A Curated Corpus of Simulink Models for Model-based Empirical Studies

Shafiul Azam Chowdhury  
University of Texas at Arlington  
Arlington, Texas

Lina Sera Varghese  
University of Texas at Arlington  
Arlington, Texas

Soumik Mohian  
University of Texas at Arlington  
Arlington, Texas

Taylor T. Johnson  
Vanderbilt University  
Nashville, Tennessee

Christoph Csallner  
University of Texas at Arlington  
Arlington, Texas

ABSTRACT
Recent years have seen many empirical studies of model-based cyber-physical systems and commercial CPS development tool chains such as Matlab/Simulink. To benefit such research, this paper presents the by-far largest corpus of freely available Simulink models to date, containing over 1,000 models.

Surprising findings based on this corpus include that (a) tool support for metric collection is not adequate and (b) users do not reuse model components as they would in object-oriented programs. The paper both confirms and contradicts earlier findings that are based on significantly fewer models, suggesting the utility of the corpus for future research. While others have not yet leveraged this model corpus, we hope that our freely available corpus and infrastructure will benefit future model-based empirical research and tool development efforts, by reducing the model-collection overhead and thus easing evaluation.

CCS CONCEPTS
• Software and its engineering → Model-driven software engineering;

KEYWORDS
Cyber-physical systems, model-based software engineering, empirical study, Simulink

ACM Reference Format:

1 INTRODUCTION
In model-based design of cyber-physical systems (CPS), engineers rapidly prototype their systems using graphical models in sophisticated development environments (e.g., Matlab/Simulink) [9]. Increased usage of such models across various industries (e.g., automotive, aerospace, and industrial automation) has elicited interest in understanding the model properties (e.g., size measures) and how they relate to quality attributes (e.g., complexity and comprehensibility) [5]. Many of these studies investigate structural model properties and propose new metrics based on the properties.

When evaluating various proposed metrics, most of the model-based studies only use a handful of models, which could adversely affect the evaluation. Besides, different studies compute measures using different sets of Simulink models, which makes their comparison problematic. Furthermore, studies often use proprietary models that are not publicly available, which makes reproducing results difficult.

A collection of publicly available models would facilitate evaluation and comparison of many model-based empirical studies. Besides, tools that operate on models (e.g., static analysis, refactoring, and clone detection tools) often require models of sufficiently large size and structural complexity, partly to evaluate scalability [3].

Building a sufficiently large collection of freely available Simulink models is thus vital and incurs non-trivial overhead. However, arbitrarily adding any model in such a collection may be undesirable. For example, studies may only be interested in complex models or in CPS domain-specific (e.g., automotive) models. Similar as for the corpus of Java programs [7], we investigate the various challenges in developing a curated collection of models (aka corpus).

Since such a model corpus is currently unavailable, insights into modeling practices are also unknown. This inspired the SLforge project to build the only-known large-scale collection of public Simulink models [2]. However, the main focus of this earlier work was testing the Simulink tool pipeline automatically.

SLforge collected 391 Simulink models and published their sources, but it did not focus on crafting a corpus. In contrast, we discuss the design challenges for creating a corpus and publish a much larger collection of 1,030 models, along with useful meta information, which would significantly reduce model-collection-overhead in future studies. Models in our corpus are large: 93 models have over 1k blocks, which is greater than the average number of blocks in the models used at Delphi, a large industrial Simulink user [4]. Furthermore, SLforge only studied the model metrics relevant to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SEsCPS’18, May 27–June 3, 2018, Gothenburg, Sweden © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-5728-9/18/05 ... $15.00 https://doi.org/10.1145/3196478.3196484
testing Simulink whereas we investigate interesting modeling practices as well as Simulink model complexity metrics. The corpus and the tools are freely available [1].

2 BACKGROUND

This section provides necessary background information on model-based design using Simulink and next, the most related work.

2.1 CPS Data-flow Models and Simulink

Here, we briefly discuss modeling abstractions in Simulink. A graphical model (of a CPS) consists of blocks, which perform operations on their inputs and pass outputs to other blocks through connection lines. Simulink offers a variety of built-in blocks, organized in libraries and allows establishing implicit or hidden connections using From and Goto blocks [6]. To facilitate custom block-behavior Simulink avails placing native code (e.g., C) using the S-function interface. Besides flat models, Simulink offers hierarchical model creation using Subsystem and Model Reference features, which we collectively call child-model representing blocks.

Figure 1 contains an example hierarchical CPS model [2]. The parent to child model relation is acyclic. But within a model Simulink permits feedback loops (circular data flow). During compilation Simulink may reject a model if it fails to numerically solve feedback loops (aka algebraic loops) using solvers. Simulink offers different simulation modes. While in Normal mode Simulink “only” simulates blocks, it also emits some code for blocks in Accelerator mode. In-depth descriptions of Simulink modeling features are available [9].

2.2 SLforge

Crafting a curated collection of open-source programs is common in most programming domains [7]. However, the only large-scale study of public Simulink models we are aware of is the SLforge project [2]. While SLforge compiles a list of 391 publicly available Simulink models, its main focus is developing Simulink testing techniques. Whereas, we discuss corpus design challenges and publish the redistributable models in a single installation file, to ease replication and comparison of model-based empirical studies. Besides the (redistributable) models, our corpus includes meta information which empirical studies may find useful.

Next, unlike SLforge, we study model metrics relevant to analyzing complexity and modeling practices in general, based on the largest collection of 1,030 publicly available models. While SLforge investigated metrics (i.e., the number of blocks and connections and maximum hierarchy depth in a model and library-usage information) relevant to automated testing, we define and investigate metrics mostly to explore model complexity. Based on our metrics data, we further perform a lightweight comparison with other empirical studies and discuss interesting findings. Furthermore, we extend SLforge’s tools to support the new metrics.

3 CORPUS DESIGN CHOICES

To identify the corpus contents we have used the following criteria, which we expect to be useful for many model-based studies.

- **CPS Domain.** To support various domain-specific studies, we identified the qualitative attribute CPS domain (e.g., Automotive or Avionics) whenever possible for each project, from author-provided descriptions and “tags”.

- **Trivial Models.** Some studies filter out example, toy Simulink models and examine them separately [2]. We included the manually-identified “trivial” models in the Simple group (Section 4.1).

- **Choice of Projects.** Extending SLforge’s model collection [2], we included 96 additional projects (each containing one or more models) from the Matlab Central repository, filtering by highest download count, and 12 projects from the SourceForge public repository.

- **Content Type.** We only include a project in the corpus if it releases models in the mdl or slx formats since these formats are most widely accepted by both engineers and analysis tools. Additionally, when projects distribute code and generated executables we include them as well since studies may choose to analyze them.

- **Test Harnesses and Libraries.** Many projects come with test harnesses and custom libraries, which may themselves be Simulink models. Since studies may choose to analyze them separately, we identified and published their list.

- **Toolbox Requirements.** We extract and include the (mandatory and optional) toolbox requirements information whenever available from the project websites.

4 A STUDY OF MODEL METRICS

Here, we define interesting model metrics and utilizing the corpus, investigate metrics related to complexity and modeling practices. We discuss findings and compare with earlier work.

4.1 Model Groups

Similar to SLforge [2], we group models into four groups: (1) Tutorial (t)—the Simulink proprietary models; (2) Simple (s)—the models we manually filtered out as toy-example ones; (3) Advanced (a)—the non-trivial models; (4) Other (o)—the models we were not able to manually study for time reasons.

4.2 Model Metrics

Mostly using min-max whisker box-plots, we discuss the following model metrics. We have not considered the custom library files (Section 3) as CPS models in this study.
We could not compute solver (Fixed-step (Fixed) and Variable-step (Var)) and simulation mode (Normal (Nor.), External (Ext.), PIL, and Accelerator (Acc.)) information for all of the models.

Table 1: Overview of the collected models: Total number of models (M), models we could compile readily—without installing additional toolboxes (Cm), hierarchical models (H), number of blocks (B), non-hidden connections (C), and all connections (Ch). We could not compute solver (Fixed-step (Fixed) and Variable-step (Var)) and simulation mode (Normal (Nor.), External (Ext.), PIL, and Accelerator (Acc.)) information for all of the models.

<table>
<thead>
<tr>
<th>Group</th>
<th>SLforge</th>
<th>Our Corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M  Cm</td>
<td>H</td>
</tr>
<tr>
<td>t Tutorial</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>s Simple</td>
<td>156</td>
<td>99</td>
</tr>
<tr>
<td>a Advanced</td>
<td>167</td>
<td>66</td>
</tr>
<tr>
<td>o Other</td>
<td>28</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>391</td>
<td>219</td>
</tr>
</tbody>
</table>

Table 2: Most frequently used blocks (besides the top-3 Inport, Outports, and SubSystem), in descending order.

<table>
<thead>
<tr>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>t Product, Constant, Sum, Gain, From, Selector, Mux, Demux, Terminator, Goto, UnitConversion, BusSelector, Fcn, Integrator, Math, Trigonom.</td>
</tr>
<tr>
<td>s Constant, Gain, S_Fun, Terminator, Sum, DataTypeConv., Demux, Mux, Scope, PMIOPort, From, Product, RelationalOp., Goto, Ground, Integr.</td>
</tr>
<tr>
<td>a Constant, From, PMIOPort, Sum, Gain, Goto, Product, Mux, Demux, RelationalOp., Switch, Fcn, SimscapeMulti., PMComp., Ground, Terminator</td>
</tr>
<tr>
<td>o Constant, Gain, Sum, Product, Termin., Mux, S_Fun, Delay, RelationalOp., Demux, From, ZeroOrderHold, DataTypeConv., Integr., Saturate</td>
</tr>
</tbody>
</table>

Figure 2: Model metrics: Total blocks (a), unique blocks (b), connections excluding hidden (c), and including hidden (d).

Figure 3: Metric results: Number of (a) child-model representing blocks, (b) contained subsystems, and (c) strongly-connected components; (d) cyclomatic complexity.

4.2.2 Simulation Complexity. We investigated whether the number of strongly-connected components in the graph representation of a model captures its complexity in terms of numerically simulating it (Figure 3c). While algebraic loops also incur simulation complexity, we found that only 18 models have such loops.

4.2.3 Child-model Representing Blocks. We count child-model representing blocks (Figure 3a) and the number of contained subsystems (NCS) (i.e., the number of blocks in that subsystem) (Figure 3b), as earlier work relates the latter to model complexity [5]. Interestingly, the distribution of the two metrics is almost identical, implying that model-referencing, which is considered as good modeling practice in general, is not widely used in the corpus-models [9].

4.2.4 Hierarchical Modeling. We found that the median number of blocks in a particular hierarchy level does not exceed 17—an observation similar to the “small class phenomenon” in object-oriented (OO) programs [10], where some 57% of the studied Java classes are smaller than 65 LOC on average. One drawback of the small class phenomenon in OO is that much of the conceptual complexity of understanding an OO program (vs. traditional procedural ones) is now in the inter-procedural call and override relationships (vs. the traditional intra-procedural control and data flow within
We found very low (\(n\) where widely used as multiple object instantiation in OO languages. To compare with the findings of a recent model-based study by Olszewska et al., we conduct a pairwise correlation analysis on the maximum hierarchy depth [5]. We compute pairwise Kendall’s \(\tau\) correlation to each other (0.05 significance level).

4.2.5 Child-model Reuse. Simulink allows reusing some Referenced Model in multiple places in the same model, which in OO may be similar to multiple instantiations of a class. However, our study found only one model using this feature, in the Tutorial group. So, at least from our model sample, it appears as if this feature is not as widely used as multiple object instantiation in OO languages. To reuse functionality, do Simulink users instead reuse S-functions? We found very low (< 0.5% median value) S-function reuse rate across all model groups.

4.2.6 MathWorks Cyclomatic Complexity. MathWorks defines an object’s cyclomatic complexity (e.g., of a block) as \(\Sigma_{i=1}^{n} o_i - 1\), where \(n\) is the number of the object’s decision points and \(o_i\) is the number of possible outcomes at the \(i^{th}\) decision point [9]. We adopt the definition as it is widely used [5] (Figure 3d), further noting that the MathWorks tool cannot compute cyclomatic complexity for non-compilable models. Consequently, we could not use the tool for a large portion of the models in the corpus.

4.2.7 Most-used Blocks. In Table 2, we examine the most frequently used 15 blocks, noting that Advanced models more frequently use blocks from the Simscape toolbox (e.g., \(PNIO\) and \(SimscapeBlock\)), which enables rapid creation of complex physical systems [9].

4.2.8 Simulation Configuration. Although these metrics do not relate to complexity, we explore the usage of solvers and simulation modes in Table 1 as these are major configuration options [2].

4.2.9 Others. We compute the number of unique blocks (distinct block-types) in a model and compilation time, primarily to explore whether these metrics correlate with cyclomatic complexity.

4.3 Replicating Earlier Model-based Studies

To compare with the findings of a recent model-based study by Olszewska et al., we conduct a pairwise correlation analysis on the metrics, namely cyclomatic complexity, compilation time, number of blocks and connections, number of child-model representing blocks and NCS, number of strongly connected components and maximum hierarchy depth [5]. We compute pairwise Kendall’s \(r\) mainly to compare with the other study. All the metrics are positively correlated to each other (0.05 significance level).

From our observation, cyclomatic complexity is mostly correlated with the maximum hierarchy depth in a model (0.5509) and the NCS metric (0.5297). In contrast, Olszewska et al. identified NCS as mostly correlated, however, they do not compute correlation with hierarchy depth count and used a single Simulink model in their study whereas we used our full model collection (minus those for which we could not collect all metrics and Simple models) models (listed in [1]).

Other observations diverge from earlier work. For example, earlier work found Matlab Central models to have ten times fewer blocks than industrial models (the latter had some 752 blocks on average) [4]. However, our current collection contains much larger Matlab Central models.

5 CONCLUSIONS

In this work, we present the largest corpus of freely available Simulink models to date. Using the corpus, we explore interesting modeling and complexity metrics. Previously unknown findings in modeling practices and a lightweight evaluation of earlier model complexity study suggest the utility of the corpus in future model-based studies, by reducing the model-collection overhead and supporting evaluation and comparison of such studies. We will endeavor to grow the corpus and investigate metrics to capture model complexity utilizing our infrastructure.

ACKNOWLEDGMENTS

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS 1464311, CNS 1713253, EPCN 1509804, SHF 1527398, and SHF 1736323, the Air Force Research Laboratory (AFRL) through the AFRL’s Visiting Faculty Research Program (VFPR) under contract number FA8750-13-2-0115, as well as contract numbers FA8750-15-1-0105, and FA8650-12-3-7255 via subcontract number WBSC 7255 SOI VU 0001, and the Air Force Office of Scientific Research (AFOSR) through AFOSR’s Summer Faculty Fellowship Program (SFFP) under contract number FA9550-15-F-0001, as well as under contract numbers FA9550-15-1-0258 and FA9550-16-1-0246. The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of AFRL, AFOSR, or NSF.

REFERENCES