SimPact: Impact Analysis for Simulink Models

Eric J. Rapos
Computer Science & Software Engineering
Miami University
Oxford, Ohio, USA
Email: rapose@miamioh.edu

James R. Cordy
School of Computing
Queen's University
Kingston, Ontario, Canada
Email: cordy@cs.queensu.ca

***Abstract*—With the increasing use of Simulink modeling in embedded system development, there comes a need for effective techniques and tools to support managing these models and their related artifacts. Because maintenance of models makes up such a large portion of the cost and effort of the system as a whole, it is increasingly important to ensure that the process of managing models is as simple, intuitive and efficient as possible. Part of model management comes in the form of impact analysis - the ability to determine the impact of a change to a model on related artifacts such as test cases and other models. This paper presents an approach to impact analysis for Simulink models, and a tool to implement it (SimPact). We validate our tool as an impact predictor against the maintenance history of a large set of industrial models and their tests. The results show a high level of both precision and recall in predicting actual impact of model changes on tests.**

I. Introduction

Simulink [3] is a graphical modeling technology used to create models, consisting of blocks, subsystems, lines and embedded code, that are used to simulate real world systems. Simulink is used to model systems for early testing and analysis, and ultimately is capable of generating the source code that will be run on embedded control units (ECUs).

Impact analysis refers to the art and science of determining which software elements may affect one another when changed [4]. Essentially, the goal is to determine the impact of a given change on the artifact and its related artifacts.

Impact analysis is part of the larger are of model management, which refers to practices associated with managing the complexities that come from software modeling. Model-driven engineering (MDE) relies heavily on models as a primary artifact, and brings with them a large number of other issues related to meta-models, model-based tests, model transformations, and multiple types of related models. Maintenance is increasingly challenging when it comes to models, because there are many more artifacts to deal with, including model-based tests, the main focus of our work.

The goal of this work is to produce algorithms and tools for impact analysis on Simulink models that can provide immediate, useful and relevant feedback in an industrial application.

This paper makes the following contributions:

- A method for determining the potential impact of changes to Simulink models on their test cases
- An implementation of the above using static and interactive model analysis to identify potential impact on test cases directly in the test files
- Industrial validation of the above on real-world automotive models provided by our industry partner

II. Related Work

A. Evolving Simulink Models and Tests

It is important to understand the relationship between these artifacts in the Simulink domain. An earlier study [9] highlighted the existence of a co-evolution relationship between Simulink models and their tests (with a correlation value of $r = 0.9, p < 0.01$), which provides the motivation to examine these relationships further using a dedicated impact analysis.

B. Model Management

Work in model management focuses on different combinations of model artifacts, such as managing models and meta-models, or models and their model-based tests. Early exploration of the Software Model Management (SMM) problem was presented by Salay et. al [12]. Their work focuses on the management of large groups of models, and the requirement to merge these models. Particularly, the authors present their Eclipse-based tool framework for SMM. After realizing that manually tracking changes when there is significant overlap of artifacts is increasingly difficult as the number of artifacts increase, MMINT, a graphical tool for interactive model management [13] was created.

One use case of model management addressed model management for regulatory compliance [8]. This work looked at the requirements and regulations set out by governing bodies, and how difficult it can be to conform to sets of overlapping, possibly conflicting, standards. Another application of model management comes in the form of assurance case reuse [7]. This work is somewhat similar to the work presented in this paper, in that both aim to reuse existing model-based artifacts, and provide tool support to do so. Our work aims to extend and expand this previous work in model management to include model test evolution and change impact analysis.

C. Impact Analysis

Impact analysis in source code is a well established area, and in particular has been explored by Rungta et. al. [11], who present an analysis of change impact to infer evolving program behaviours. The change impact analysis in this paper shares a common motivation and is conceptually similar, but focuses on the co-evolution of Simulink models and their tests rather than source code.

In the modeling domain, Briand et. al. have studied impact analysis and change management for UML models [5] and present an automation of this based on UML designs [6]. While similar to our work, the implementations and applications differ; our work focuses on Matlab Simulink models, for which these kinds of impact analysis techniques have not previously been effectively applied.

DiffPlug [1] provides a range of analyses, including visual differencing of Simulink models and a single-level implementation of impact analysis which they call signal tracing. While DiffPlug provides the ability to select a block and trace its signals to explore what-if scenarios, it does so only at the current model level forcing users to traverse the model level-by-level by hand to identify affected target inputs and/or outputs. By contrast, SimPact traverses the entire model hierarchy automatically, tracing and highlighting all potential impact from a given block at all levels. Moreover, DiffPlug functions entirely in a separate custom model editor, whereas SimPact is integrated directly into the native Simulink environment.

Recently Mathworks added the Simulink Design Verifier (SDV) [2] to the Simulink toolset. SDV has a Model Slicer feature that provides a dependency analysis similar to SimPact for single models. The main difference between SDV and our work is that SDV's slicer is restricted to a single model, whereas SimPact provides automatic differencing across consecutive model versions to infer the impact of changes and support model evolution over time.

Early work in Simulink model slicing was done by Reicherdt and Glesner [10], however there is room for expansion of their work to supplement their earlier findings.

III. Simulink Impact Analysis

In this section, we present a two phase process for impact analysis of Simulink model version evolution. The first phase involves model differencing to generate a list of changes between model versions (change isolation), and the second determines the potential impact of those changes on inputs and outputs via recursive propagation of changes.

A. Change Isolation

The first step in determining the impact of changes at the model level on test values is the identification and isolation of the model changes. This process is essentially an exercise in model differencing; a well documented and studied process.

We began with the model differencing script of Rapos and Cordy [9] (which uses the internal Simulink comparison operator). Their algorithm was adapted to produce only a list of handles to Simulink objects in the updated model that had been modified or added, and a list of those in the previous model that had been deleted; both lists are then combined.

B. Determining Impact of Changes on Test Values

Given the list of changes, the potential impact of these changes on test values can then be determined. This is done by iterating over each change, locating it in the model, and propagating its potential impact forward and/or backward as chosen by the user. This propagation is done using a recursive algorithm in each direction that traverses signal lines to find the next blocks, and then recursively finds the impacted blocks from that block, and so on. The exploration ends at any top level signal that corresponds to an input or output of the model, at which point that input or output identifier is added to the list of values that are potentially affected by the change. Figure 1 shows a pseudocode representation of this algorithm. The end result of this analysis is both an $inputList$ and an $outputList$, representing the potentially affected test inputs and outputs.

```
function impactAnalysis(changes)
  for each item in changes
    currentChange = item
    forward(currentChange)
    backward(currentChange)
  end
end impactAnalysis

function forward(currentChange)
  if currentChange is TopLevelOutput
    % there is nowhere left to go forward
    listOfOutputs.add(currentChange)
  else
    for each item in currentChange.outputs
      if not (visited.contains(item))
        visited.add(item)
        forward(item)
      end
    end
  end
end forward

function backward(currentChange)
  if currentChange is TopLevelInput
    % there is nowhere left to go backward
    listOfInputs.add(currentChange)
  else
    for each item in currentChange.inputs
      if not (visited.contains(item))
        visited.add(item)
        backward(item)
      end
    end
  end
end backward
```

Fig. 1. Pseudodode listing for impact analysis

C. Understanding the Results

The first and most important point in understanding the results is that the test inputs and outputs identified are *candidates* for impact. The goal of this work is to automatically determine only the *potential* impact of the changes made between versions on the tests. Further processing of the results is required, and this can be done in one of two ways: (i) the developer can manually inspect, validate and edit the potentially affected test values or (ii) they can use Simulink simulation to generate new candidate values for the potentially affected outputs using existing inputs.

IV. Tool Implementation: SimPact

In this section we present an implementation of our impact algorithms in the tool SimPact (Simulink Impact Analysis). SimPact is implemented as a set of Matlab scripts that run in the Simulink interactive development environment (IDE), invoked either from the Matlab command window or interactively from context menus automatically added to the Simulink interface. We discuss SimPact in the context of two different use cases: static analysis for test maintenance, and interactive analysis for change planning.

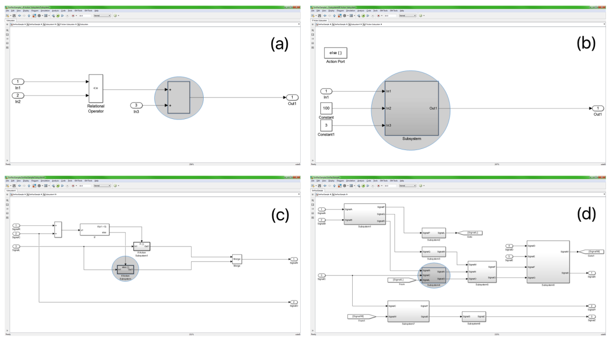
Fig. 2. Tracking Changes Manually

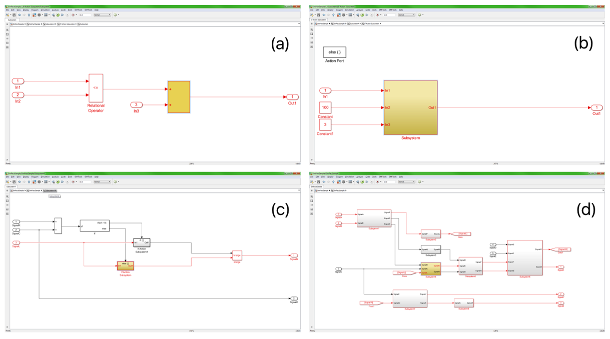
Fig. 3. SimPact Interactive Analysis Plugin highlights impact at all levels

A. Static Analysis: Test Updates

The static analysis application of SimPact works from the Matlab command line. SimPact takes as arguments two Simulink model files, normally two consecutive versions of the same model. SimPact performs the analysis from the previous sections, and yields a list of potentially impacted inputs and outputs. This kind of analysis is useful after a number of updates or changes have been applied to a model and the developer wants to consider the potential impacts of those changes on test inputs and outputs. We can consider this as the batch processing mode of SimPact, identifying potential test impacts of a model update after the fact.

SimPact provides two options for this static analysis: *Highlight Potential Changes in Test File*, and *Generate Suggested Outputs*. For both of these options, in addition to the model files, the previous test case file for the model is required. If the first option is chosen, then in addition to identifying the potentially impacted inputs and outputs in the Simulink IDE, SimPact highlights the columns that contain the potentially impacted values directly in the test case file, and creates a new version of the test case file for manual inspection. The second option takes this one step further, and makes use of existing industry tools to generate suggested output values via simulation for the previous input values.

B. Interactive Analysis: Development Scenarios

The interactive analysis application of SimPact arose from discussions with our industry partner, particularly the desire to select a particular model block, and to see, directly in the Simulink IDE, what the potential impact of a change to that block would be. The need for this kind of *"what if"* analysis comes from the inherent complexity in understanding multi-level hierarchical models developed in Simulink. For example, Figure 2 shows a Simulink system in the top left corner (a). If developers need to know the potential impact of a change the block in the center of the system (a sum block) in order to plan some other calculation, they can easily see the impact at that particular level. However, as they navigate up the model hierarchy (through (b), (c), and (d)) it becomes increasingly difficult to keep track of changes, and almost impossible to enumerate all of the potential impact. In the Figure we have highlighted the source of the embedded change at each level; in practice no such identification is visible in Simulink.

Based on this it was determined that an interactive plugin capable of interacting with Simulink models directly would be the most effective implementation. The actual calculation of the potential impact does not change, aside from the fact there is no need to find the differences; the selected block becomes the one difference. Essentially the interactive analysis uses only the second half of the static analysis. Each of the options (forward, backward, and both) is added to the Simulink context menu for blocks using a custom $sl\_customization.m$ file with the required code to invoke SimPact. This special file is Matlab's method of plugin integration.

To demonstrate this capability, Figure 3 demonstrates how all changes would be highlighted in the same use case explained previously for Figure 2. The selected block, and any parents containing it, are colored yellow, while all potentially impacted blocks are outlined in red.

V. Validation

As SimPact is essentially a prediction tool, our validation of its effectiveness was conducted by comparing its predictions against the ground truth of observed test case changes between historical versions of industrial models. This section details the experimental design and observed results of our validation, and a discussion of the findings.

A. Model Set

We were very fortunate to have access to a production application consisting of a large set of Simulink automotive domain Models and Tests over several releases from our industry partner. They are structured into nine main components (referred to as *rings*), each of which is made up of a number of sub-components. In total there are 55 sub-component models and 9 ring integration models, for a total of 64 different models in the system (not all of which exist in all of the releases).

In total there are 15 releases of the application from a version control system (VCS), however there are several releases during which there were no changes and no versions added to the VCS (releases 4, 5, 9, 12, and 14). Since those releases involved no evolution, for our experiment they were removed entirely, along with a 6th release removed for similar reasons; this left 9 releases to examine. The term "release" in the context of this work does not actually refer to a final production version of the system. All of the versions examined are actually pre-release. In the context of our industry partner, a release refers to a milestone in development; every incremental time block, the release number is increased.

For each of the models, there is an associated test suite provided along with the models. For the purposes of this application, each test suite is contained in an Excel workbook comprised of a number of Excel spreadsheets (using the tabs in Excel), each containing test inputs and expected outputs for the given model over a number of time steps. Each row represents a time step, and each column is an input value or an expected output value. The tests are run by simulation in Simulink, and the results compared with the expected outputs.

B. Experimental Design

In order to observe the effectiveness of SimPact's prediction abilities, a comparison against a ground truth is necessary. For each of the 8 of the evolution steps (i.e. the move from one version to the next e.g. release 13 to release 15), a four step process was used.

(1) SimPact for Change Prediction - For every pair of consecutive model versions, the static analysis implementation of SimPact was run to determine what the potential impact on the test cases would be. The result of this step for each pair was a list of potentially impacted inputs and outputs.

(2) Observing Actual Test Changes - A test differencing script was developed to identify test changes. For each difference observed the change is noted. The final output is a list of test inputs and outputs that have been changed between the two versions. These actual changes form the ground truth to be compared against.

(3) Calculation of Precision and Recall - Given the predicted potential change and actual observed change, precision and recall were calculated using the formulas below. Since different developers may have different use cases, and since SimPact can predict both forward and backward impact, precision and recall were calculated separately for inputs and outputs. The following definitions and calculations were used for this step of the experiment:

- *True Positive (TP)* - Test values that were predicted by SimPact that actually changed
- *False Positive (FP)* - Test values that were predicted by SimPact that did not actually change
- *False Negative (FN)* - Test values that were not predicted by SimPact that actually changed
- *True Negative (TN)* - Test values that were not predicted by SimPact that did not actually change

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

(4) Combining Precision and Recall Metrics - Because of the high number of 4-tuples (input precision, input recall, output precision, and input recall) obtained from the experiment, and the lack of a single performance score, it was difficult to discern the effectiveness of SimPact. Therefore average precision and recall were calculated for every model in each of the evolution steps, leaving only eight 4-tuples of results (one for each evolution step). An average of these eight averages was calculated to provide precision and recall for input and output over the entire model set. Finally, the precision and recall values were combined into an F-measure to provide their harmonic mean using the formula below.

$$F - measure = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

C. Results

The results of the validation experiment to evaluate the precision and recall of SimPact in test change prediction are shown in Table I. Overall the results are very promising, but with a few areas for improvement, or rather discussion.

The overall performance of the tool is dominated by three particular evolution steps: releases 1-3, releases 7-8, and releases 10-11. While these three results are the worst performing, these evolution steps were identified by our industry partner as those just prior to a major release, thus there is an increased number of changes occurring in these evolution steps. If these comparisons are removed from the results, the F-measures for input and output increase to 0.93 and 0.95 respectively. From this, it is possible to determine that SimPact works more effectively at a finer grain, that is, on smaller amounts of changes. This actually better aligns the tool, as smaller increments are the intended use case. A more common usage would likely be on a day to day basis to determine the impacts of that day's changes, rather than at the end of the release, capturing seven weeks of changes.

When performing a prediction experiment, typical areas of concern are false positives and false negatives. When it comes to false positives, these are inputs and outputs predicted to potentially require change by SimPact which did not actually change. Previous work in Simulink evolution has shown that there are some instances of model change that do not require changes in the test cases [9]; these form part of the collection of false positives. Further to this, from the perspective of software quality, a small number of false positives is of relatively low concern, as they simply mean that additional scrutiny will be used in examining the effect of changes.

False negatives, which are the changes made to the test values that SimPact was not able to predict the need for, are of slightly more concern. These cases could potentially mean

TABLE I
SimPact Precision and Recall for Predicted Impact on Test Case Inputs and Outputs

	Inputs		**Outputs**	
	Prec.	**Recall**	**Prec.**	**Recall**
release 1-3	0.66	0.67	0.58	0.65
release 3-6	0.91	0.91	0.85	0.85
release 6-7	0.99	0.98	0.99	0.97
release 7-8	0.76	0.83	0.61	0.62
release 8-10	0.89	0.90	0.85	0.85
release 10-11	0.80	0.81	0.84	0.85
release 11-13	0.98	1.00	0.98	1.00
release 13-15	1.00	1.00	0.96	0.95
Average	**0.87**	**0.89**	**0.83**	**0.84**

F-Measure, Inputs	**0.88**
F-Measure, Outputs	**0.84**

that some necessary changes to test cases may be overlooked when using SimPact. While it has yet to be confirmed, from an examination of the cases of poor performance and the existence of false negatives, it appears that the missed changes in output values may be caused by changes to the input values in the tests. Essentially the change in the test output values was not due to the change in the model, but rather to an independent change in the test inputs, something SimPact does not have access to during its prediction.

VI. Future Work

One of the shortcomings of SimPact is still that it provides *potential* impact, and *suggested* test values using simulation. One possible area of future work is to increase the confidence in its ability to specifically predict exact impact, and to provide test case values based on more informed local domain knowledge rather than simulation. These improvements will require significant work.

Additionally, since our work on impact analysis is similar to software slicing, further work in the area of model slicing may be possible. Currently SimPact is capable of producing forward and backward slices of a model from a given point, and even both slices together, but it is of interest to extend it to be able to compute the intersection of slices, in order to determine which sections of the model are co-relevant to two (or more) blocks in the model. A use-case for this function would be selecting a single input and and single output and determining the intersection of the forward slice of the input and the backward slice of the output to identify the components of the model that are part of both, allowing for a single snapshot of the system.

One possible area of expansion is the integration of SimPact with a version control system in order avoid the necessity of having both versions of the model on the developer's workstation. Access to the version control system would allow SimPact to check a newly checked-in version directly against the previous version. Another possibiity is the provision of an interactive review of potential test case changes. Currently, these are either marked in the test file or generated via simulation, then left for the developer to edit by hand. It may be useful to have the developer see the changes as they are suggested and either approve or reject them interactively. No doubt many other tool support ideas will emerge over time as SimPact is used, from both the research perspective and from experiential feedback from our industry partners.

VII. Summary & Conclusion

SimPact is able to accurately trace impact of changes to any given model block through the rest of a model, traversing the hierarchy to the top level inputs and outputs, thus identifying potential impact on test case values. This was implemented in both an interactive and static analysis tool. Static analysis works on batch changes between two versions of the same model, and interactive analysis works directly in an open Simulink model, accessible from the block context menu.

SimPact was validated by calculating precision, recall and F-measure of its ability to predict changes in inputs and outputs of actual industrial test cases. F-measures for input and output test change predication were calculated to be 0.87 and 0.84 respectively (and with the removal of non-standard use cases, 0.95 and 0.93 respectively). As a prediction tool SimPact provides promising results contributing to the goal of supporting Simulink model evolution.

Acknowledgments

The authors would like to thank our industry partners for their continued support in providing valuable feedback and testing our tool in their environment. Specifically we would like to thank Costantino Rotella for his time and assistance. The authors would also acknowledge that this work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), as part of the NECSIS Automotive Research Partnership with General Motors, IBM Canada and Malina Software Corp., and by the Ontario Ministry of Research, Innovation and Science through an Ontario Research Excellence grant.

References

[1] Diffplug unleashes Simulink's potential. https://www.diffplug.com/features/simulink. Accessed: 2017-02-06.

[2] MathWorks Simulink Design Verifier. https://www.mathworks.com/products/sldesignverifier/features.html. Accessed: 2017-03-24.

[3] MathWorks Simulink Product Page. http://www.mathworks.com/products/simulink/. Accessed: 2015-10-27.

[4] R.S. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[5] L.C. Briand, Y. Labiche, and L. O'Sullivan. Impact analysis and change management of uml models. In *Proceedings of the International Conference on Software Maintenance, 2003 (ICSM 2003)*, pages 256–265, Sept 2003.

[6] L.C. Briand, Y. Labiche, and G. Soccar. Automating impact analysis and regression test selection based on UML designs. In *Proceedings of the International Conference on Software Maintenance (ICSM 2002)*, ICSM '02, pages 252 – 261, 2002.

[7] S. Kokaly, R. Salay, V. Cassano, T. Maibaum, and M. Chechik. A model management approach for assurance case reuse due to system evolution. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, pages 196–206, New York, NY, USA, 2016. ACM.

[8] S. Kokaly, R. Salay, M. Sabetzadeh, M. Chechik, and T. Maibaum. Model management for regulatory compliance: A position paper. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering*, MiSE '16, pages 74–80, New York, NY, USA, 2016. ACM.

[9] E.J. Rapos and J.R. Cordy. Examining the co-evolution relationship between Simulink models and their test cases. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering*, MiSE '16, pages 34–40, New York, NY, USA, 2016. ACM.

[10] Robert Reicherdt and Sabine Glesner. Slicing matlab simulink models. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 551–561, Piscataway, NJ, USA, 2012. IEEE Press.

[11] N. Rungta, S. Person, and J. Branchaud. A change impact analysis to characterize evolving program behaviors. In *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM 2012)*, pages 109–118, Sept 2012.

[12] R. Salay, M. Chechik, S. Easterbrook, Z. Diskin, P. McCormick, S. Nejati, M. Sabetzadeh, and P. Viriyakattiyaporn. An eclipse-based tool framework for software model management. In *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, eclipse '07, pages 55–59, New York, NY, USA, 2007. ACM.

[13] A. Di Sandro, R. Salay, M. Famelis, S. Kokaly, and M. Chechik. Mmint: A graphical tool for interactive model management. In *Proceedings of the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), Demo and Poster Session*, pages 16–19, 2015.