, M., Di Penta, M., Falcarin, P., Ricca, F., Torchiano, M., Tonella, P.: A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. Empirical Software Engineering **19**, 1040–1074 (2014)

[34] Atallah, M., Bertino, E., Elmagarmid, A., Ibrahim, M., Verykios, V.: Disclosure limitation of sensitive rules. In: Proc. 1999 Workshop on Knowledge and Data Engineering Exchange (KDEX'99), pp. 45–52 (1999). IEEE

[35] Fill, H.-G.: Using obfuscating transformations for supporting the sharing and analysis of conceptual models. In: Robra-Bissantz, S., Mattfeld, D. (eds.) Multikonferenz Wirtschaftsinformatik 2012 - Teilkonferenz Modellierung Betrieblicher Informationssysteme. GITO Verlag, Braunschweig (2012)

[36] Nacer, A.A., Goettelmann, E., Youcef, S., Tari, A., Godart, C.: Obfuscating a business process by splitting its logic with fake fragments for securing a multi-cloud deployment. In: 2016 IEEE World Congress on Services (SERVICES), pp. 18–25 (2016). IEEE

[37] Martínez, S., Gerard, S., Cabot, J.: On the need for intellectual property protection in model-driven co-engineering processes. In: Reinhartz-Berger, I., Zdravkovic, J., Gulden, J., Schmidt, R. (eds.) Enterprise, Business-Process and Information Systems Modeling, pp. 169–177. Springer

[38] Weber, T., Weber, S.: Model everything but with intellectual property protection - the deltachain approach. In: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. MODELS '24, pp. 49–56. Association for Computing Machinery, New York, NY, USA (2024)

[39] Gupta, N., Chen, F., Tsoutsos, N.G., Maniatakos, M.: Obfuscade: Obfuscating additive manufacturing cad models against counterfeiting: Invited. DAC '17. Association for Computing Machinery, New York, NY, USA (2017)

[40] Zhou, M., Gao, X., Wu, J., Grundy, J., Chen, X., Chen, C., Li, L.: ModelObfuscator: Obfuscating model information to protect deployed ML-based systems. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2023, pp. 1005–1017. Association for Computing Machinery

[41] Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical Report 148, University of Auckland, New Zealand (1997)

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

[43] Bubenik, R.G.: Optimistic computation. PhD thesis, Rice University (1990)

[44] Shrestha, S.L., Chowdhury, S.A., Csallner, C.: Replicability study: Corpora for understanding simulink models & projects. In: 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–12 (2023)

[45] Selonen, P., Kettunen, M.: Metamodel-based inference of inter-model correspondence. In: 11th European Conference on Software Maintenance and Reengineering (CSMR'07), pp. 71–80 (2007). IEEE

[46] Goedicke, M., Lucke, U.: Research data management in computer science - nfdixcs approach. In: Demmler, D., Krupka, D., Federrath, H. (eds.) 52. Jahrestagung der Gesellschaft Für Informatik, INFORMATIK 2022, Informatik in Den Naturwissenschaften, 26. - 30. September 2022, Hamburg. LNI, vol. P-326, pp. 1317–1328. Gesellschaft für Informatik, Bonn, Bonn, Germany (2022)

[47] Laban, F.A., Bernoth, J., Goedicke, M., Lucke, U., Striewe, M., Wieder, P., Yahyapour, R.: Establishing the research data management container in nfdixcs. In: Sure-Vetter, Y., Goble, C.A. (eds.) 1st Conference on Research Data Infrastructure - Connecting Communities, CoRDI 2023, Karlsruhe, Germany, September 12-14, 2023. TIB Open Publishing, New York City, NY, USA (2023)