Abstract—Model-based languages such as MATLAB/Simulink are crucial for the development of embedded software systems. To adapt to changing requirements, engineers commonly copy and modify existing systems to create new variants. Commonly referred to as clone-and-own, this reuse strategy is easy to apply and beneficial in the short term, but it entails severe maintenance and consistency issues in the long term, leading to a huge amount of redundant and similar assets. Moreover, a later transition towards structured reuse such as with software product lines inevitably requires the comparison of all existing variants prior to the actual migration. However, current work mostly revolves around the comparison of only two systems and despite approaches proposed that can cope with more, such are not applicable to embedded software systems such as MATLAB/Simulink.

In this paper, we bridge this gap and propose Static Connectivity Matrix Analysis (SCMA), a novel comparison procedure that allows for the evaluation of multiple MATLAB/Simulink model variants at once. In particular, we transform models into a matrix form which is used to compare all models and to identify all similar structures between them, even with model parts being completely relocated during clone-and-own. We allow engineers to tailor results and to focus on any arbitrary variant subset, enabling individual reasoning prior to migration. We provide a feasibility study from the automotive domain, showing our matrix representation to be suitable and our technique to be fast and precise.

I. INTRODUCTION

Embedded systems are prevalent in various industrial domains such as factory automation, avionic, automotive and rail [1]. In such fields where large, complex, and safety-critical systems are developed, their reliability and maintainability are vital [2]. Model-Driven Engineering (MDE) is a paradigm commonly used in these domains and is known to improve reliability [3]. Instead of manually implementing functionality with imperative programming languages, MDE uses function-block-based designs [1]. To this end, modeling languages such as MATLAB/Simulink\(^1\) are used, which enforce modularity and are considered to enhance maintainability [4], [5]. While they allow for functionality to be implemented on a more intuitive level for engineers, overall development and evolution of embedded software systems remains a challenging and time-intensive task. In order to adapt to new requirements, engineers commonly reduce both work and time effort by copying and subsequently modifying existing systems [6].

\(^1\)MathWorks\(®\)- http://www.mathworks.com/simulink/ - July 2018

Denoted clone-and-own [7], this reuse approach is easy to use, as existing functionality is copied rather than reimplemented. However, in the long-run, this technique has severe implications on maintenance and evolution of the variant portfolio [8], [9]. Among others, redundancies emerge that need to be evolved and maintained separately as knowledge about commonalities and differences between variants is not present.

Recent approaches addressed this problem by exploiting modularity of modeling languages for variant composition, and thus, mitigate the drawbacks of clone-and-own [10], [11]. While this may help to fight the symptoms of clone-and-own, it does not allow for a comprehensive understanding of such variants. It also does not tackle the root cause, which is unstructured reuse. Thus, redundant artifacts across systems and the inherently increased maintenance effort are still pervasive [9].

In contrast, Software Product Lines (SPLs) [12], [13] facilitate strategic reuse and promote maintainability while preventing redundancies between related software systems [14]. In practice, however, it is hard to foresee the entire scope of functionality required upfront [15]. Thus, families of similar systems often emerge ad-hoc using clone-and-own, resulting in a proliferation of legacy systems without information about their relations [16], [17]. The need to transition from clone-and-own towards an SPL often becomes evident only after variant genesis and requires major migration [15], [18]–[20].

Unfortunately, reactively migrating variants to an SPL or taking other measures to instantiate structured reuse, and thus, improving maintainability, poses an enormous challenge to practitioners [8], [21]–[23]. In particular, current approaches incrementally compare pairs of related systems [24], which considerably impose efficiency and accuracy of the results [25], [26].

Hence, we argue that it is inevitable to consider all relevant systems at once for their strategic migration towards an SPL. To allow for their assessment and eventually, migration towards an SPL practice, it is indispensable to compare all system variants with each other rather than contending with a restricted evaluation such as incremental pairwise approaches. However, in MDE, most work addressing SPL migration strategies and system maintainability are only applicable to two systems, which limits applicability in-the-wild [26], [27]. Although approaches have been proposed that evaluate multiple models [22], [26]–[28], they are not applicable to an entire family of complex MATLAB/Simulink model variants.

\textbf{Comparing Multiple MATLAB/Simulink Models Using Static Connectivity Matrix Analysis}
In this paper, we tackle the aforementioned problems and propose our Static Connectivity Matrix Analysis to compare multiple MATLAB/Simulink models. To cope with an entire portfolio of models, we introduce the Connectivity Matrix, an intermediate representation that exploits the data flow and modular construction inherent to MATLAB/Simulink models. With our technique, we provide a comprehensive overview of the entire model hierarchy and highlight in gray the contained SMs labeled 2, 3, and 6. For clarity, models throughout this paper contain unique block labels only. The corresponding graph representation includes an artificial SM named root, highlighted in gray respectively. It comprises all blocks residing on the first hierarchical layer δ₀ to illustrate M₀’s model hierarchy being a tree structure. We refer to the SM labeled δ on the second hierarchical layer δ₁ as the child system of its respective parent system, the SM labeled 2 in Figure 1. Every SM on any layer δ_j is the root of its respective subtree and, thus, every model structured with SMs can be represented as a tree (cf. Figure 1).

The remainder of this paper is structured as follows. We provide background information on MATLAB/Simulink models and properties we utilize for our technique and outline descriptors to abstract from complex systems (Sec. II). We introduce our descriptor, the Connectivity Matrix (CM) and propose our SCMA (Sec. III). We assess our technique using a feasibility study from the automotive domain (Sec. IV) and propose our B. Model Abstraction Using Descriptors

In this section, we provide details on MATLAB/Simulink models, properties that we utilize for our proposed technique, and introduce descriptors [29] to abstract from complex systems.

A. MATLAB/Simulink

MATLAB/Simulink is a block-based behavioral modeling language that utilizes functional blocks and signals to specify certain software system functionality. It is vital for the development of embedded software systems in various industrial domains such as avionic and automotive engineering [30], [31]. Such models constitute the central development artifact and are used to generate code for operation on microcontrollers [32].

Each block of a MATLAB/Simulink model either represents a specific functionality or it is used to structure the model. Every block has a set of syntactical and semantical properties that allow for its identification and to be compared with other blocks. Focusing on MATLAB/Simulink, the following block properties are of interest for the remainder of this paper:

- **function**: Represented function, i.e., what the block is used for.
- **label**: Non-unique textual name of the block.
- **interfaces**: For incoming and outgoing data:
  - **in-ports**: A block contains an arbitrary number of in-ports. Each in-port connects to exactly one out-port.
  - **out-ports**: A block contains an arbitrary number of out-ports. Each out-port can connect to one or more in-ports.
- **signal**: A directed edge, connecting in-ports and out-ports.

Industrial MATLAB/Simulink models comprise thousands of blocks to capture complex system behavior [16], [30], [32]. Logically connected blocks are commonly grouped together and encapsulated within a Subsystem (SM) block. SMs can be nested and structured the model horizontally, constituting a model hierarchy [1]. Every SM resides on a specific hierarchical layer δ_j that corresponds with its nesting depth.

Complex models comprise numerous hierarchical layers and can exhibit a total hierarchical depth of ten and more [33]. In Figure 1, we depict a simple MATLAB/Simulink model M₀, and highlight in gray the contained SMs labeled 2, 3, and 6. For clarity, models throughout this paper contain unique block labels only. The corresponding graph representation includes an artificial SM named root, highlighted in gray respectively. It comprises all blocks residing on the first hierarchical layer δ₀ to illustrate M₀’s model hierarchy being a tree structure. We refer to the SM labeled δ on the second hierarchical layer δ₁ as the child system of its respective parent system, the SM labeled 2 in Figure 1. Every SM on any layer δ_j is the root of its respective subtree and, thus, every model structured with SMs can be represented as a tree (cf. Figure 1).

B. Model Abstraction Using Descriptors

To allow for the analysis of multiple MATLAB/Simulink models in their entirety, their complexity needs to be reduced first. Extensively applied in various fields such as robotics, image processing, network and electrical circuit design [34]–[37], descriptors abstract from complex systems by describing salient system information and representing them in a simpler format. Descriptors can be compared efficiently, and thus, even allow for the comparison of large quantities of data [38]. A descriptor should exhibit the following key characteristics [39]:

- The descriptor should be easy to extract from the original model.
- There should be a low probability of mismatch, that is, two distinct models should not result in the same descriptor.

Given their numerical efficiency, matrices have prevailed as one of the most widely utilized descriptor formats [29], [40], [38]. Furthermore, matrices are a generally accepted representation for graph structures and regarding their numerical efficiency, are intrinsically suitable for large-scale graph transformation and analysis procedures [41]. As shown in Figure 1, function block diagrams such as MATLAB/Simulink inherently constitute such graph structure. Consequently, we utilize matrices in our technique to derive a descriptor, the Connectivity Matrix, to abstract from MATLAB/Simulink models.
III. STATIC CONNECTIVITY MATRIX ANALYSIS

In this section, we propose Static Connectivity Matrix Analysis to identify and group all similar SM structures across all MATLAB/Simulink model variants, regardless of their location or their input order. Given SCMAs’ workflow in Figure 4, we illustrate its four sequentially processed phases using the three models shown in Figure 5 for the remainder of this section.

A. Descriptor Creation - The Connectivity Matrix

We introduce our descriptor, the Connectivity Matrix, to abstract MATLAB/Simulink SMs into a matrix representation. Inherently characteristic of function block diagrams, MATLAB/Simulink models and, thus, SMs, are the composition of directly connected blocks, each of them having a specific function.

The CM exploits this property to approximate a given SM. Precisely, a CM represents which two block functions directly connect and how often they connect within the evaluated SM. For the models \(M_1\) and \(M_2\) from Figure 2, we show the corresponding CMs created by SCMA, \(CM_1\) and \(CM_2\), in Figure 3. The depicted models contain six signals as well as eight blocks in its row and column construction. Only the entries of the every single CM has the same dimensions and is identical in its row and column construction. For readability reasons, non-present connections are left blank.

The CMs from Figure 3 highlight in gray the single connection present in both abstracted models \(M_1\) and \(M_2\) from Figure 2, connecting the functional block types Gain and Outport. To ensure all CMs to have the same dimensions, SCMA preprocesses all input models and generates a dictionary of all distinct block functions. The size of the dictionary then determines the dimensions of every single CM. For instance, \(CM_2\) in Figure 3 contains the block Integrator as a row and column entry, although this block function is not present in the corresponding model \(M_2\). However, this function is present within the input model \(M_1\). It is, therefore, part of the dictionary and used to construct the CMs.

The retrieved dictionary is ordered and constitutes both CM axes, resulting in CMs being quadratic (cf. Figure 3). Moreover, when preprocessing SMs to create the dictionary, we already store all connections between any block functions within the corresponding CM. The order of the dictionary’s entities can be chosen arbitrarily after preprocessing but it must be fixed, and thus, static for subsequent CM creation. As illustrated in Figure 4, every input model is processed separately and for each SM, a corresponding CM is created. For a MATLAB/Simulink model comprising \(k\) SMs, \(k\) CMs are generated by SCMA plus one additional CM to represent connections present on the top hierarchical layer \(\delta_0\) (cf. Sec. II-A). Each CM holds a reference to the specific SM it represents.

In Table I, we list all entities required to transform any SM from a MATLAB/Simulink model into its respective CM. We explicate the applied transformation procedure in Algorithm 1.

![Figure 2: Models M1 & M2](image1)

![Figure 3: CMs for M1 & M2](image2)

Fig. 2: Models \(M_1\) & \(M_2\)

Fig. 3: CMs for \(M_1\) & \(M_2\)

For the remainder of this paper, we refer to a connection as both functions of a specific signals’ source- and target block. Within any CM, the source of a connection always resides on the x-axis whereas its target resides on the y-axis (cf. Figure 3). Connections can be present multiple times in a certain SM. If, for instance, blocks with the functions Gain and Outport connect \(n\) times, the corresponding entry in the CM would be \(n\).

We refer to the CM as static, meaning that upon instantiation, every single CM has the same dimensions and is identical in its row and column construction. Only the entries of the CMs vary, depending on the individual connections present within the specific SM abstracted by the CM (cf. Figures 2 & 3). To illustrate its four sequentially processed phases using the three models shown in Figure 5 for the remainder of this section.

![Algorithm 1: Creating the Connectivity Matrices](image3)

**Algorithm 1:** Creating the Connectivity Matrices

<table>
<thead>
<tr>
<th>Input: (\Phi, M_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: (\Phi)</td>
</tr>
<tr>
<td>(CM_i) (\rightarrow) (\emptyset)</td>
</tr>
<tr>
<td>(\forall SM_j \in M_i) do</td>
</tr>
<tr>
<td>(A(n \times n)) (\rightarrow) (\emptyset)</td>
</tr>
<tr>
<td>(\forall B_{SM_j} \in SM_j) do</td>
</tr>
<tr>
<td>(\forall Signals \varphi_{out} \in B) do</td>
</tr>
<tr>
<td>(x \leftarrow function(b))</td>
</tr>
<tr>
<td>(y \leftarrow function(target(\varphi_{out})))</td>
</tr>
<tr>
<td>(A_{ij}(x, y) \leftarrow (A_{ij}(x, y) + 1))</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>(CM_i \leftarrow CM_i \cup {A_{ij}}) Store the matrix (A_{ij}) in the set of matrices for the current model (M_i)</td>
</tr>
<tr>
<td>(\Phi \leftarrow \Phi \cup {CM_i})</td>
</tr>
<tr>
<td>return (\Phi)</td>
</tr>
</tbody>
</table>

**TABLE I:** Entities Required for Creating the CM

| \(M_i\) | A MATLAB/Simulink model |
| \(SM_j\) | A SM from the model \(M_i\) |
| \(A_{ij}\) | A CM representing the SM \(j\) from the model \(M_i\) |
| \(CM_i\) | Set of all CMs for the model \(M_i\) |
| \(\Phi\) | Set of all sets \(CM_i\) for all models \(M_i\) |

Fig. 4: Workflow of the Static Connectivity Matrix Analysis (SCMA)
Preprocessing the models from Figure 5 yields eleven distinct block functions. The model \( M_A \) comprises nine functions, \( M_B \) introduces the *Summation* and *Inverse* function located in its SMs \( B:3 \) & \( B:15 \) (cf. blocks *SUM* & 18) and \( M_C \) does not exhibit new functions. The order in which models are processed to retrieve all distinct functions for the dictionary can be arbitrary.

Using the model \( M_B \) from Figure 5 as input for Algorithm 1, an empty set \( CM_B \) is created (cf. Line 1). For every SM, present within the evaluated model \( M_B \) (cf. Line 2), a new CM \( A_B \) is created that holds references to both, the specific SM \( A \), it abstracts as well as the correponding MATLAB/Simulink model (cf. Line 3). The dimensions of any CM correspond to the size of the retrieved dictionary, hence \( n=11 \) for the three models shown in Figure 5, and therefore, \( A_j \) \( (11 \times 11) \). For every block within the current SM, all outgoing signals \( \varphi_{out} \) are evaluated and the connection established between the source and the target blocks’ function is stored in the corresponding field of the CM (cf. Lines 4-8 & Figure 3). An existing entry is simply incremented by one when processing a connection that is already present within the CM. Every CM \( A_B \) is stored in the set \( CM_B \) prior to processing further SMs (cf. Line 11). Once all SMs of \( M_B \) have been processed, the resulting set \( CM_B \) is stored in \( \Phi \) (cf. Line 13).

For the models from Figure 5, all CMs generated by SCMA are shown in Figure 6, illustrating that they preserve the parent-child relation that exists between the associated SMs. Hence, CMs exhibit a hierarchical depth \( \delta_j \) and fully resemble the entire model hierarchy (cf. Figures 5 & 6). Illustrated in Figure 6, we refer to the CM labeled \( B:3 \) as the parent \( P \) of its respective child, the CM labeled \( B:15 \) (cf. Sec. II-A).

### B. Comparing Connectivity Matrices

To identify all similar CMs across all models despite relocation, SCMA compares all generated CMs with each other. More precisely, any CM from a model \( M_X \) is compared with all other CMs except those from \( M_X \). For instance, the CM \( A:0 \) from Figure 6 is, therefore, not compared with itself or another CM (e.g. \( A:3 \)) associated with the model \( M_A \), but with each CM associated with another model (here: \( M_B \) and \( M_C \)). With SCMA, the focus is not to analyze a single but multiple model variants to support their migration towards an SPL. However, even the former can be achieved by SCMA using a single model and its copy only. For each CM comparison, a similarity value \( 0 \leq \omega \leq 1 \) is calculated that represents the normalized distance between all two entries with the same \( x \)- and \( y \)-coordinates. Thus, \( \omega \) reflects to what extent the respective SMs exhibit the same connections between certain functions.

Algorithm 2 details the procedure applied with SCMA to compare any two CMs. The entities required for Algorithm 2 are provided in Table II and complement those listed in Table I.

<table>
<thead>
<tr>
<th>TABLE II: Entities Required for Comparing CMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega(A_{ij}, B_{xy}) )</td>
</tr>
<tr>
<td>( A(x, y) )</td>
</tr>
<tr>
<td>( \Omega )</td>
</tr>
</tbody>
</table>

#### Algorithm 2: Comparison of the Connectivity Matrices

**Input:** \( \Phi \)  
**Output:** \( \Omega \)

1. \( \Omega \leftarrow \emptyset \)

2. **foreach** \( CM_i \in \Phi \) **do** 
   
3. \( j \leftarrow (i+1) \)

4. **while** \( j < |\Phi| \) **do** 

5. **foreach** Matrix \( A \in CM_i \) **do**

6. **forall** Matrix \( B \in CM_j \) **do**

7. \( \omega(A, B) = \sum_{x=1}^{n} \sum_{y=1}^{n} \left\{ \begin{array}{ll} A(x,y) = B(x,y) = 0 & \sum_{x=1}^{n} \sum_{y=1}^{n} A(x,y) = B(x,y) = 0 \\ 0 & \text{else} \end{array} \right. \)

8. \( \Omega \leftarrow \Omega \cup \{ \omega(A, B) \} \)

11. **end**

12. **end**

13. **return** \( \Omega \)

Fig. 5: MATLAB/Simulink Models Used as a Running Example for SCMA

![Fig. 5: MATLAB/Simulink Models Used as a Running Example for SCMA](image-url)

Fig. 6: CMs created by SCMA for the Models from Figure 5

![Fig. 6: CMs created by SCMA for the Models from Figure 5](image-url)
Prior comparison, \( \Omega \) is initialized to store every calculated similarity value \( \omega \) along with its associated CMs \( A \& B \) (cf. Line 1). Every set of CMs within \( \Phi \) (cf. Line 2) is compared with all remaining sets representing different models (cf. Lines 2 to 4). For instance, given the sets \( CM_{A\&C} \) from Figure 6, Algorithm 2 compares all CMs present within \( CM_A \) with all CMs contained in \( CM_B \) and \( CM_C \). Consequently, for \( CM_B \), only \( CM_C \) remains and for \( CM_C \) itself, no further comparisons are necessary. This is because the similarity value calculation (cf. Line 7) is based on the minimum and maximum of the specific matrix entries, and thus, is commutative. Entries that are non-present in both CMs and by that, indicating non-existing connections in the SMs, are disregarded. Taking such connections into account would wrongfully increase the similarity value, and thus, adversely affect its soundness. Within \( CM_1 \) and \( CM_2 \) from Figure 3, a total of five entries exist that are non-zero in either of both CMs. For instance, for the connection Gain - Outport, the calculated similarity value would be \( \frac{1}{2} = 0.5 \). Comparing the CMs from Figure 3 in their entirety yields a similarity value of \( \frac{(\frac{1}{2}) + 3(\frac{1}{2}) + (12+0)}{4} = 0.125 \). For the CM sets \( CM_{A\&C} \) from Figure 6, representing the MATLAB/Simulink models \( M_{A\&C} \) from Figure 5, Algorithm 2 yields a total of 47 distinct CM comparisons. In Table III, we list all similarity values \( \omega \) calculated by Algorithm 2 for the CMs illustrated in Figure 6. Gray entries represent comparisons between CMs for which the associated SMs originate from the same MATLAB/Simulink model (cf. Figure 5). Blank entries depict comparisons that are obsolete because of the commutativity of \( \omega \).

### TABLE III: Similarity Values for CMs from Figure 6

<table>
<thead>
<tr>
<th>( \omega )</th>
<th>A:0</th>
<th>A:3</th>
<th>A:10</th>
<th>B:0</th>
<th>B:3</th>
<th>B:10</th>
<th>B:15</th>
<th>C:0</th>
<th>C:10</th>
<th>C:15</th>
<th>C:19</th>
</tr>
</thead>
<tbody>
<tr>
<td>B:0</td>
<td>1.0</td>
<td>0.42</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.42</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B:3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.16</td>
</tr>
<tr>
<td>B:10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.42</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B:15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.16</td>
</tr>
<tr>
<td>C:0</td>
<td>1.0</td>
<td>0.42</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.42</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C:3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.16</td>
<td>0.16</td>
</tr>
<tr>
<td>C:10</td>
<td>0.16</td>
<td>0</td>
<td>0.16</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>C:15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C:19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CM comparisons such as \( \omega(A:0,B:0) \) reflect identical SMs (cf. Figure 5), and thus, exhibit a similarity value of \( \omega = 1.0 \). This holds for other comparisons as well, such as for those representing the SMs \( A:10 \) and \( B:10 \) or \( B:3 \) and \( C:3 \). Looking at \( A:0 \) and \( C:10 \) in both, Table III and Figure 5, there is one common connection, Subsystem to Outport (cf. blocks 10 to 2 & 19 to 12). Considering all connections, this accounts for a minute, but not necessarily negligible similarity value \( \omega \).

Furthermore, the information provided in Table III permits a preliminary assessment of more than two input models. For instance, \( A:0 \) is not only identical to \( B:0 \) but also to \( C:0 \).

### C. Structuring Connectivity Matrices - The Forest Creation

SCMA utilizes the CM comparisons to group together similar CMs within nodes. We regard CMs similar if their comparison result exceeds a threshold \( \omega_{min} \). We do not preset this value, but given their domain knowledge, allow practitioners to define it either prior to SCMA or afterwards as part of an optional filtering step (cf. Sec. III-D). The nodes get connected if the contained CMs exhibit a parent-child relation (cf. Sec. II-A & Figure 6). As a result, nodes form trees. Since each CM corresponds to a SM, a tree represents a similar hierarchical SM structure between multiple models. Regardless of their hierarchical depth or structural location across models, SCMA creates all trees, and thus, identifies all similar SM structures between all input models. We store all trees within a forest.

In Table IV, we list the entities required for the forest creation, complementing those listed in Tables I & II. We detail the procedure utilized for the forest creation in Algorithm 3.

### TABLE IV: Entities Required for Creating the Forest

<table>
<thead>
<tr>
<th>( G_t )</th>
<th>Tree with index ( t ) and nodes ( V ) and edges ( E )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Psi )</td>
<td>Forest containing all generated trees</td>
</tr>
<tr>
<td>( P_X )</td>
<td>Parent CM of the CM ( X ) (cf. Figure 6)</td>
</tr>
<tr>
<td>( \bar{\omega} )</td>
<td>Average of all similarity values ( \omega ) for a node</td>
</tr>
</tbody>
</table>

### Algorithm 3: Forest Creation Using Connectivity Matrices

**Input:** \( \Omega \)

**Output:** \( \Psi \)

1. \( \Psi \leftarrow \{ G = (V, E) \} \)
2. **foreach** \( (V_t, E_t) \in \Psi \) **do**
   3. \( insertionPossible \leftarrow false \)
   4. **TreeLoop:**
      5. **foreach** \( (V_t, E_t) \in \Psi \) **do**
         6. if \( \exists k \in V_t : \{ P_A, P_B \} \cap V \neq \emptyset \) then
            7. if \( \forall x \in V : \{ A, B \} \cap v \neq \emptyset \) then
               8. if \( \forall x \in V : \omega(x, y) > 0 \) and \( y \in \{ A, B \} \) then
                  9. \( v \leftarrow v \cup \{ A, B \} \)
                  10. \( \bar{\omega}(v) \leftarrow \sum_{(x, y) \in \omega(x, y), x \neq y} \)
                  11. **insertionPossible** \leftarrow true
                  12. **break** TreeLoop
               13. **end**
            14. **else**
               15. \( z \leftarrow \{ A, B \} \)
               16. \( \bar{\omega}(z) \leftarrow \omega(A, B) \)
               17. \( V_t \leftarrow V_t \cup \{ z \} \)
               18. \( E_t \leftarrow e(k, z) \)
               19. **insertionPossible** \leftarrow true
               20. **break** TreeLoop
            21. **end**
         22. **end**
      23. **end**
   24. **if** \( insertionPossible \) **then**
      25. \( G_{|\Psi|+1} \leftarrow (V_{|\Psi|+1}, E_{|\Psi|+1}) \)
      26. \( z \leftarrow \{ A, B \} \)
      27. \( \bar{\omega}(z) \leftarrow \omega(A, B) \)
      28. \( V_{|\Psi|+1} \leftarrow V_{|\Psi|+1} \cup \{ z \} \)
      29. \( \Psi \leftarrow \Psi \cup G_{|\Psi|+1} \)
   30. **end**
31. **end**
32. return \( \Psi \)
All CMs within each set $CM_i$ (cf. Table I) are sorted in a descending order with respect to their hierarchical depth $\delta_j$. We thus ensure that for any CM comparison, the respective parent CMs $P_i$ (cf. Sec. III-A), have already been processed. This way, Algorithm 3 can at all times establish the parent-child relation between compared CMs if such is present. With each set $CM_i$ being sorted, the order in which they are processed by Algorithm 3 can be arbitrary. In other words, the input order for MATLAB/Simulink model variants, reflected by the sets $CM_i$, is irrelevant for the forest produced by SCMA.

Setting the similarity threshold $\omega_{min}$ to 0 causes Algorithm 3 to process all comparisons provided in Table III exceeding that value. Every $\omega$ is evaluated separately (cf. Line 2) and the associated CMs are then either inserted into an existing node within a tree (cf. Lines 8-12), used to create a new node within an existing tree (cf. Lines 15-22) or utilized to start a new tree (cf. Lines 25-31). For every $\omega$ and its associated CMs, every tree is evaluated separately (cf. Line 5) and a new tree is created only if no insertion in any existing tree is possible (cf. Lines 3 & 25). If an insertion is possible, no further trees are evaluated (cf. Lines 12 & 21). Consequently, no two trees can exist that exhibit the same CM comparison, resulting in the forest to be duplicate-free. Line 6 specifically returns true when comparing top level elements (i.e. $A:0 & B:0$) that do not have parent CMs. However, no node exists within the current tree $G_t$ that either contains $A:0 & B:0$ (cf. Line 7). Hence, the CMs cannot be grouped within an existing node but are stored within a new node (cf. Line 16). Followed by $A:0 & C:0$, Line 7 now holds because $A:0$ has already been processed. Within a node, we only group together those CMs that for all possible combinations exhibit a similarity value greater zero (cf. Line 8). This way, a node being part of a larger structure only contains CMs that are at all similar to each other.

Consequently, grouping $A:0 & C:0$ together with $A:0 & B:0$ requires $\omega(B:0:C:0) \geq 0$ to be greater than zero. This holds (cf. Table III), and thus, the CMs are grouped together and the overall similarity value for that node $\tilde{\omega}$ is recalculated as the average of all comprised $\omega$ values. When evaluating $\omega(A:0:C:10)$, no tree exists that contains a node comprising both parent CMs (cf. Figure 6). Hence, a new tree and a node are created for that comparison (cf. Lines 26 & 27) and added to the forest (cf. Line 30) so that they can be used when processing further comparisons. For the comparisons from Table III, we provide all three trees generated by SCMA in Figure 7.

**D. Filtering the Forest**

The forest can be filtered to provide tailored information depending on individual demands. For instance, the similarity threshold $\omega_{min}$ is customizable and can even be set after forest creation. All CM comparisons not exceeding the specified value are then removed from the trees. Therefore, stakeholders must not necessarily set thresholds in advance, which bears the risk of losing information, especially without precise domain knowledge. For instance, setting $\omega_{min}=0.2$ would remove Tree 2 from the forest in Figure 7 while $\omega_{min}=0.5$ would additionally remove parts of Tree 1.

Specifically, we provide two filtering procedures $\chi_r$ and $\chi_s$ that can be applied to the forest $\Psi$ and filter it given a customizable subset $\Delta$ of all input models $M$. Hence, $\chi_r$ and $\chi_s$ constitute a projection [42] of the forest to a refined subset, allowing practitioners to tailor the forest to perform further reasoning. Applying $\chi_r$ results in the same output as for the forest creation $\lambda$ but only for that subset $\Delta$ alone.

$$\chi_r(\Psi, \Delta) = \lambda(\Delta), \quad \Delta \subset M \land |M| \geq |\Delta| + 2$$

More precisely, $\chi_r$ removes all CM comparisons that contain a CM with the corresponding model not contained within $\Delta$. For instance, $\Delta = \{M_A, M_B\}$ would remove Tree 3 from the forest in Figure 7 because the corresponding model for the CM $C:22$ is not in $\Delta$, and thus, the only node of Tree 3 is removed.

The second filtering procedure $\chi_s$ only retains nodes that contain CM comparisons encompassing all models within $\Delta$. For instance, given $\Delta = \{M_A, M_B, M_C\}$, $\chi_s$ would remove the bottom node from Tree 1 because it does not comprise a CM associated with $M_A$. Applying this $\Delta$ but with procedure $\chi_r$ would retain that bottom node because $B:15$ and $C:15$ associate to $M_A$ and $M_B$. Thus, $\chi_s$ produces a subset of $\chi_r$’s results.

$$\chi_s(\Psi, \Delta) = \chi_s(\chi_r(\Psi, \Delta)) \neq \lambda(\Delta)$$

Filtering can be reverted and reapplied with different settings. Thus, results can be tailored to specific demands while SCMA needs to process input models only once. With SCMA, we allow engineers to compare all input variants. Afterwards, filtering enables them to explicitly focus on a specific subset of these variants. For instance, engineers can perform additional analysis which targets certain SMs to identify more fine-grained variability [43] and to obtain profound understanding.
In this section, we provide our objectives, information about the analyzed models, and the data analysis guidelines [44] we used.

A. Research Questions

With SCMA, we compare all system variants and identify all similar structures between them. For our empirical evaluation, we use F-measure, an approach widely used in software engineering that combines precision and recall [45]. We focus on the following research questions:

RQ1: Can we regard CMs suitable to abstract MATLAB/Simulink models?

For our proposed SCMA, CMs are crucial and only suitable if they fulfill the characteristics defined in Sec. II. Hence, we evaluate if they are easy to extract and have a low probability of mismatch.

RQ2: What level of precision and recall can we achieve with SCMA?

Precision and recall are vital for engineers to accept our technique. We refer to precision as the extent to which each generated tree reflects a similar hierarchical CM structure between all analyzed models. We refer to recall as the extent to which each tree only contains CMs that are similar and that exhibit such hierarchical relation.

RQ3: Is SCMA’s performance reasonable when scaling up?

Especially in an industrial environment, an acceptable runtime is essential for our proposed technique to be applicable in practice. We refer to performance as the total runtime required and its distribution over SCMA’s three mandatory phases: CM creation, CM comparison, and forest creation. (cf. Sec. III).

B. Setup

To assess the feasibility of our proposed technique and our descriptor, the CM, we conducted a case study with real-world models from the automotive domain. Using an exemplary driver assistance system (DAS) from the publicly available SPES_XT² project, we artificially generated a set of model variants by identifying self-contained parts within the DAS model and extracting them. The extracted parts we used for the composition of model variants are listed in Table V, along with information on their overall size and structural complexity.

<table>
<thead>
<tr>
<th>Model name &amp; Abbreviation</th>
<th>#blocks</th>
<th>#SMs</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmergencyBreak ’EB’</td>
<td>409</td>
<td>43</td>
<td>7</td>
</tr>
<tr>
<td>FollowToStop (req. CC) ’FTS’</td>
<td>699</td>
<td>77</td>
<td>11</td>
</tr>
<tr>
<td>SpeedLimit ‘SL’</td>
<td>497</td>
<td>57</td>
<td>10</td>
</tr>
<tr>
<td>CruiseControl ’CC’</td>
<td>671</td>
<td>74</td>
<td>11</td>
</tr>
<tr>
<td>Distinctive (req CC) ’DT’</td>
<td>728</td>
<td>78</td>
<td>11</td>
</tr>
</tbody>
</table>

SMs – subsystem blocks, $D$ – max. hierarchical depth, req. – requires

Using the project documentation, we identified dependencies for FTS & DT that prohibit using them in isolation. Respecting the identified dependencies given in Table V, we combined the listed DAS model parts and created a total of 19 different variants that explicitly address a clone-and-own scenario. For instance, the largest model variant created contains all DAS model parts listed in Table V. Other variants contain only one DAS model part, e.g., FTS or two parts respectively, e.g., FTS and EB. From a clone-and-own standpoint, functionality was copied to the new variant and then extended by adding EB. The DAS model contains a total of 37 distinct block functions.

C. Data Analysis Guidelines

For the suitability of CMs, we evaluate their compliance with the characteristics defined for descriptors (cf. Sec. II). Hence, we first assess algorithmic complexity of the CM creation. Second, we evaluate whether (a) distinct MATLAB/Simulink SMs correctly result in distinct CMs and (b) whether distinguishable SMs exist that wrongfully result in identical CMs. To assess the feasibility of our proposed technique, manual evaluation of all possible 524,268 combinations⁵ is infeasible. For precision and recall, we, therefore, focus on 18 comparisons, ranging from the smallest including only two systems to the largest possible comparison that includes all 19 model variants. The corresponding trees, generated by SCMA, were evaluated by an expert well familiar with the DAS model. Results were assessed directly within the MATLAB/Simulink environment. For performance, we state the algorithmic complexity and examine the actual runtime and its distribution over SCMA’s mandatory phases. Each comparison was performed 10 times and the average was calculated to account for runtime deviations inherently present in a non-closed system. We implemented our technique in Java⁴ using Eclipse⁵ and its Modeling Framework⁶.

V. Results

The SPES_XT case study was evaluated on a Dual-Core i7 processor with 12 GB of RAM, running Windows™ 7 on 64bit. We can only show aggregated data in this section, but detailed results as well as a screencast on SCMA can be found online⁷.

RQ1: Suitability of the CM as a Descriptor

According to Sec. II-B, our descriptor is considered suitable if it is (a) easy to extract given a SM and (b) exhibits a low probability of mismatch for multiple SMs. For (a), creation of any CM inevitably requires a dictionary to be built in advance (cf. Sec. III-A). Hence, preprocessing input models to retrieve such a dictionary is mandatory for the CM creation, and thus, part of the overall process of deriving our descriptor. The dictionary is build by evaluating all blocks of all input models with every block and its function retrieved only once.

To this end, every SM is retrieved separately and all connections are stored accordingly within the associated CM. For each block within any SM, its own function as well as, for any of its outgoing signals, the respective target blocks’ function are retrieved once (cf. Sec. II-A, III-A & Algorithm 1). Consequently, the CM creation exhibits a linear complexity of:

$$O(n)$$ - Complexity of CM Creation

Figure 8 depicts the runtime required to create all CMs for a certain subset of model variants (cf. Sec. IV-B) given their size in the total number of contained blocks, provided on the x-axis.

$$\sum_{k=0}^{n} \binom{n}{k}$$ because comparing no or only one variant can be omitted.

⁴Eclipse Foundation® - https://eclipse.org/ - July 2018
⁵Eclipse Foundation® - https://eclipse.org/modeling/emf - July 2018
²Software Platform Embedded Systems ‘XT’, TU München - spes2020, informatik.tu-muenchen.de/spes_xt-home.html - July 2018
Each data point represents a comparison, ranging from the smallest with two models (≈ 1.500 blocks) to the largest with all 19 models (≈ 14.300 blocks). For the latter, 1528 CMs are created in ≈ 19 milliseconds, a runtime we consider acceptable.

![Graph of runtime vs total number of blocks](image)

**Fig. 8: Runtime of the Overall CM Creation**

To check for mismatches, we first analyzed the dissemination of the similarity values for all ≈ 1.1 million CM comparisons performed when using all 19 model variants as input to SCMA. In Figure 9, we show the results with similarity values ω on the x-axis and their occurrence in percentage on the y-axis.

![Graph of similarity distribution](image)

**Fig. 9: Distribution of Similarity Values for CMs**

Our data reveal that only two peaks appear within the similarity value distribution. First, with ≈ 80%, the majority of CM comparisons exhibit a similarity of ω = 0, and thus, indicate distinct SMs. Secondly, ≈ 5% of all CM comparisons exhibit a similarity value of ω = 1, and thus, indicate identical SMs. Moreover, the remaining ≈ 15% distribute almost equally with a slight separation of values for ≈ 0.3 ≥ ω ≥ 0.6 (cf. Figure 9). For the four similarity value increments from ω = 0.2 to ω = 0.8 shown on the x-axis in Figure 9, we manually evaluated 100 CM comparisons with their associated SMs varying in size, yielding a total of 400 comparisons. Additionally, we evaluated 2000 comparisons with a similarity value of ω = 0 as well as another 2000 for those with ω = 1. For the former, all comparisons (100%) correctly represented MATLAB/Simulink SMs that were distinct. For the latter, we found 1863 (93.15%) to correctly represent identical SMs while for the remaining 137 (6.85%), associated SMs were not identical. For these 137 comparisons, we identified all respective 274 SMs to be very small and, thus, to comprise only a few blocks. We found their connections to be identical but rearranged. For example, two SMs, each with Import blocks connecting to a Summation block, but at different interfaces (cf. Sec. II-A).

Given the linear runtime required for their extraction and our manual evaluation on a subset of 4400 comparisons, we consider the CM to be easy to derive from a given system and for such, to exhibit a low probability of mismatch. As a result, we argue that the CM is suitable to abstract MATLAB/Simulink models (and contained SMs), evolved from clone-and-own.

**RQ2: Precision and Recall of our Technique**

SCMA may generate a vast amount of trees, depending on the number of input models and their structural diversity. Furthermore, practitioners can define a similarity threshold to exclude those CM comparisons not exceeding it from the forest creation. Depending on their settings, the forest size can drastically vary as similarity values distribute almost equally (cf. Figure 9).

In Table VI, we provide details about the trees we evaluated for our 18 model comparisons, which is an excerpt from all trees generated by SCMA. We give more information online.

**TABLE VI: Trees Evaluated For Precision & Recall**

<table>
<thead>
<tr>
<th>Tree Size in Terms ofContained Nodes</th>
<th>1-2</th>
<th>2-3</th>
<th>3-4</th>
<th>4-21</th>
<th>20-30</th>
<th>41-44</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compared Model Variants:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-2</td>
<td>10/152</td>
<td>2/2</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-3</td>
<td>10/361</td>
<td>2/2</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-4</td>
<td>10/718</td>
<td>2/2</td>
<td>2/2</td>
<td>2/3</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-5</td>
<td>10/1369</td>
<td>3/5</td>
<td>2/5</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-6</td>
<td>10/1974</td>
<td>3/5</td>
<td>2/7</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-7</td>
<td>10/2985</td>
<td>3/9</td>
<td>2/9</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-8</td>
<td>10/3670</td>
<td>3/9</td>
<td>2/9</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-9</td>
<td>10/4632</td>
<td>3/13</td>
<td>2/13</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-10</td>
<td>10/6143</td>
<td>3/17</td>
<td>2/17</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-11</td>
<td>10/6586</td>
<td>3/17</td>
<td>2/17</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-12</td>
<td>10/7831</td>
<td>3/21</td>
<td>2/21</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-13</td>
<td>10/9763</td>
<td>3/26</td>
<td>2/25</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-14</td>
<td>10/11287</td>
<td>3/32</td>
<td>2/31</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-15</td>
<td>10/13609</td>
<td>3/38</td>
<td>2/37</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-16</td>
<td>10/15381</td>
<td>3/44</td>
<td>1/243</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-17</td>
<td>10/18063</td>
<td>3/50</td>
<td>1/249</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-18</td>
<td>10/20113</td>
<td>3/58</td>
<td>1/257</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>1-19</td>
<td>10/23185</td>
<td>3/66</td>
<td>1/265</td>
<td>1/1</td>
<td>1/1</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>180</td>
<td>49</td>
<td>47</td>
<td>5</td>
<td>34</td>
<td>16</td>
</tr>
</tbody>
</table>

Due to the sheer number of generated trees, we set the similarity threshold for CMs used for the forest creation to ω = 1. In Table VI, we list the model variants included in the comparison on the y-axis. For instance, I-10 refers to the comparison of 10 model variants (cf. Sec. IV-B). Moreover, SCMA generates trees of seven different sizes in terms of the number of comprised nodes and we list them at the top of Table VI. For each of the 18 comparisons and the tree sizes listed in Table VI, we provide the total number of generated trees and the number of manually evaluated trees. For instance, for the comparison I-10, SCMA generated 17 trees of size three, for which we manually assessed three trees regarding precision and recall (3/17).
In Table VI, fields left blank indicate tree sizes not created by SCMA for that specific comparison. Stated at the bottom of Table VI, we manually evaluated a total of 348 trees of various sizes. For instance, we evaluated 180 trees of size one (i.e., comprising only one node) while we evaluated a total of 16 trees with size thirty. We assessed precision and recall directly within the MATLAB/Simulink working environment and we provide more detailed information on our website.7

For precision, all evaluated trees and their comprised nodes respectively, correctly represented a similar SM structure between the compared model variants. Furthermore, CMs contained within such nodes at all times correctly indicated a parent-child relation (cf. Sec. II & Figure 6). In other words, evaluated trees always did reflect a hierarchical structure that was similar between all analyzed models. For recall, we found all trees to be complete and by that, no CM comparison to be erroneously missing. Hence, each tree reflects a similar and complete hierarchical structure between all variants used as input.

Consequently, we argue that our proposed technique is precise and exhibits a high recall for the evaluated comparisons.

**RQ3: Performance of our Technique**

SCMA comprises three sequentially processed phases that determine its overall performance (cf. Sec. III-A-III-C & Figure 4). For the first phase, CM creation, we showed it to be of linear complexity (cf. Figure 8). The second phase, CM comparison, requires all created CMs to be compared with each other. Given their commutativity (cf. Sec. III-B), \( n^2 \) distinct comparisons can be performed for \( n \) CMs. Thus, their comparison depicted in Algorithm 2 exhibits a computational complexity that is quadratic in the total number of CMs:

\[
O(n^2) \quad \text{- Complexity of CM Comparisons}
\]

The third phase, forest creation (cf. Sec. III-C), inserts every generated CM comparison either into an existing tree or utilizes it to found a new tree. To this end, each tree must be assessed and the comprised nodes must be checked for their compatibility (cf. Lines 6-8 in Algorithm 3). In the worst case, a new tree is created for every single CM comparison. For \( n \) CMs, a total of \( \sum_{n=1}^{n-1} n = \frac{n^2-n}{2} \) trees can be constructed and, thus, need to be evaluated. Hence, the computational complexity for the forest creation is quadratic in the number of CM comparisons:

\[
O(n^2) \quad \text{- Complexity of the Forest Creation}
\]

Combining all stated complexities for the three mandatory phases, SCMA exhibits a quadratic computational complexity. Supporting that assessment, Figure 10 illustrates the overall runtime required by SCMA to process a given number of CMs. In particular, we illustrate SCMA’s runtime with regard to the number of used CMs, defined by the similarity threshold \( \omega \). Each data point in Figure 10 represents a model comparison (cf. Table VI). We provide information on the combined size of all models, included in the comparison by the number of blocks, on the x-axis and the respective runtime on the y-axis.

**Threats to Validity**

For the evaluation, we utilized models from the automotive domain only and assessed feasibility using a single study. We developed SCMA independently and prevented ourselves from being biased to that specific domain. The results may not be generalizable, however, we argue that models from the automotive domain are of considerable complexity, which makes us confident that our technique is applicable to other domains as well. Nevertheless, we acknowledge that other domains may exhibit peculiarities we did not consider and which may adversely affect our technique. We furthermore acknowledge that clone-and-own sceneries present in industry may not be fully reflected by the SPES_XT models we evaluated.

We abstract models using descriptors and compare them to calculate a similarity value. Our descriptor only approximates syntactical but not semantical equality. Thus, practitioners might judge differently on the suitability of the descriptor. They may also question the procedure we use to compare them. We argue that we have shown our descriptor to be suitable in a clone-and-own scenario and that we allow practitioners using their domain knowledge to adjust or replace the comparison procedure accordingly to meet their specific demands.

![SCMAs Runtime in Relation to the Number of CMs](image.png)

**Fig. 10: SCMA’s Runtime in Relation to the Number of CMs**

For the largest comparison that includes all 19 model variants, comprising \( \approx 14.300 \) blocks and \( \approx 220.000 \) CMs (cf. Figure 10), SCMA required \( \approx 47 \) minutes. Only considering comparisons with a similarity value of \( \omega = 0.4 \) or greater drastically reduces the runtime to under 5 minutes for all 19 variants. For \( \omega = 0.6 \), SCMA terminates in just \( \approx 2 \) minutes for all 19 variants. The overall runtime distributes as follows. CM creation accounts for \( \approx 3.4\% \), CM comparisons require \( \approx 0.8\% \) and \( \approx 95.8\% \) are needed for the forest creation.

With the analysis and structuring of an entire model family of 19 model variants but without expertise of domain engineers, we conservatively argue that SCMA scales very well, and thus, is applicable to MATLAB/Simulink models of industrial size.

With the similarity threshold set to \( \omega = 1 \), SCMA identifies all identical structures and, thus, type-1 and type-2 clones [46]. However, by adjusting the threshold, SCMA can also capture type-3 and, thus, near-miss clones that contain relocated model fragments and small additions and removals. Details on how SCMA copes with different clone types can be found online.

7. [Link to website](http://example.com)
VI. RELATED WORK

Current model comparison techniques mainly focus on the analysis of just two models [27]. In [47], the authors propose an approach to derive the differences of MATLAB/Simulink models and semantically lift them to ease comprehensibility. While we also aim to reinstate maintainability by enhancing system comprehensibility, the approach proposed in [47] is applicable only to two models. Respectively, model clone detection approaches are proposed in [6] and their applicability is shown. However, our work fundamentally differs from most clone-detection approaches as our technique is specifically designed to cope with multiple models whereas work such as [6], [47] is limited to comparing only two models. Although clone-detection approaches exist that can cope with more [48], applicability is shown only on small case studies and found clones are not aggregated. Our work aggregates similar structures to provide an extensive overview of multiple models while work such as [48] does not. In [1], the authors identify variation points within multiple MATLAB/Simulink models, also using a matrix representation for their approximation. Their approach facilitates on finding cliques of maximum size, an NP-hard problem. Hence, applicability is shown only using small models while our approach is designed to cope with an entire variant portfolio. In [22], the authors identify similarities between Unified Modeling Language (UML) model variants, mapping them to features to allow for their SPL migration. With our work, we do not identify features. The approach that was first proposed in [27] and extended upon by visualizing results in [22], [49], however, indispensably requires domain knowledge to map system artifacts to features. Our approach was motivated by the fact that such domain knowledge is not present in a clone-and-own scenario. In [28], the authors utilize EMF Compare to identify variability by means of the Common Variability Language (CVL). Similar to the approach proposed in [27], domain knowledge is required in the process. Much like [28], authors propose to use CVL in [50] to identify commonalities and differences within model variants for their migration towards an SPL practice. Their work also relies on domain knowledge and the definition of a base model. Our work differs by that no domain knowledge is required upfront and no specific starting point has to be defined. Utilizing CVL to achieve the identification of feature locations with a model family, authors in [51] evaluate different search algorithms. We do not focus on features and, in contrast to [51], do not require domain knowledge upfront. Proposing the ECCO tool in [10], [52] the authors identify, much like our work, similar structures between system variants. Their approach targets source code while our approach targets model-based languages. Furthermore, domain knowledge is required for their analysis. In [53], the authors slice MATLAB/Simulink models. Although they utilize trees for their approach, much like our technique, their work focuses on a single model only. Our approach, however, targets multiple models. In [54], compatibility between MATLAB/Simulink model versions and variants is evaluated. Models are transformed into state machines and compared with each other. However, comparison is only performed on two models at a time and precise domain knowledge is required to determine which model versions and variants are to be compared. Our work focuses on a larger data set and does not require such knowledge. An approach to extractively generate a SPL from a set of system variants is proposed in [55]. Unlike our approach, their work focuses on source code rather than model-based languages. Furthermore, evaluation is done using only three variants while we utilized a portfolio comprising nineteen variants. Assessing source code in [56], domain knowledge is again required for the analysis proposed by the authors. Extending upon NICAD [57], a code clone detection tool, authors leverage it to cope with MATLAB/Simulink models in [58] and introduce SIMONE. Using their textural form rather than a descriptor for comparison, SIMONE, much like our approach, identifies type-3 and, thus, near-miss clones [46] within model subsystems. However, SCMA furthermore aggregates results with respect to hierarchical structures. An approach to compare multiple UML models is proposed in [26]. Classes are compared based on their attributes, assuming them to vary between different models. With MATLAB/Simulink, however, blocks always have the same attributes (i.e. name & function) and it is rather their value that differs. With the computational complexity of their approach, we argue that the work in [26] is not applicable to MATLAB/Simulink.

VII. CONCLUSION AND FUTURE WORK

To adapt to new requirements, clone-and-own, describing the process of copying and modifying existing systems in an unstructured and undocumented fashion, prevails. It eases development efforts in the short-term but unfortunately, results in a proliferation of almost-alike redundant copies of assets with information on their relations rendered incomprehensible. As a result, system maintainability is severely impeded and sustainable development may be at risk. Consequently, it is of superior interest for engineers to regain comprehensibility of the emerged model portfolio. In practice, such models are of considerable complexity and size. Hence, techniques must scale to deal with an entire model family, which is a limitation for current techniques. We address this problem by proposing a new technique for the comparison of an arbitrary number of MATLAB/Simulink models, called Static Connectivity Matrix Analysis (SCMA). In its core, our technique creates a descriptor, called Connectivity Matrix (CM), that abstracts salient information of actual models, and thus, allows for an efficient comparison of an entire model portfolio at large scale. Moreover, based on CMs, we are able to create trees that resemble structural similarities of the original models. Our evaluation shows that creating CMs is fast and that SCMA is precise and scales even for a vast amount of models. Thus, our technique can be applied even to real-world scenarios to recreate information about commonalities across several models.

For future work, we plan to extend our evaluation with further industrial case studies. We also intend to discuss our technique with engineers to identify and tackle limitations they may see. Ultimately, we plan to apply clustering to our results to quantify redundancies within models, and thus, their reduction potential.

ACKNOWLEDGMENTS

Supported by the DFG (German Research Foundation) (SCHA 1635/12-1). We thank Remo Lachmann, Tina Murse and Alexander Knüppel for their strong support and guidance on this paper.


