Noname manuscript No.
(will be inserted by the editor)

Clone Detection in Matlab Stateflow Models

Jian Chen · Thomas R. Dean · Manar H. Alalfi

Received: date / Accepted: date

Abstract Matlab Simulink is one of the leading tools for model based software development in the automotive industry. One extension to Simulink is Stateflow, which allows the user to embed Statecharts as components in a Simulink Model. These state machines contain nested states, an action language that describes events, guards, conditions and actions and complex transitions. As Stateflow has become increasingly important in Simulink models for the automotive sector, we extend previous work on clone detection of Simulink models to Stateflow components.

While Stateflow models are stored in the same file as the Simulink models that host them, the representations differ. Our approach incorporates a pre-transformation that converts the Stateflow models into a form that allows us to use the SIMONE model clone detector to identify candidates and cluster them into classes. In addition, we push the results of the Stateflow clone detection back into the Simulink models, improving the accuracy of the clones found in the host Simulink models.

We validated our approach on the Matlab Simulink/Stateflow Demo set. Our approach showed promising results on the identification of Stateflow clones in isolation, as well as integrated components of the Simulink models that are hosting them.

Keywords Model · State Machine · Stateflow

J. Chen
School of Computing, Queen's University
Kingston, Canada E-mail: chenj@cs.queensu.ca

T. Dean
Electrical and Computer Engineering, Queen's University
Kingston, Canada E-mail: dean@cs.queesnu.ca

M. Alalfi
School of Computing, Queen's University
Kingston, Canada E-mail: alalfi@cs.queesnu.ca

1 Introduction

Model Driven Engineering (MDE) has become popular in industry as a way to build and maintain complex modern software. It is increasingly common to develop software using model-based methodology in embedded software, particularly in areas where risk to life or property is an issue such as the automotive sector. Simulink[1] is a modeling language that has been widely used in the development of automotive embedded systems. One component of Simulink is Stateflow[2], an environment for modeling and simulating combinatorial and sequential decision logic based on hierarchical state machines (i.e. state charts (Harel, 1987)) and flow charts. Developers use MathWorks Simulink/Stateflow and model-based design to develop system models, verify designs using stimulation, and generate production codes in industry.

In code-based software development, the simple reuse of code segments by copy & paste can cause code clones. Koschke (Koschke, 2006) detailed the root causes of code clones, and Roy and Cordy(Roy and Cordy, 2007) classified these reasons into four categories: development strategy, maintenance benefits, overcoming underlying limitations, and cloning by accident.

Similarly, in model-driven development environments, developers copy parts of existing models and reuse them in new models resulting in model clones, as has been observed in Simulilnk model development (Cordy, 2013). The potential impact of identifying redundancy at the higher levels of abstraction provided by models makes clone detection in models important since it can help in testing design consistency and completeness before implementation. In this paper, we describe extensions to SIMONE (Alalfi et al, 2012a) to detect near-miss clones in Stateflow components.

This work is part of model pattern engineering project, one of Network on Engineering Complex Software Intensive Systems for Automotive System (NECSIS)[3] projects, the purpose of which is discovery, cataloguing, and formalization of sub-model patterns in automotive models. Clone detection can provide an initial approximation to the pattern set. Our work is in phase one of the larger project, and the goal is to discover and identify common sub-model patterns in a lager example model set obtained from our industrial partners at General Motors.

This is an extended version of a paper presented at the International Workshop on Software Clones 2014 (Chen et al, 2014). In addition to the description of clone detection in Stateflow models, this paper extends our previous paper in several ways. We refine and present a definition of model clones for Stateflow similar to the previous definition we used for Simulink in SIMONE. We show how the results of Stateflow clone detection can be used to improve the detection of clones in the host Simulink models. Both detection and contextualization were validated using the MATLAB demo set.

[1] www.mathworks.com/products/simulink
[2] www.mathworks.com/products/stateflow
[3] www.necsis.ca

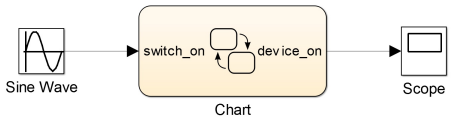


Fig. 1 A simple example of a Stateflow block within a Simulink model

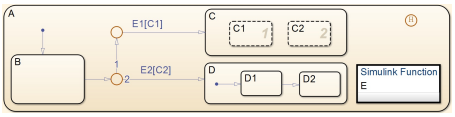


Fig. 2 A simple Stateflow chart example. The chart is the parent of the state A, state A is the parent of the B, C, and D states. Exclusive state *D1* and *D2*. Parallel state *C1* and *C2*. The chart also includes a history junction and a Simulink function, and transitions junctions

2 Background

This section gives an overview of some of the background needed to understand our work. We begin by discussing Matlab/Stateflow models followed by an overview of SIMONE, a near-miss model clone detector used in our research, then we present our definition of model clones in Stateflow.

2.1 Stateflow Models

MATLAB[4] is an interactive computing environment and high level programming language. Simulink is an extension to MATLAB that provides a block diagram environment for designing, modeling, and simulating reactive systems. Stateflow is an extension to Simulink that provides an environment for modeling and simulating combinatorial and sequential decision logic based on hierarchical state machines (i.e. state charts (Harel, 1987)) and flow charts. Figure 1 shows a simple example of Simulink model that consists of a Sine Wave block, a Scope block, and a single Stateflow block.

A set of graphical and nongraphical objects and the relationships between those objects form a typical Stateflow chart. Each Stateflow block is an equivalent of Stateflow chart in Simulink model. Figure 2 shows a simple Stateflow chart model. The blocks in this example do not have an explicit purpose; they only present a typical Stateflow model's structure. A state can use a hierarchy to express more complex models. The example in Figure 2 shows three levels of hierarchy. State A is an outer state(or superstate); state B is a substate(or child). State A is the parent of the state B, C, and D. Every superstate has a decomposition, which can be exclusive(state *D1* and *D2*) or parallel(state *C1* and *C2*). When state C is active, then both state *C1* and *C2* are active. If state D is active, then either state *D1* or *D2* is active.

In Matlab, all Simulink and Stateflow models are stored in model files with the .mdl extension. A model file is a structured text file that contains blocks

[4] www.mathworks.com/products/matlab

```
Model {
  Name    "powerwindow"
  ...
  Simulink.ConfigSet {
    $PropName      "ActiveConfigurationSet"
    $ObjectID      1
  }
  BlockDefaults {
    ForegroundColor   "black"
    ...
  }
  AnnotationDefaults {
    HorizontalAlignment   "center"
    ...
  }
  System {
    Name     "powerwindow"
    ...
    Block{
        BlockType SubSystem
        Name  "control"
        ...
        System{
            Name "control"
            ...
            Block{
                BlockType S-Function
                ...
                Tag    "Stateflow S-Function powerwindow 1"
                ...
            }
        }
    }
    ...
  }
}
# Finite State Machines
#
#    Stateflow Version 7.6 (R2011b) dated Jul 8 2011, 18:16:10
#

Stateflow {
  machine {
    id      1
    name     "powerwindow"
    ...
  }
  chart {
 ...
  }
  state{
   ...
  }
  state{
   ...
  }
  ...
}
```

Fig. 3 Snippet of the textual representation of the MATLAB power window demo model

of text nested in braces with *key-value* pairs that describe the properties of the model. The file starts with the Simulink model where all elements are combined into a single Model section and stored in hierarchical order. Figure 3 shows an example of MDL file textual representation. The *Model* section includes all the Simulink model elements and the model parameters, configuration set, and configurations references. The *BlockDefaults* section includes the default settings for all blocks in the model. The *AnnotationDefaults* section includes default settings for annotations in the model. The *System* section includes parameters that describe each system and subsystem in the model. The subsystem is nested in the parent system. Each system section contains blocks, lines, and annotations. Each Stateflow model is represented as a single block in the Simulink model textual representation.

The Stateflow models are stored at the end of the same file, and all Stateflow elements form a single Stateflow block with linear structure. The textual representation of Stateflow model does not follow the Simulink subsystem representation. That is, the description of sub states are not nested within the description of super states. Figure 4 shows an excerpt of the textual representation of a Stateflow model, which in turn is embedded in a Simulink model file.

2.1.1 SIMONE

SIMONE(Alalfi et al, 2012a) is a model clone detector designed for detecting near-miss submodel clones in Simulink models. SIMONE extends the code-based clone detector NiCad (Cordy and Roy, 2011). NiCad is a text comparison software clone detection system with a plugin architecture that uses TXL(Cordy, 2006). NiCad has been successfully used for finding cloned codes in a range of languages, including C, Java, Python, C#, and WSDL(Martin and Cordy, 2010). SIMONE extends the NiCad code clone detector engine to analyze the internal textual representation of Simulink MDL files.

SIMONE extracts all potential clones at a specified level of granularity from the Simulink MDL files as clone candidates. Then SIMONE normalizes the clone candidates by filtering out some attributes such as layout, canonically sorting blocks, and renaming attribute values to eliminate any unwanted differences to make the comparison process more precise and accurate. Last, SIMONE compares the clone candidates line-by-line using the Longest Common Subsequence algorithm(Hirschberg, 1977). Then SIMONE computes a percentage of unique items for each potential clone and use the number of unique lines in each as a measure of similarity. If the percentage of unique items in both line sequences of potential clones is below a given threshold, the pair is considered to be clones.

```
Stateflow {
  machine {
    id       1
    name      "powerwindow"
    ...
  }
  chart {
    id       2
    name      "control"
    windowPosition    [24 266 702 602]
    viewLimits      [0 843.043 2.915 444.795]
    zoomFactor      1.282
    screen      [1 1 1280 1024 1.041666666666667]
    treeNode          [0 22 0 0]
    ...
  }
  state {
    id       3
    labelString      "passengerneutral\nentry:\nmoveUp = 0;\nmoveDown = 0;"
    position      [724.059 27.423 98.524 90.095]
    fontSize      12
    ...
    treeNode              [15 0 0 6]
    ...
  }
  ...
  junction {
    id       23
    ...
    linkNode              [5 0 0]
    ...
  }
  transition {
    id       24
    labelString     "after(100,ticks)"
    src { ... }
    dst { ... }
    ...
    linkNode              [5 0 25]
    ...
   }
 ...
}
```

Fig. 4 Snippet of the Stateflow textual representation in the MATLAB power window demo model

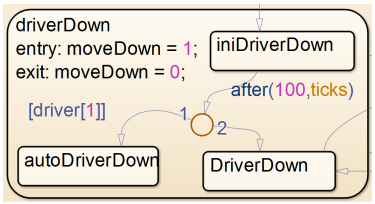
2.2 Stateflow TXL Grammar

In order to use the existing clone detection tool SIMONE, the first step is to build a Stateflow TXL grammar allowing TXL to parse Stateflow models. We derive a TXL grammar from a large set of Stateflow model examples in the public domain by using iterative grammar techniques(Stevenson and Cordy, 2012). Our grammar identifies all observed elements of the Stateflow models, including machines, charts, states, transitions, junctions, events, data, instances, targets and other elements.

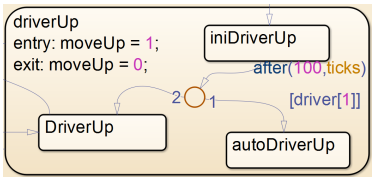
(a) Hydraulic Monitor

(b) Position Monitor

Fig. 5 Example of a type 1(exact) clone in a Stateflow model. Both (a) and (b) in the sf_aircraft_screen_library model and they contain the same number of states and transitions. The two *failure_logging* states can be considered as the cloned fragments.



(a) driverDown Substate



(b) driverUpn Substate

Fig. 6 Example of a type 2(renamed) clone in a Stateflow model. Both are in the power-window model.

2.3 Clones in Stateflow Models

Software clones are segments of code that are similar according to some definition of similarity (Koschke, 2006). Model clones can be defined similarly as the way of code clone is defined. Model clones are similar or identical fragments in software models. However, this is a rather vague definition, because the possible ways in which fragments can be identified in models. Generally speaking, models are typically represented by graphs. Model clones are similar subgraphs of these graphs(Alalfi et al, 2012a).

We adapt the definition of clones we used in Simulink clone detection(Alalfi et al, 2012a) for Stateflow model clones. The graph elements(states, transitions and junctions etc.) are the fragments of a Stateflow model. For our purposes, clones in Stateflow are models that are structurally similar. For example, the same structure states and transitions with different labels, conditions and ac-

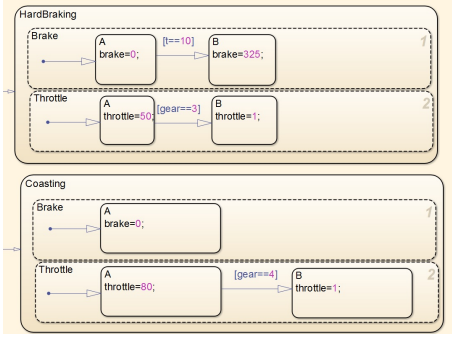


Fig. 7 Example of a type 3(near-miss) clone in sf_test_vectors model.

tions are considered clones. Our research group has categorized the three model clone types, and we tailored them here for Stateflow as follows:

- Type 1 (exact) model clones are identical model fragments, ignoring variations in visual presentation, layout, and formatting, Figure 5 shows an example of type 1 clones where all Stateflow components are exactly the same. In general, the location and layout of the Stateflow elements may change and still remain a type 1 clone.
- Type 2 (renamed) model clones are structurally identical model fragments, ignoring variations in labels, values, types, and the variations from Type 1. Figure 6 shows an example of Type 2 (renamed) model clone. The figure shows two states from the MATLAB power window demo model. The two states contain nested states, junctions an transitions that are identical in structure, and in some cases, identical in labels. The differences are the names of the states, the names of the substates and some of the labels.
- Type 3 (near-miss) model clones are model fragments with further modifications such as small additions or removals of model elements such as charts, states, translations, junctions, events, data, instances, targets and other elements, in addition to the variations from Type 1 and 2 clones. Figure 7 shows an example of Type 3 near miss model clone. These also represent complex substates of a state model, however in addition to changes in labels, the second element of the clone class has fewer substates and transitions than the first.

3 Approach

Figure 8 shows the three stages of our approach. The first stage transforms the Stateflow textual representation into a hierarchical textual structure as the initial input. The second stage, implemented as a plugin, normalizes the initial input to remove irrelevant elements and rename irrelevant naming differences to make the process of clone identification more accurate. The final stage identifies potential clone candidates and clusters them into classes.

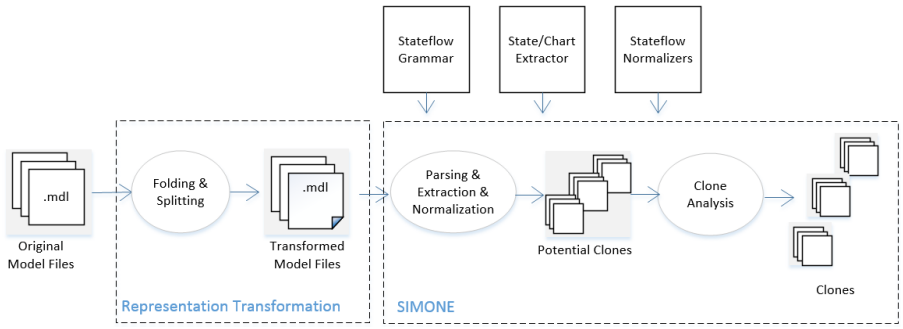


Fig. 8 Steps of our approach

After Stateflow model clone identification, we also add a contextualization step in order to improve the accuracy of Simulink model clone detection by enabling the identification of model clones in Simulink models that have Stateflow components embedded in them.

4 Representation Transformation

In the previous section, we gave an overview of the Stateflow model representation at the text level, all *states*, *junctions*, *transitions* and other elements are sequentially stored in the Simulink model file. This linear text representation of each object is the first challenge for Stateflow model clone detection. In this section, we show how to transform the liner representation of the model into a hierarchical version and separate the actions of a state to represent each as separate attributes.

4.1 Structure Folding

We restructure the textual representation of Stateflow to explicitly show the Stateflow hierarchy(The MathWorks Inc, 2014), which folds each object to its parent object to form a nested textual presentation, called *Folding*. The purpose of folding phase is to bring all the elements referenced by the attributes into a self contained unit so that all related elements can be extracted as one potential clone fragment by the extractor. This step is similar to (Martin and Cordy, 2011) and (Antony et al, 2013). Martin et al. presented the technique for extracting the elements of each operation in WSDL(Web service Description Language) and consolidating them into a self-contained unit. Antony et al. applied this technique to the XMI text representation to reveal the hidden hierarchical structure of the model and granularize behavioural interactions into conversational units.

Figure 9 shows an example from the MATLAB demo set. The *treeNode* and *linkNode* attributes are used to preserve the hierarchical structure of Stateflow models (Dominguez, 2012). These attributes contain references to other

```
Stateflow {
  machine {
    id       1
    name     "powerwindow"
    ...
  chart {
    id       2
    name     "control"
    ...
    treeNode          [0 22 0 0]
    ...
  }
  ...
  state {
    id       4
    labelString     "emergencyDown\nentry:\nmoveUp = 0;\nmoveDown = 1;"
    ...
    treeNode     [2 0 22 0]
    ...
  }
  state {
    id       15
    labelString     "driverNeutral\nentry:\nmoveUp = 0;\nmoveDown = 0;"
    ...
    treeNode     [22 3 0 14]
    ...
  }
  state {
    id       22
    labelString     "safe"
    ...
    treeNode     [2 15 0 4]
    ...
  }
  ...
  junction {
    id       23
    ...
    linkNode     [6 0 0]
    ...
  }
  ...
  transition {
    id       29
    labelString     "[obstacle]"
    ...
    src {
      id       22
      intersection      [2 1 0 0.34 713.8395 156.0148 0 83.5443]
    }
    dst {
      id       4
      intersection      [3 0 1 0.6093 762.5552 117.5173 0 -83.5437]
    }
    ...
    linkNode     [2 59 0]
    ...
  }
 ...
}
```

Fig. 9 Textual Representation of MATLAB powerwindow model chart (used in MATLAB demo set)

Stateflow elements and the *treeNode* element is used to represent the tree structure of components that can contain other components(i.e. charts and states), while the *linkNode* attribute represents primitive elements such as transitions and junctions. We create a transformation to fold each child object into its parent object. The program folds the textual block of any Stateflow objects such as substates, junctions and transitions and put them in the parent block. The transformation takes a folding approach that examines each element in turn and inserts it into the appropriate parent element. At the same time, the nested elements are sorted by type: first states, then transitions, and finally junctions.

```
Stateflow {
    machine {
        id 1
        name "powerwindow"
   ...
        chart {
            id 2
            name "control"
            ...
       state {
                id 4
                labelString "emergencyDown\nentry:\nmoveUp = 0;\nmoveDown = 1;"
            }
            state {
                id 22
                labelString "safe"
                ...
                state {
                    id 13
                    labelString "driverUp\nentry: moveUp = 1;\nexit: moveUp = 0;"
                    ...
                }
             ...
             transition {
                id 29
                labelString "[obstacle]"
                ...
                src {
                    id 22
                    intersection [2 1 0 0.34 713.8395 156.0148 0 83.5443]
                }
                dst {
                    id 4
                    intersection [3 0 1 0.6093 762.5552 117.5173 0 - 83.5437]
                }
  ...
            }
 ...
}
```

Fig. 10 Folding textual presentation result of powerwindow model

Figure 10 shows a simplified version of the new representation. In the figure, every Stateflow object has been folded into its parent object properly, it clearly shows the nested structure.

4.2 Label Splitting

The *labelString* is an important attribute for *states* and *transitions*. Some of Stateflow properties such as state names, actions and transition conditions are encoded as a single string of *labelString* attribute. The *labelString* in a state has a general format shown in Listing 1. The first line is the name of a state, and the following lines are a set of actions after each keywords: *entry*, *during* and *exit*. These actions are executed at the different phase of a state, i.e.,entry actions are at the activation of a state; during actions are at the simulation phase; and the exit actions are executed when a state is going to be deactivated.

Listing 1 State labels general format

```
name
entry: entry actions
during: during actions
exit: exit actionsn envent_name: on event_name actions
bind: events
```

The *labelString* of transition has different syntax and semantics. Listing 2 shows a general format of transition label. Transition labels contain event triggers, conditions, condition actions and transition actions.

Listing 2 Transition labels general format

```
event[condition]{condition_action}/transition_action
```

Our model clone detection tool is based on comparing line as a whole. So a difference in a single part of a state or transition label renders the entire line different. We separate the single *lableString* into multiple lines to improve the precision of our model clone detection. We introduce new attributes and split the state labels into several new attributes and its own value. The state name, if present, is encoded using a new *textlabel* attribute. The entry, during and exit actions, when present, are encoded using separate attributes of similar names (*entrylabel*, *duringlabel*, and *exitlabel*).

Transition label has different components with the state labels, so we introduce four new attributes to the textual presentation. We separate each of the components of the transition labels into separate attributes. These components are identified by the new attributes *eventlabel*, *conditionlabel*, *condition action*, and *actionlabel*. This provides us with finer grained control over the comparisons used for clone detection. For example we can distinguish between a change in an event label from a change to both an event label and the code given by the action of the transition.

5 Extraction And Normalization

Generally, clone detection tools are hunting for fragments of codes or models to compare as clone candidates, which are extracted from their representation. To extract Stateflow model fragments, we add a new extractor to SIMONE.

After the extraction, the extracted fragments can be normalized to improve the precision and recall of the clone detection phase. In this section, we discuss the transformations that provide the initial results of our model clone detection.

5.1 Extraction

Identifying and extracting the potential clones is the first stage of clone detection. We could use the entire *Stateflow* section of the file, but that would not provide comparison of fragments at the state level. Two similar states might be ignored, and only the highest level state machines are compared. To achieve a finer level of comparison fragment, we need to define the granularity for Stateflow.

5.1.1 Granularity

We provide two granularities of clone candidates for Stateflow. The first, *chart granularity* extracts all of the Stateflow charts as clone candidates. Charts in Stateflow represent entire machines. A Simulink model may have more than one chart, each of which may be instantiated multiple times as blocks in the Simulink Model. The second level of granularity, *state granularity*, extracts all states in all charts as clone candidates. This allows us to identify cloned state machines that are nested within states.

5.2 Normalization

In this phase, three new transforms are implemented in TXL for both states and charts to normalize the result of the model files from the previous extraction step. The three transforms are filtering, renaming and sorting.

5.2.1 Filtering

Listing 3 shows an example of the extracted textual representation for Stateflow with all elements in chart, state, transition and junction. There exist a number of elements (*windowPosition*, *viewLimits*, *position*, *fontSize*) related to layout and formatting, which have no meaning from the model cloning point of view. Even a small change in an element such as font or position can make identical model fragments look very different when compared in the textual representation level and prevent SIMONE from finding them as clones. In order to avoid irrelevant differences overwhelming the similarities in fragments of models, we designed a filtering plugin to identify and remove irrelevant elements from extracted fragment potential clones. Due to the lack of definitive documentation for the text form of Stateflow model files, we gradually tune our filters to remove irrelevant attributes as they are discovered. In the end, our filtering transformation removes ten elements at the state level and seventeen elements at the charts level to reduce the representation of model.

Listing 3 Example snippet of the extracted fragment used by Stateflow to store graphical models.

```
chart {
          id 2
          name "control"
          windowPosition [26.88 228.48 713.28 391.68]
          viewLimits [0 843.043 2.915 444.795]
          zoomFactor 1.282
          screen [1 1 1024 768 1.041666666666667]
          treeNode [0 22 0 0]
          firstTransition 28
          viewObj 2
          machine 1
          ssIdHighWaterMark 64
          decomposition CLUSTER_CHART
          firstEvent 60
          firstData 61
          chartFileNumber 1
          executeAtInitialization 1
          supportVariableSizing 0
          state {
              id 4
              labelString "emergencyDown\nentry:\nmoveUp = 0;\nmoveDown = 1;"
              entrylabel "entry:"
              entrylabel "moveUp = 0;"
              entrylabel "moveDown = 1;"
              textlabel "emergencyDown"
              position [724.059 27.423 98.524 90.095]
              fontSize 12
              chart 2
              treeNode [2 0 22 0]
              subviewer 2
              ssIdNumber 2
              type OR_STATE
              decomposition CLUSTER_STATE
          }
          ...
          transition {
                    id 48
                    labelString "after(100,ticks)"
                    eventlabel "after(100,ticks)"
                    labelPosition [621.046 350.571 71.297 14.763]
                    fontSize 12
                    src {
                        id 19
                        intersection [3 0 1 0.547 623.5507 348.601 0 0]
                    }
                    dst {
                        id 26
                        intersection [0 0.5304 - 0.8477 - 1 580.8606 367.8496 0 0]
                    }
                    midPoint [605.6555 356.8862]
                    chart 2
                    linkNode [13 0 49]
                    dataLimits [580.861 623.551 348.601 367.85]
                    stampAngle - 0.294
                    subviewer 2
                    slide {
                        sticky BOTH_STICK
                        arcL -29.0042
                    }
                    executionOrder 1
                    ssIdNumber 42
                }
                ...
                junction {
                    id 26
                    position [577.1478 373.7835 7]
                    chart 2
                    linkNode [13 0 0]
                    subviewer 2
                    ssIdNumber 57
                    type CONNECTIVE_JUNCTION
                }
       ...
}
```

Total states(1499) & charts(339)	Extractor Only		Filtered Only		Filtered & Renamed		Filtered, Sorted & Renamed	
	state	chart	state	chart	state	chart	state	chart
Clone pairs	205	514	151	275	281	728	271	676
Clone class	24	27	20	23	44	27	43	30

Table 1 Initial results of the Stateflow model clones found in the Matlab demo set.

5.2.2 Renaming

Filtering improves the similarity of the clones, but SIMONE was not able to find all exact and near-miss state clones in the example model set. In order to identify the missing model clones, we must also remove naming differences.

We use a fixed value "x" to rename all attributes that represent internal information, and they are not relevant to clone comparison. Agile parsing is used during the parsing phase to grammatically distinguish the attributes that need to be renamed from those that should not be renamed.

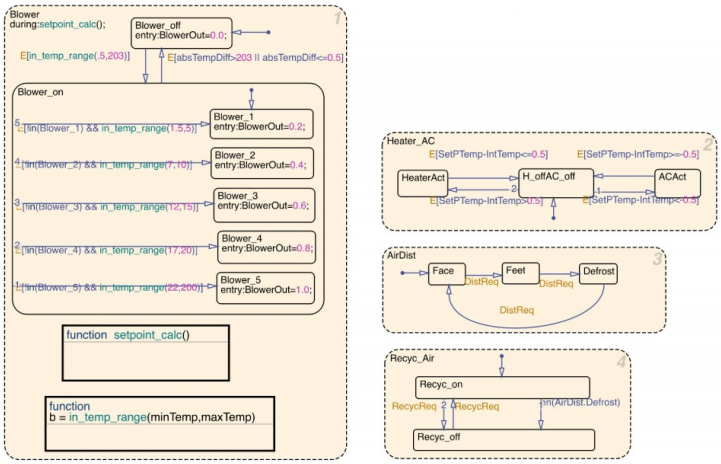
5.2.3 Sorting

While renaming improves clone detection, we found that the order of objects in two identical models may be different from each other. SIMONE compares potential clones line by line. Thus the order of graphical objects in textual representation of a model does not change its graphical meaning but it will affect the identification of clones. We developed a sorting plugin, which sorts the states by the number of nested elements.

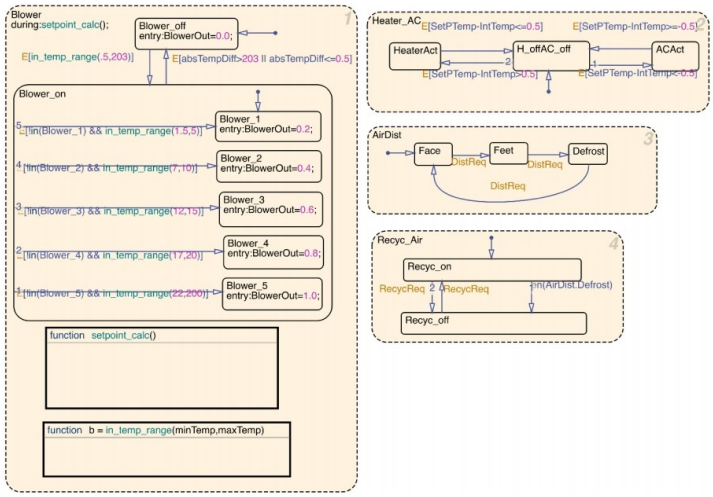
6 Experiment

We have conducted two main experiments using our new stateflow analysis approach and tool, the first experiment evaluates our tool on all of the publicly available Simulink models, including all of Matlab Central, and all of the demonstration systems distributed with Simulink. There are a total of 269 model files that contain Stateflow in the demo set[5]. Our initial, baseline experiment uses only the candidates extracted at both levels of granularities without any normalization. Using a threshold of 30% difference(i.e. at least 70% of the lines are the same) and a minimal clone size of 100 lines, we were able to extract 1499 states and 339 charts and find several clones in the demo set. A clone class is the equivalence class induced by the clone pair relationship. If a and b are clone pairs, and b and c are clone pairs, then a, b and c form a clone class. Table 1, the *Extractor only* column, shows the initial results. We found 205 state clone pairs clustered in 24 clone classes, and 514 chart clone pairs clustered in 27 clone classes.

[5] They come with Matlab installation located at the Matlab installation directory.



(a) ClimateControlSystem/Temperature Control Chart



(b) Temperature Control Chart

Fig. 11 Example of Stateflow clones from sldemo_auto_climate_elec.mdl and sldemo_auto_climatecontrol.mdl in Matlab demo automotive models

Examination of the results revealed models that are identical in the graphical view do not have one hundred percent similarity. At the same time, we also found some states having identical graphical representation do not show up in the clone detection result. If we adjust the threshold a little higher, some missing states will be found in the clone detection report. Thus a normalization step is required to improve the precision. The most obvious differences were differences in layout attributes, and normalizing these attributes could improve clone detection. Thus, we repeated the experiment after adding the filtering module, and after adding the sorting filtering modules. After verifying

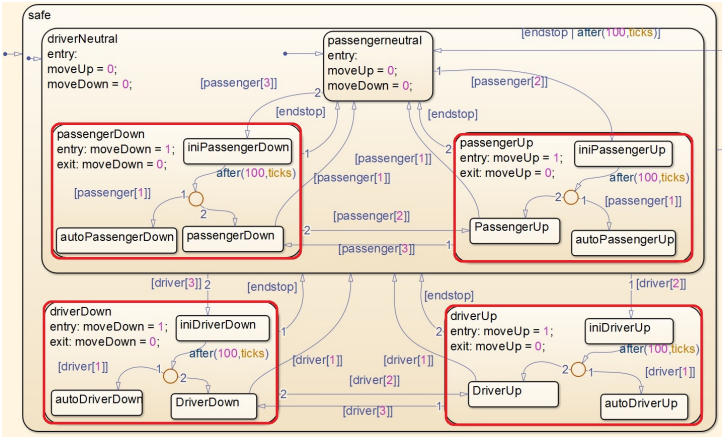


Fig. 12 A Type 2(renamed model clone), this example is in the powerwindow model. The four red states are similar to each other. SIMONE similarity 76%

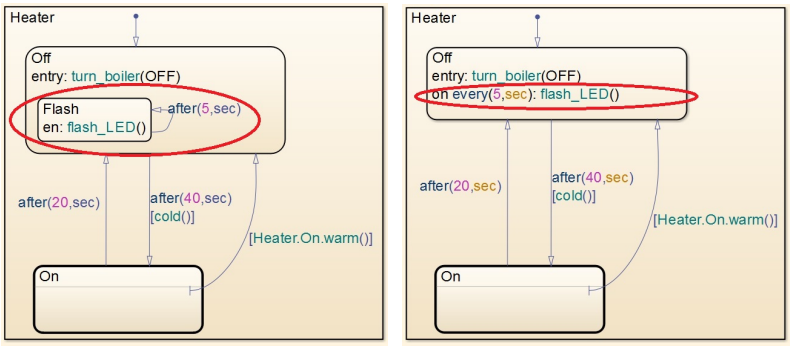
the results by hand, we found that the normalization of the states and charts are necessary to improve Stateflow clone detection precision.

Table 1 shows the total number of clone pairs and classes detected by each of the normalization options described in 5.2 . Filtering reduced the total number of clones by removing the false positives generated by similarities only in unimportant attributes. Renaming increased the number of clones detected by allowing different names to match. Sorting improved the quality resulting in slightly fewer clone pairs, but a few more clone classes.

The similarity of some of the clone pairs identified by using only the extractor is increased when using the filtering module. However, the filters do not identify more clone pairs and clone classes. The filtering can improve some similarity but not significantly. Figure 11 shows an example at chart level from two different Stateflow demo models, sldemo_auto_climatecontrol and sldemo_auto_climate_elec, which include the identical Temperature Control Chart.

Renaming significantly improved recall in finding exact and near-miss exact state and chart clones in the Stateflow demo models. Table 1, the *Filtered & Renamed* column, shows the result of renaming. We found 281 state clone pairs clustered in 44 clone classes, and 728 chart clone pairs clustered in 27 clone classes. New type 2 and type 3 clones were identified and the following examples are some of these cases.

Figure 12 shows an example type 2 clone of four different states in one chart in the powerwindow model of the Simulink example set. As you can see from the figure, the structure of each state is exactly the same, but the names and labels have been changed, replacing the string “passengerDown” with the string “passengerUp”, “driverDown” and “driverUp”. Figure 13 shows another type 3 clone between Bang-Bang Controller/heater state of sf_boiler



(a) sf_boiler/Bang-Bang Controller (b) sldemo_boiler/Bang-Bang Controller

Fig. 13 A Type 3(near-miss model clone), (a) in sf_boiler model and (b) in sldemo_boiler. SIMONE similarity 81%.

model and Bang-Bang Controller/Heater state of sldemo_boiler model of the Staeflow demo set. The red circles shows the difference between the two states.

Our second experiment evaluates our tool on models obtained from our industrial partner. These models are grouped into 9 sets called rings. The model set has 10 versions with a total of 426 models that include stateflow charts. Eleven of these model files are testing libraries. We were unable to parse 6 of the files. This left 409 model files, which contain 10655 charts and 22904 states in total. There are 276 lines in a chart on average, and and average of 13 lines in a state. We found that the states in this sample of models do not contain nested states. Thus we only conducted our analysis on the chart level of granularity.

Our evaluation is done in two ways. The first is to run clone detection on each of the 10 versions separately across all of the rings (i.e. the same version of the model files in each ring). The other was to examine run clone detection on all versions of each ring separately. The presence of clones across versions of a particular ring are not surprising as the same chart is present in each of the versions. What is more interesting is that there are 10 classes of clones in each version that are used in multiple rings. Some of these are very simple charts, while others are a bit more complex. Once filtered the 10 classes drop to 6 classes of clones, which remains consistent even after renaming and sorting. The main reason is that many of these charts were very similar, so the renaming and sorting did not remove any differences that were not already above the threshold. In the cases that we observed, the names of the states were identical, as were the structure of the states and transitions. The only differences between these charts were the events and actions on the transitions. Discussions with our industrial partner indicates that they use Stateflow in these cases for simple tasks and reuse proven solutions from one system in another. These classes appear to be a good basis for a set of automotive specific patterns in Stateflow.

Versions	# Models	Charts	States
Ver1	54	1409	3024
Ver2	22	575	1232
Ver3	54	1409	3024
Ver4	37	964	2072
Ver5	54	1409	3024
Ver6	21	547	1176
Ver7	48	1248	2688
Ver8	54	1404	3024
Ver9	15	234	504
Ver10	56	1456	3136

Table 2 The statistical information on the GM set by version.

Rings	# Models	Charts	States
Ring1	42	1105	2352
Ring2	33	866	1848
Ring3	56	1456	3136
Ring4	40	1040	2240
Ring5	21	546	1176
Ring6	21	546	1176
Ring7	90	2184	4704
Ring8	56	1456	3136
Ring9	56	1456	3136

Table 3 The statistical information on the GM set by ring.

As a parser-based technique, precision is not an issue for the NICAD engine (Roy and Cordy, Mutation 2009), and the real issue is recall, which we have addressed in the paper. Even so, precision was validated for *all* our test results by comparing to the original models by hand. Our tests included systems with models of over 100,000 source lines, which are parsed and processed in under a minute, and we continue to test larger scalability.

	Extractor Only			Filtered Only		Filtered & Renamed		Filtered, Sorted & Renamed	
	pairs	classes	cputime(min)	pairs	classes	pairs	classes	pairs	classes
Ver1	483250	10	16.75	585310	6	710859	6	710859	6
Ver2	80018	10	0.66	96958	6	117833	5	117833	5
Ver3	483250	10	15.99	585310	6	710859	6	710859	6
Ver4	226703	10	4.48	274618	6	333555	5	333555	5
Ver5	483250	10	19.95	585311	7	710858	7	710858	7
Ver6	72933	9	0.57	88368	5	107352	5	107352	5
Ver7	381748	10	12.2	462388	6	561552	5	561552	5
Ver8	483250	10	18.75	585310	5	710802	5	710802	5
Ver9	13329	9	0.07	16164	5	19647	5	19647	5
Ver10	519740	10	26.71	629500	6	764456	5	764456	5
Ring1	292294	12	8.85	354034	8	430111	6	430111	6
Ring2	180349	13	2.88	218464	9	265295	8	265295	8
Ring3	519848	9	26.42	764456	5	764456	5	764456	5
Ring4	265080	9	6.91	321080	5	389880	5	389880	5
Ring5	72933	9	0.57	88368	5	107331	5	107331	5
Ring6	72933	9	0.57	88368	5	107331	5	107331	5
Ring7	1169116	10	29.47	1416076	6	1720572	5	1720572	5
Ring8	519848	9	24.07	629608	5	764456	5	764456	5
Ring9	519848	9	32.23	629608	5	764456	5	764456	5

Table 4 Initial results of the Stateflow model clones found at chart level in the GM set.

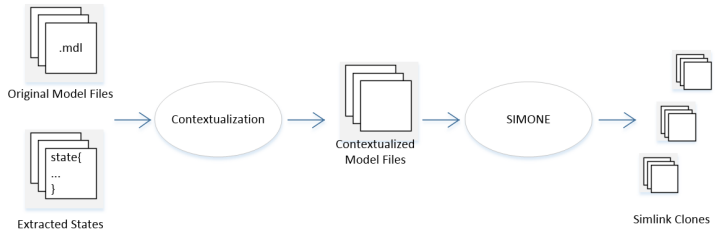


Fig. 14 Steps of contextualization

7 Contextualization

In the previous sections, we discussed how we extended SIMONE to detect clones in Stateflow models. However, clones in Simulink models and Stateflow models are detected separately when we apply SIMONE to the Simulink models. Since the Simulink blocks that link to the Stateflow models do so with a single attribute, similarity of the referenced Stateflow models is ignored when computing Simulink clones.

In this section we discuss how we can use the results of the Stateflow clone detection to improve the accuracy of the Simulink clones. We call this process contextualization, a process of putting the Stateflow model in the context of the Simulink model, for the purpose of improving the accuracy of SIMONE. Figure 14 shows the steps of contextualization. The first stage embed the extracted states into its parents Simulink to form the contextualized model

Total systems(1388)	System Only	Full lines	One line	Weighted lines
Clone pairs	4100	4902	4182	5058
Clone classes	24	42	25	32

Table 5 Initial results of the contextualization Stateflow model clones found in the Matlab demo set.

files as SIMONE's initial input. The second stage uses SIMONE to identify potential clone candidates and clusters them into classes.

The link between the Simulink and Stateflow models is encoded in two attributes. There is a *chartFileNumber* attribute in each separate Stateflow model in the file that acts as an identifier. The second attribute is the *Tag* attribute that is present in each of the blocks that represent Stateflow models in the Simulink model. This is a one to many relationship that allows a single Stateflow chart to be used in multiple places in a Simluink model. The format of the *Tag* attribute starts with the characters "Stateflow S-Function", a model file name, and the index number.

We insert each chart into its parent block by the referencing number. The contextualization is similar to the work done by Grant et al. (S. Grant and Skillicorn, 2011) in identifying contextual clones in WSDL documents. We investigate three different ways in which we can contextualize the models: *full lines*, *one line*, and *weighted lines*.

7.1 Contextualization via full lines

Full lines means we put the entire state machine back into its parent Simulink block. The chart text we used for full lines is the chart textual representation after normalization (folding, line splitting, renaming, etc.). Listing 4 shows the snippet of the contextualized textual representation of the MATLAB power window demo. The *System* block has an "S-Function" block which contains the *control* chart.

In our experiment, we picked only models that contained both *Stateflow* and *chart* blocks that we could examine for the contextualization. The normalization plugins of SIMONE conformed with Stateflow, so we used filtering, blind renaming, and sorting plugins, and a threshold of 30% at the *system* level of granularity. We were expecting a more accurate result when using SIMONE on the combined models using the above parameter settings. Column two in table 5 shows the result of the full lines experiment. SIMONE can detect more clone pairs and clone classes after the contextualization.

When examining the results, we can ignore all the system without chart clones as they remain the same as before, and focus on those system clones containing charts. Changes in results can be classified into three categories:

1. System clones were still clones.
2. System clones were missing.
3. New System clones appeared.

Listing 4 Example snippet of the contextualized fragment of "power_window_control_system" system in powerwindowlibsa.mdl.

```
System {
    Name "power_window_control_system"
    ...
    Block {
        BlockType SubSystem
        Name "control"
        ...
        System {
            Name "control"
            ...
            Block {
                BlockType S-Function
                Name " SFunction "
                SID       "45::15"
                Tag       "Stateflow S-Function powerwindowlibsa 1"
              chart {
                 chartFileNumber 1
                  ...
                  data {
                      ...
                  }
                  ...
                  event {
                     ...
                  }
                  state {
                      ...
                      id "x"
                      ...
                  }
                  transition {
                      ...
                  }
                  ...
            }
            Block {
                BlockType Terminator
                Name " Terminator "
            }
            ...
        }
    }
    ...
}
```

SIMONE is a line-based clone detector, so the number of lines of a chart embedded in a system potential clone candidate affects the result of the contextualized clone detection. If the number of lines in embedded Stateflow models overwhelms the number of lines in the original Simulink host system, then the charts' similarity will dominate the clone detection result. In our test set, the average number of lines in a system is 216 and the average number of lines in a chart is 342. We can see the effect from the following three categories.

Category one

System clones had the same or similar charts in the cloned Simulink models. In our test set, there were several versions of the "powerwindow" model with

	Systems	System Only		Full Lines		One Line		Weighted Lines	
		pairs	classes	pairs	classes	pairs	classes	pairs	classes
Ver1	16159	103188	71	749996	83	103188	71	103189	72
Ver2	6456	16622	40	124181	51	16622	40	16622	40
Ver3	16138	103189	71	749997	83	103189	71	103190	72
Ver4	11169	49244	46	352969	57	49244	46	49244	46
Ver5	16124	103258	70	750065	81	103258	70	103258	70
Ver6	6119	15218	37	113175	48	15218	37	15218	37
Ver7	14389	81952	59	593014	70	81952	59	81952	59
Ver8	16320	104963	64	751578	75	104963	64	104963	64
Ver9	2788	2916	23	20943	32	2916	23	2916	23
Ver10	16899	112967	61	808339	72	112967	61	112967	61
Ring1	11232	57763	74	449570	85	57763	74	57763	74
Ring2	9382	39220	43	280747	52	39220	43	39220	43
Ring3	16983	114345	64	809501	73	114345	64	114345	64
Ring4	12591	58166	69	412946	78	58166	69	58166	69
Ring5	6420	16001	37	113882	46	16001	37	16001	37
Ring6	6438	16027	38	113908	47	16027	38	16027	38
Ring7	25532	256632	71	1822510	82	256632	71	256632	71
Ring8	16880	113402	78	808558	87	113402	78	113402	78
Ring9	17103	113146	65	808302	74	113146	65	113146	65

Table 6 Initial results of the contextualization Stateflow model clones found in the GM set.

a similar system "power_window_control_system" and all of them contained an identical chart "control". SIMONE can identify the "power_window_control_system" system clones both before and after contextualization as the identical chart.

Category two

System clones contained different charts and the size of the chart overwhelmed the hosting Simulink subsystem. For example, SIMONE reported that system "fp_verify_current/detect_obstacle" in powerwindowlibsa.mdl, and system "Mixing & Combustion" in sldemo_fuelsys.mdl are cloned at 82% similarity. The size of both systems is 162 lines; the size of the "delay_detection" chart is 175 lines and the size of the "EGO Sensor" chart is 89. Obviously, the result of Simulink clone detection depends on the similarity of charts which dominate the similarity of the Simulink model.

Category three

Category three was the detection of the new cloned pair. Original non-cloned Simulink models contained the same or similar charts, and the size of the chart was big enough to lead to the clone detection result. Two demo systems were repored based on a comparison of 1214 lines, and the chart took about 810 lines. So the similarity of the chart contributed to this clone detection result.

7.2 Contextualization via one line

The problem with the full line contextualization is that the size of the embedded Stateflow models overwhelmed the size of the host Simulink graph. In the one line contextualization we put a single reference line of Stateflow to its parent. First we perform a Stateflow clone analysis, in which SIMONE reports clone classes by clustering similar charts into groups and assigning a unique id. We invent a new Simulink attribute, *classid* which we insert into the host Simulink model, the value of which is the clone class id from the Stateflow clone analysis. Those Stateflow charts that did not belong to a clone class (i.e. not similar to any other Stateflow chart) are assigned unique class ids distinct from the detected clone classes. Listing 5 is an embedded one line version of the previous example. Column three in table 5 shows the result of the one line experiment.

Listing 5 Example snippet of the one line contextualized fragment of "power_window_control_system" system in powerwindowlibsa.mdl.

```
System {
    Name "power_window_control_system"
    ...
    Block {
        BlockType SubSystem
        Name "control"
        ...
        System {
            Name "control"
            ...
            Block {
                ...
                Tag       "Stateflow S-Function powerwindowlibsa 1"
                classid   2
            }
            ...
        }
    }
    ...
}
```

The one line contextualization clone detection result still falls in the same three categories. The clone pairs in category one, system clones were still cones, take the majority of the result. There are 4038 clone pairs belonging to category one in 4100 total clone pairs and most of them have the same similarity percentage as before, some of them are within ±1% range. Sixty two systems were 71% similar before contextualization and the single line contextualization took them below the threshold. Several other clone pairs were created when the extra lines pushed the similarity just over the threshold. So the *one line* changes have a small effect on the contextualization.

7.3 Contextualization via weighted lines

Weighted lines means putting an average weight of a Stateflow clone back to its parent. Full line (i.e., entire chart) is aggressive and one line is marginal. Thus, we did another experiment that gives each chart a weight by inserting

the *classic* attribute multiple times into the Simulink block. Based on our group experience, the average lines of a block are about ten lines, so we give each chart a weight of ten lines. Listing 6 is an embedded weighted lines version of the previous example. Column three in table 5 shows the result of the weighted lines experiment.

The result of weighted lines is similar to the previous one line experiment.

Listing 6 Example snippet of the weighted lines contextualized fragment of "power_window_control_system" system in powerwindowlibsa.mdl.

```
System {
    Name "power_window_control_system"
    ...
    Block {
        BlockType SubSystem
        Name "control"
        ...
        System {
            Name "control"
            ...
            Block {
                ...
                Tag       "Stateflow S-Function powerwindowlibsa 1"
                classid   2
                classid   2
                classid   2
                classid   2
                classid   2
                classid   2
                classid   2
                classid   2
                classid   2
                classid   2
                  ...
            }
            ...
        }
    }
    ...
}
```

There are 3603 clone pairs belonging to category one and still taking the majority part of the result. The variation in the level of similarity is ranging from -4% to +7%, and most of the changes are within ±2%. Some small size clones like 126 lines clones can get 7% change, and some larger clones do not change at all. There are 497 clone pairs in category two. The similarity is ranging from 71% to 89% with most of them having 71%,72% and 73% similarity.

7.4 Contextualization discussion

From the above experiments, we can see each approach has its own merits. *Contextualization via Full lines* can obtain the best result of the combination

clone detection, if the goal of the model clone detection is to identify the duplication of Simulink models that contain identical or similar Stateflow models. The comparison of clone detection will take into account every line of both Simulink and Stateflow models in the full lines approach; and we can examine model clones from a larger perspective. If the goal of model clone detection is more focus on the Simulink models, then the *contextualization via one line* approach would be better. In this approach, the Stateflow model is just represented as one single line in its parent Simulink model, so it will not affect the comparison of Simulink too much. Meanwhile, we still have the Stateflow information inside the Simulink model. *Contextualization via weighted lines* presents a more flexible way to detect the duplication of combining two type of models. It can avoid the Stateflow model overwhelm the Simulink model and also remain enough Stateflow model information during the comparison.

8 Related work

While code clone detection has been extensively researched (Roy et al, 2009), research on model clones identification has received less attention (Deissenboeck et al, 2010). Thus far, a few researchers have tried to find the clones in UML behavioural models and Matlab/Simulink models.

Liu et al. (Liu et al, 2006) proposed a suffix-tree based algorithm to identify duplications in UML sequence diagrams. They converted the 2-dimensional sequence diagram to a 1-dimensional array and constructed a suffix tree from the 1-dimensional array. Their approach identified the common prefixes in the suffix tree and ensured that the duplications detected are extractable and reusable sequence diagram as refactoring candidates.

Antony et al.(Antony et al, 2013) proposed an approach for identifying near-miss interaction clones in reverse-engineered UML behavioural models. They used a text-based technique and worked on the level of XMI. Their approach transformed the XMI sequence diagram serialization into a contextualized form and extracted the sefl-contained units of behavioural interaction as clone candidates. A standard code clone detector is applied to identify cloned behavioural interactions from the large set of contextualized textual representation.

Störrle (Störrle, 2013) proposed an approach to identify clones in UML models, specifically class, activity, and use case diagrams. The approach is based on model matching and model querying(Störrle, 2009). He implemented the MQ_{lone} tool to evaluate this algorithm. The tool transforms XMI files, that are generated from UML domain models by using a contemporary UML case tools as input, into Prolog files. Using model matching technique to generate the output from the input model in the query. Störrle uses a different definition of model clones. His definition requires that the structure of the models are the same and that the labels on each of the model elements are similar. Thus, his approach identifies Type 1 and Type 2 clones, but not Type 3 near-miss

clones. He also claims the approach is extendable to Simulink and Stateflow models. However, the approach has not been demonstrated on StateFlow.

The majority of mode clone detection approaches have been tailored for Simulink models(Deissenboeck et al, 2010; Alalfi et al, 2012a; Deissenboeck et al, 2008; Alalfi et al, 2012b; Stephan et al, 2012; Pham et al, 2009), and these techniques either use graph based comparison or text-based techniques to do clone detection on Simulink models. None of them has been applied to Stateflow models.

Deissenboeck et al.(Deissenboeck et al, 2008) present one of the first methods to detect the duplication in Simulink models especially in automotive domain. The approach is based on graph theory and can detect model clones in Simulink and other graph based data-flow models. In their approach, models are presented as a flattened multigraph where each block and linear connections are normalized by assigning a value. The duplications are checked by performing a depth first search to find matching paragraphs. They implemented their algorithm as a part of the quality analysis framework ConQAT(Deissenboeck et al, 2005) which is publicly available as open source software[6]. Juergens et al.(Juergens et al, 2009) adapted this algorithm to form a clone detection tool chain *CloneDetective*, which is designed as an open source "workbench for clone detection research" and based on the open source tool ConQAT.

Pham et al.(Pham et al, 2009) proposed an other graph-based clone detection tool for Matlab/Simulink models called *ModelCD*, which consists of two algorithms, *eScan* and *aScan*. In their approach, the model were represented as a parsed, labelled directed graph and larger clones were identified by adding edges to smaller, already detected clones. The *eScan* algorithm was designed to detect exact clones achieved by an advanced canonical labelling technique, and the *aScan* algorithm was designed to exact and approximate clones by computing a vector-based approximation of the structure with a subgraph.

Al-Batran et al.(Al-Batran et al, 2011) noted that these approaches only consider syntactic clones, so they extended these approaches to cover semantic clones that may have similar behaviour but different structure by using the pattern-based normal-form approach, which normalized model graphs using the models semantic information.

Hummel et al.(Hummel et al, 2011) present an index-based algorithm for model clone detection that is incremental and distributable. In their approach, a Simulink/Matlab model was represented as a directed multigraph and the normalization assigned labels to relevant edges and blocks. The canonical label were computed for each subgraph in a clone index, which is a list of all subgraphs having the same size. The clone retrieval process merged clone pairs with same size.

[6] http://www.conqat.org

9 Conclusions and Future Work

SIMONE has been successfully used in finding near miss subsystem clones in Simulink models (Alalfi et al, 2012a). It adapted a text based code clone detector NiCad to enable the identification of graphical model clones. In this work, we present an extension for SIMONE to perform clone detection in Stateflow models. We define two levels of granularity *charts* and *states* in SIMONE to identify model clones for Stateflow models. Charts in Stateflow represents entire state machines. States can contain other Stateflow objects to form a multilevel complex state in a hierarchical structure. State granularity extracts the states from the Stateflow model files as clone candidates. The extension was evaluated against those MATLAB example models that contain Stateflow models.

Besides identifying Stateflow model clones, we also investigate explicating the state machines into the parent Simulink model, using the similarity of state machines to improve the accuracy of Simulink clones. We import all the state charts referenced by the Simulink blocks into the self-contained unit in the textual representation level by using three different ways full lines, one line, and weighted lines. We have packaged up the Stateflow clone detector in SIMONE and made it available for download, and we are currently working on packaging the explication of Stateflow clone results into the main version of SIMONE as well.

The capabilities of SIMONE could be further improved in several research directions. The initial clone detection results from the Matlab example set are similar machines with variations in labels(i.e. state and transition names) and other attributes such as position. We still need to evaluate our approach on more Stateflow models, as well as to refine our SIMONE plugin to improve clone detections. We also found some clone classes that appear to be embedded Matlab code for use by state and transition labels. Improving our approach to better deal with embedded code is also a line of future research. We also can address our model clone issues further to turn the model clones into model patterns, so that we can better assist and understand model reuse in model development environment.

References

Al-Batran B, Schätz B, Hummel B (2011) Semantic clone detection for model-based development of embedded systems. In: Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems, Springer-Verlag, Berlin, Heidelberg, MODELS'11, pp 258–272, URL http://dl.acm.org/citation.cfm?id=2050655.2050681

Alalfi M, Cordy J, Dean T, Stephan M, Stevenson A (2012a) Models are code too: Near-miss clone detection for Simulink models. In: ICSM, pp 295–304

Alalfi M, Cordy J, Dean T, Stephan M, Stevenson A (2012b) Near-miss model clone detection for Simulink models. In: IWSC, pp 78–79

Antony E, Alalfi M, Cordy J (2013) An approach to clone detection in behavioural models. In: Reverse Engineering (WCRE), 2013 20th Working Conference on, pp 472–476, DOI 10.1109/WCRE.2013.6671325

Chen J, Dean TR, Alalfi MH (2014) Clone detection in matlab stateflow models. In: Proceedings of the 8th International Workshop on Software Clones, Elec. Comm. EASST, vol 63, p 13 pp.

Cordy JR (2006) The TXL source transformation language. Sci Comput Program 61(3):190–210

Cordy JR (2013) Submodel pattern extraction for simulink models. In: Proceedings of the 17th International Software Product Line Conference, ACM, New York, NY, USA, SPLC '13, pp 7–10, DOI 10.1145/2491627.2492153, URL http://doi.acm.org/10.1145/2491627.2492153

Cordy JR, Roy CK (2011) The NiCad clone detector. In: Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension, IEEE Computer Society, Washington, DC, USA, ICPC '11, pp 219–220, DOI 10.1109/ICPC.2011.26, URL http://dx.doi.org/10.1109/ICPC.2011.26

Deissenboeck F, Pizka M, Seifert T (2005) Tool support for continuous quality assessment. In: Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on, pp 127–136, DOI 10.1109/STEP.2005.31

Deissenboeck F, Hummel B, Jürgens E, Schätz B, Wagner S, Girard JF, Teuchert S (2008) Clone detection in automotive model-based development. In: Proceedings of the 30th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE '08, pp 603–612, DOI 10.1145/1368088.1368172, URL http://doi.acm.org/10.1145/1368088.1368172

Deissenboeck F, Hummel B, Juergens E, Pfaehler M, Schaetz B (2010) Model clone detection in practice. In: IWSC, pp 57–64

Dominguez ALJ (2012) mdl2smv: A tool for translating automotive feature models in matlab's stateflow to smv. https://cs.uwaterloo.ca/ aljuarez/mdl2smv.html, accessed: 2014-11-06

Harel D (1987) Statecharts: A visual formalism for complex systems. Science of Computer Programming 8(3):231–274

Hirschberg DS (1977) Algorithms for the longest common subsequence problem. J ACM 24(4):664–675, DOI 10.1145/322033.322044, URL http://doi.acm.org/10.1145/322033.322044

Hummel B, Juergens E, Steidl D (2011) Index-based model clone detection. In: Proceedings of the 5th International Workshop on Software Clones, ACM, New York, NY, USA, IWSC '11, pp 21–27, DOI 10.1145/1985404.1985409, URL http://doi.acm.org/10.1145/1985404.1985409

Juergens E, Deissenboeck F, Hummel B (2009) Clonedetective - a workbench for clone detection research. In: Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, ICSE '09, pp 603–606, DOI 10.1109/ICSE.2009.5070566, URL http://dx.doi.org/10.1109/ICSE.2009.5070566

Koschke R (2006) Survey of Research on Software Clones. In: Dagstuhl Seminars

Liu H, Ma Z, Zhang L, Shao W (2006) Detecting duplications in sequence diagrams based on suffix trees. In: Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific, pp 269–276, DOI 10.1109/APSEC.2006.32

Martin D, Cordy JR (2010) Towards web services tagging by similarity detection. In: Chignell M, Cordy J, Ng J, Yesha Y (eds) The Smart Internet, Lecture Notes in Computer Science, vol 6400, Springer Berlin Heidelberg, pp 216–233

Martin D, Cordy JR (2011) Analyzing web service similarity using contextual clones. In: Proceedings of the 5th International Workshop on Software Clones, ACM, New York, NY, USA, IWSC '11, pp 41–46, DOI 10.1145/1985404.1985412, URL http://doi.acm.org/10.1145/1985404.1985412

Pham N, Nguyen H, Nguyen T, Al-Kofahi J, Nguyen T (2009) Complete and accurate clone detection in graph-based models. In: Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on, pp 276–286, DOI 10.1109/ICSE.2009.5070528

Roy CK, Cordy JR (2007) A survey on software clone detection research. SCHOOL OF COMPUTING TR 2007-541, QUEEN'S UNIVERSITY 115

Roy CK, Cordy JR, Koschke R (2009) Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming 74(7):470 – 495

S Grant JC D Martin, Skillicorn D (2011) Contextualized semantic analysis of web services. In: WSE 2011, pp 33–42

Stephan M, Alafi M, Stevenson A, Cordy J (2012) Towards qualitative comparison of simulink model clone detection approaches. In: IWSC, pp 84–85

Stevenson A, Cordy JR (2012) Grammatical inference in software engineering: an overview of the state of the art. In: Hedin (Eds.), Pre-proceedings of the Fifth International Conference on Software Language Engineering (SLE 2012), Fakultät Informatik, Technische Universität, pp 206–225

Störrle H (2009) VMQL: A generic visual model query language. In: Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on, pp 199–206, DOI 10.1109/VLHCC.2009.5295261

Störrle H (2013) Towards clone detection in uml domain models. Software and Systems Modeling 12(2):307–329

The MathWorks Inc (2014) Stateflow Hierarchy of Objects. http://www.mathworks.com/help/stateflow/ug/stateflow-hierarchy-of-objects.html, [Online; accessed March-2014]