Review of Clone Detection in Models

Jyoti Khanna
CSE Department,
DeenBandhuChhotu Ram
University of Science and
Technology,
Murthal, Haryana, India

Rajvir Singh
CSE Department,
DeenBandhuChhotu Ram
University of Science and
Technology,
Murthal, Haryana, India

Ritu Garg
CSE Department,
DeenBandhuChhotu Ram
University of Science and
Technology,
Murthal, Haryana, India

Abstract:- **Nowadays Model based methodology is used for development. Models are designed before coding. As clones exist in the source code, similarly clones exist in the models. It affects the quality of software and increase maintenance cost. Many solutions have been proposed for code clones, but a little work has been done on model clones. In this paper techniques for model clone detection have been discussed.**

Keywords:- Code clones, Model clones, UML model, Matlab/Simulink, Graph.

I. INTRODUCTION

Copy and Paste of fragments has been used in software development. This strategy is known as cloning. Cloning can be either at designing level or at implementation level. At implementation level clones exist in the source code and during designing clones exist in the models. Clones increase redundancy in the software which cause problem in software maintenance. And cloning also increase probability of bugs and maintenance cost. So clones need to get removed. Many Solutions have been proposed to remove clones. Many strategies have been applied for code clones, but a few solutions are proposed for model clones. There are many challenges in identifying model clones. Strategies of Model Clones are discussed in next section.

A. Type of Model Clones [1]:

Till now there is no proper definition for model clones. There is no proper classification of Model clones, still these are classified based on some criteria and given below.

1) Type 1 (exact model clones): In this type, model fragments may vary in visual presentation, layout and formatting otherwise they are identical to each other.

2) Type 2 (renamed model clones): These model fragments are structurally identical and these may be varied in labels, values, types, visual presentation, layout and formatting.

3) Type 3 (near-miss model clones): Model fragments may vary in position or connection and there may be additions or removals of blocks or lines in addition to variations as in Type-1 and Type-2.

II. LITERATURE REVIEW

Dhavleesh Rattan et al. [2] has proposed a technique to detect clones in UML class diagram. The graphical Unified Modeling Language (UML) is increasingly replacing conventional programming languages for developing software systems [3]. In UML model's graph the nodes are the classes and the edges are the relationship of two classes. Nodes of UML model are heavy and dense because they contain information. Because of this, detecting clone in UML diagram give better results. Nodes of Matlab/Simulink models are light weighted. So isomorphic graph comparison are applied in Simulink, but can't be applied in UML models. Reference [2] has used following steps to detect clones in UML class diagram as shown in figure 1:

1) Model is created using any tool.
2) The model is exported to XMI (XML Metadata Interchange) file format. Since XMI is a standard given by OMG, it is built in most of the modeling tools.
3) XMI file is preprocessed and is stored in the form of tree using DOM API's and XML parsing.
4) Sub-trees are compared and similarity is reported in the form of model clones.

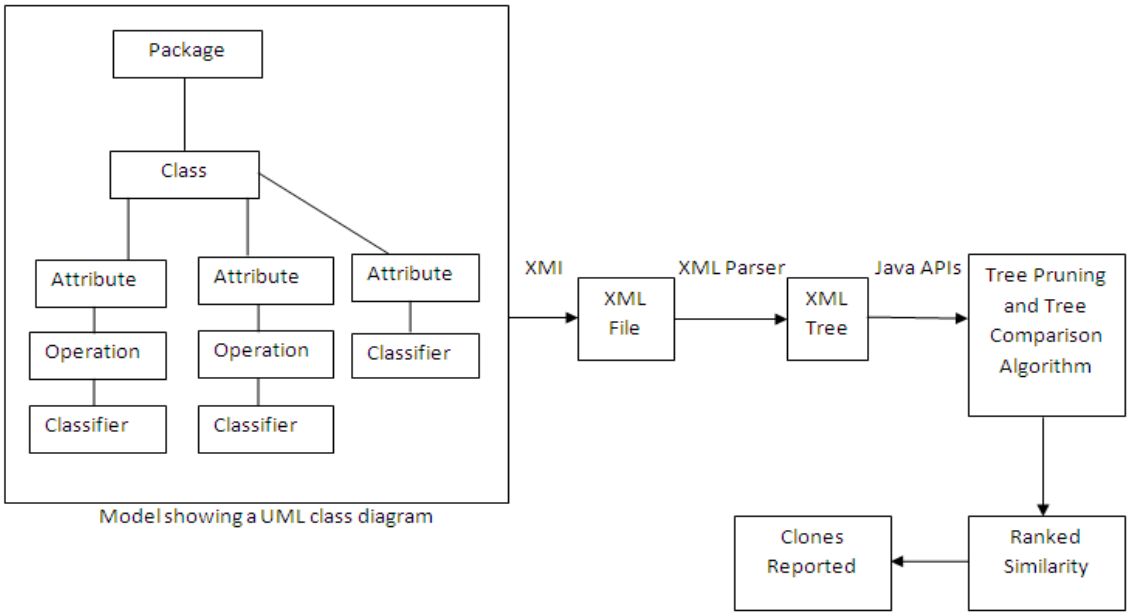

Fig. 1: Block Diagram of clone detection [2]

In [4] UML diagrams are encoded in XMI files to find out differences between these diagram. This technique has been explored to detect clones. The elements of diagram element are compared and similarities are measured between those elements. Based on the values of similarities these elements are reported as clones. In [4] comparison is done on the basis of Id but here comparison is done on trees of XMI trees. Every subtree is compared with other subtree. This Technique is scalable. In this technique comparison is done on trees which are better than textual clone detection in XMI file. It avoids irrelevant repetitions. This technique is implemented in java. UML model's nodes are loosely connected and heavy so better results are obtained.

Manar H. Alalfi et al. [1] has introduced SIMONE. SIMONE uses text based clone detector NICAD [5] to detect near miss clones in the Matlab/Simulink model. Simulink stores textual representation of models on disk. This representation is given as input to SIMONE. Following steps are used in this technique:

1) Simulink TXL grammar
2) Extractor Plug in
3) Filtering
4) Sorting
5) Renaming

In the first step Simulinks are converted into TXL grammar. NICAD is language sensitive clone detectors. It uses TXL parser [6]. So it needs to convert Simulink in TXL grammar. Grammar inference techniques are used for this. This grammar
identifies all simulink constructs, including models, systems, block, lines, ports etc.

In the second step Extractor Plug in are used to extract potential clones. Potential Clones are result of extraction and normalization of instances. NICAD identifies structurally meaningful clones i.e. classes, method, blocks and lines. NICAD uses relaxed textual comparison on those clones. Modeling languages are hierarchical. Three levels are there in Simulinks:

1) Model Granularity: Entire Simulink models as clones. Simulink models consist of (sub-) systems, which themselves are built up from blocks and lines.

2) System Granularity: On the Simulink "system" (subsystem) level, clones are identified in two dimensions.

- Exact subsystem clones across two different models,
- Near-miss subsystem clones within a single mode

3) Block Granularity: Blocks represent a group of parts that work together for a specific functionality.

Simulinks are different than programming languages. So extractor plug in are designed in a different way.

In the third step filtering is done. When simulinks are converted into textual form, meaning of models is changed. And a little change in attributes such as color and fonts will not identify identical clones. So filter plug-ins are designed to remove these irrelevant differences. Recalling is improved by filtering.

In the fourth step Sorting is done. Even after filtering some clones are not detected, because when simulinks are converted into textual form then order of block, lines, branches and ports will be changed in textual form. So canonical sorting is implemented on models. Sorting plug-ins sort Blocks by Type Name, Sort Lines by Source Block, Sort Ports by Port Name and Sort Branches by Destination Block.

In the fifth step sophisticated blind renaming plug-ins are used for Simulink. Problems of linear representation are resolved by sorting. SIMONE can find out exact and near-miss exact subsystem clones. But to find all type 2 i.e. renamed subsystem clones renaming is required. The generic renaming algorithm provided with NICAD to rename identifiers in other programming languages. This algorithm cannot be used for Simulink. In Simulink model texts are represented as quoted strings. And some texts like

block types and line types are not renamed. To rename, TXL agile parsing techniques are used to distinguish elements grammatically which has to be renamed and which need not to be renamed. This transformation is installed as a renaming plug-in for Simulink. The plug-in anonymize all names and values associated with elements and blocks, preserving only Block Type and Line Type elements for comparison, allowing for detection of near-miss type 2 subsystem clones in Simulink models.

This technique does not report false positive. Recall of this technique is very good.

Florian Deissenboeck et al. [7] has used graph based theory to detect clones. This technique works on Matlab/Simulink models. This approach consists of three steps: In the first preprocessing is done then at second level Simulinks are normalized and clone pairs are extracted. And in the last step pairs are clustered to find substructure used more than twice in the model.

In the preprocessing phase models are read and flatten them. Unconnected lines are removed. In the normalization step label is assigned to block and line. Label may consists of some attributes which are relevant for differentiate between them. If two blocks have same label they are considered as equivalent. Some information is also included to labels which depend upon type of class to be detected. For example if relation operator block is used, then type of operator like less than and greater than is also included. In case of lines, indices of the source and destination of ports are stored in the label. This graph will be multi-graph because a simulate block may have multiple ports and each will be connected to a line. Nodes are processed in breadth-first-search manner. Three sets C, S and D of current nodes, seen nodes and done nodes are managed respectively. If a node is currently built, no processing is done on the node. If current node exists in seen node, it is considered as clone for corresponding node. A node pair is considered a clone pair if it follows a mapping P. All block pairs of P follow two conditions:

$$L(x) = L(y) \quad (1)$$

$$(u, x), (v, y) \pounds E \text{ and } L((u, x)) = L((v, y))$$

Or (2)

$$(x, u), (y, v) \pounds E \text{ and } L((x, u)) = L((y, v))$$

If a sub-graph exists in the graph n times then above method would report n*n-1/2 clone pairs. In this phase we connect them into single class. This algorithm can be applied to real world models.

Problem with this algorithm is that it report large number of false positives.

Pham et al. [8] proposed a tool ModelCD. It uses two algorithms escan and ascan which detect efficiently and accurately exactly matched and approximate model clones respectively in Matlab/Simulink. A Simulink model is represented as a sparse, labeled directed graph. Clones in that model are considered as its weakly connected and non-overlapping sub-graphs. Clones are detected into three steps: generating, grouping, and filtering.

In the first phase blocks are combined to form composite blocks as in ConQAT [9]. Basically, it consists of three tasks: parsing, flattening and labeling. This phase results a labeled, directed graph *G* in which the set of nodes *V* represents Simulink blocks, the set of directed edges *E* represents the signal lines and the labeling function *T* assigns the labels to nodes and edges. There are multiple signal connections between two blocks, which causes G to be a multi-graph.

In second phase, isomorphic graphs are grouped to generate larger isomorphic candidates with extension of edge using depth first order in escan. While in ascan, hashing and maximal clique cover methods are used for the vectors using breadth first order. This technique is incremental and is able to detect model fragments with modifications. Escan produces complete and accurate clone results with higher quality and much more quantity but at larger running time. Ascan detects approximate clone matching by using a vector-based technique, exas [10]. Two structural patterns are used by exas in a graph or sub-graph (p, q)-node and n-path. Exas uses the occurrence-count vector of the features as its characteristic vector. Occurrence-count vector is extracted from that fragment. That is, each position in the vector is indexed for a feature and the value at that position is the number of occurrences of that feature in the fragment. The model clone granularity is number of blocks here.

Third phase includes filtering. Filtering process is applied to remove the redundant groups. Ascan performs filtering at level *k* in this way, it needs to check redundancy only between the groups created at that level and the ones at level (*k*-1).

ModelCD provides good scalability, completeness and high precision.

Liu et al. [11] proposed a tool DuplicationDetector. It detects duplications in sequence diagram. Sequence diagrams are used as interaction diagrams to describe behaviors of use cases, operations and collaborations. It describes how the processes operate and in what order. The duplications occur because of system's complexity , poor design and reluctance to restructure the design and due to various existing scenarios with a main execution flow and several alternate flows. These duplications hamper maintainability and reusability [12].

In preprocessing phase, the 2-dimensional sequence diagrams are converted to 1-dimensional array. The arrays are concatenated into a long array and a suffix tree is constructed using it. Longest common prefix of two suffixes is identified in form of reusable sequence diagram as refactoring candidate.

It is an intra system clone detection approach. It results in high precision and recall.

Hummel et al. [13] pioneered a tool that is based on incremental instead of batch mode clone detection. It takes input Simulink/matlab model. In preprocess phase, it converts the model into a directed multi graph and assigns labels to relevant blocks. In detection phase, graph isomorphism is determined which is based on canonical labeling (unique code invariant to ordering of vertices and edges). A small change need not entire detection using index- based algorithm that is incremental and distributable. Hash code is used as a heuristic.To reduce runtime hash code is generated using md5 hashing. In post-processing phase, cloning information is filtered, prevented

or used by the clone management tools. It is reused by clone detector ConQat [9]. Clone index is created for all the sub-graphs of same size. On basis of canonical labeling clone index is calculated and similar labels are hashed. Due to index update and clone retrieval the run time is less which results in fast retrieval but for small models only as it has not been verified on large models.

III. COMPARISONS

Table I: Comparisons of techniques

	Technique applied for	Scalability	Precision	Recall
Manar H. Alalfi et al. [1]	Matlab/Simulink	Medium	High	Not Well
Dhavleesh Rattan et al. [2]	UML models	High	High	High
Florian Deissenboeck et al. [7]	Matlab/Simulink	High	Less	Medium
Pham et al. [8]	Matlab/Simulink	High	Less	High
Liu et al. [11]	Sequence Diagram	Less	High	High
Hummel et al. [13]	Matlab/Simulink	Medium	High	Medium

IV. CONCLUSION

Model clones are as harmful as code clones. Model clones also increase maintenance cost and probability of bugs. So these need to be removed. Different tools have been proposed for model clone detection. Each tool is designed for particular model. Different methodologies are used by tools. Each methodology has its own advantage and limitations. Still very few solutions are available for model clone detection. Many methodologies are expected to be proposed in the future.

FUTURE SCOPE

Many other solutions can be found out for model clone detection. There is no proper classification of model clones. Model clones should be classified in proper manner. With better classification, clones can be detected easily. Many other techniques other than refactoring should be proposed.

REFERENCE:

[1] M. H. Alalfi, J. R. Cordy, T. R. Dean, M. Stephan, A. Stevenson, SIMONE: Models are Code too, Near-miss Clone Detection for Simulink Models, IEEE, 2012.
[2] D. Rattan, R. Bhatia, M. Singh, Model Clone Detection based on Tree Comparison, IEEE, 2012.
[3] T. Weilkiens, Systems Engineering with SysML/ UML, MorganKaufmann, 2007.
[4] U. Kelte, J. Wehren and J. Niere, A generic difference algorithm for UML models, Proceedings of SE 2005, Essen, Germany, 2005.
[5] C. K. Roy and J. R. Cordy, NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization, in 16th Int. Conf. on Program Compreh., 2008.
[6] J. R. Cordy, The TXL source transformation language, Sci. Comput. Program, 2006.
[7] F. Deissenboeck, B. Hummel, E. Juergens, B. Schätz, S. Wagner, J.-F. Girard and S. Teuchert, Clone detection in automotive model-based development, Proceedings of 30th International Conference on Software Engineering, Leipzig, Germany, 2008.
[8] N.H. Pham, H.A. Nguyen, T.T. Nguyen, J.M. Al-Kofahi, T.N. Nguyen, Complete and accurate clone detection in graph based models, in: Proceedings of 31st International Conference on Software Engineering (ICSE˝09), Vancouver,Canada, 2009.
[9] B. Hummel, E. Juergens, L. Heinemann, M. Conradt, Index-based code clonedetection: Incremental, distributed, scalable, in: Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM˝10), Timisoara,Romania, 2010.
[10] H.A. Nguyen, T.T. Nguyen, N.H. Pham, J.M. Al-Kofahi, and T.N. Nguyen. Accurate and Efficient Structural Characteristic Feature Extraction for Clone Detection. In *FASE'09*, Springer-Verlag, 2009.
[11] H. Liu, Z. Ma, L. Zhang, W. Shao, Detecting duplications in sequence diagrams based on suffix trees, in: Proceedings 13th Asia-Pacific Software Engineering Conference (APSEC˝06), Bangalore, India, 2006.
[12] B. Selic. What˝s new in UML 2.0? Technical report, IBM Rational Software, April 2005.
[13] B. Hummel, E. Juergens, D. Steidl, Index-based model clone detection, in:Proceedings of 5th International Workshop on Software Clones, Honolulu,USA, 2011.