

Literature Analysis on Model based Slicing

Rupinder Singh
Thapar University Patiala
CSED

Vinay Arora
Thapar University Patiala
CSED

ABSTRACT

To understand and test a large software product is very challenging task. One way to ease this is program slicing technique that decomposes the large programs into smaller ones and another is model based slicing that decomposes the large software architecture model into smaller models at the early stage of SDLC (Software Development Life Cycle). This literature analysis presents an overview of Model based slicing, including the various general approaches and techniques used to compute slices.

General Terms

Slicing, UML Models

Keywords

Model Based Slicing, Dependency Graph, UML/OCL Constraints, Feature Based Slicing, Model Transformation

1. INTRODUCTION

In recent years, due to the increase in size and complexity of software products the importance of architectural design has been increased. The architecture of an object-oriented software system defines its high level design structure and allows an architect to reason about various properties of the system at higher level of abstraction. For this, Unified Modeling Language (UML) is best option and widely used to represent and construct the architecture of software system with the help of its various model diagrams. UML diagrams describe structural and behavioral aspects of architecture [2]. Structural models (e.g., class diagrams, object diagrams, component diagrams) are used to describe various relations among objects, such as aggregation, association, composition and generalization/specialization etc. On the other hand, the behavioral models (e.g. communication and sequence diagrams, activity diagram, state diagrams) are used to describe a sequence of actions, states and their interaction, through which a use case is realized. The task of analyzing UML Models is bit challenging since the information regarding system can be distributed across several model views.

For better visualization of architecture, impact analysis and for test case generation the properties of system architecture with slicing can be taken into account. For this various ideas, approaches and slicing techniques has been proposed by various academicians' authors and researchers. Major focus of this literature is to provide review of slicing approaches and techniques present for UML models. Section 2 provides a brief review of Slicing of UML models and various techniques and approaches used by the researches. Section 3

provides the list of tools used for model based slicing. Section 4 provides the conclusion of this literature analysis.

2. SLICING UML MODELS

Slicing UML Models is a process of decomposition to extract and identify relevant model parts or related elements across model that corresponds to user defined slicing criterion. In Model based slicing several types of model relations, and dependency such as class-class, class-operation, operation-operation, class-object, object-object, guard condition in sequence diagram , conditional predicate, control flow , data flow etc., need to be taken into account. In this work, sequence diagram has been taken into account and various approaches present till date for slicing UML diagram have been listed.

2.1 Slicing Methodologies for UML Models

2.1.1 Using Dependency Relationships

Dependency Graph is an intermediate representation step while slicing UML Models that can describe the various types of dependencies. Zhao [3] introduced the concept of architectural slicing which operates on architectural description of software system. According to the proposed architectural description there will be three types of dependencies. First is component-connector dependency where information flows from port (interface) of a component to role of a connector. Second type is connector component dependency in which information flow is from role of connector to port of component. Third type of dependency is additional dependency which can be used to represent a relation between two ports or roles within a component or connector. To compute the architectural slice a two phase algorithm has been proposed that works on SADG (software architectural dependency graph). First phase of algorithm is to compute slice 'Sg' over the SADG and second phase is to compute an architectural slice in which each element of 'Sg' is mapped to equivalent source code of description. As an extension of his previous work Zhao [4] introduced Architectural Information Flow Graph with three type of information flow arcs: Component-connector, Connector-component, internal flow arcs to apply the slicing technique on software architecture precisely.

Wang et.al [5] presented a method for slicing hierarchical automata. The importance of Wang's algorithm is its ability to remove the hierarchies and concurrent states, which are irrelevant to the properties of the hierarchical automata. The given approach was based on representing the UML state

charts by hierarchical automata for modeling dynamic aspects of software. The proposed method reduces the state space during model checking of UML state chart. The output slice proposed by technique is Extended Hierarchical automata (in which a set of dependence relations is specified after analyzing characteristics such as hierarchy, concurrency and synchronization.) instead of UML State chart models.

Wu and Yi et al. [6] developed an approach that comprises different class relationships to define dependence relations corresponding to the relations among classes. Based upon these set of dependency relations they construct a dependence graph of UML class diagram. Their proposed model can be used in two important applications: slicing the architecture and measurement of coupling between components. As their graph representation has been derived from class diagrams alone, usefulness is limited to understanding only the static aspects of a modeled system.

Kagdi et al. [7] proposed the concept of model slicing to support maintenance of software through the understanding, querying and analyzing large UML models. Kagdi developed model slices from UML class models. His approach was to extract parts of a class diagram in order to construct sub models from a given model of a system. However, class models are lacking of explicit behavioral information and represent only structural behavior. For the purpose of model slicing they define a model 'M' as directed multi graph for finite set of elements, their set of relationships, and a function that maps elements to elements via a relationship.

Langehove [8] presents an algorithm for reducing the number of interference dependencies in state charts by using the concept of slicing with concurrent states. The proposed approach considers data dependencies from the definition and use of variables that are common to parallel executing statements. The core idea of approach is to define happens-before relation between states and transitions of orthogonal regions to improving the degree of refinement in measurement of interference dependencies. He achieves this by exploiting the internal broadcasting mechanism and maintaining the state chart's execution semantics.

Bae et.al [9][10] proposed UML metamodel slicer to manage the complexity of UML metamodels which addresses to all UML diagram by modularizing metamodels into small metamodels. The proposed approach extracts diagram-specific metamodels from the UML metamodel because the diagram-specific metamodels consist of a considerable small number of elements and relationships. The UML Metamodel Slicer generates a metamodel, 'MMdt' by a given set of key elements 'KEdt'. The elements in 'KEdt' are used for identifying the model elements as slicing criteria which are relevant to the diagram type 'dt'.

Sen et al. [11] presented an approach for pruning metamodels. The proposed pruner takes input slicing criteria, i.e. classes, operations, etc of the metamodel to slice the architecture and extract all the mandatory dependencies between them. The

pruner resulted into an output slice that satisfies all the structural constraints imposed by the input metamodel.

Lallchandani et al. [12] propose a technique for constructing dynamic slices of UML models using the integrated state-based information. In order to achieve this they proposed an algorithm Architectural Model Slicing through MDG Traversal (AMSMT) that first represents a model dependency graph to collect all the information about every dependency at different states of variables. Researchers proposed an algorithm that generate the dynamic slices correspond to any slicing criteria by traversing the model dependency graph which holds all the dependency of variable. Such slices can be used for studying the impact of design changes, reliability prediction, understanding large architectures because it holds the object's state information. By using the same algorithm (AMSMT) researchers had implemented a prototype architectural slicing tool called SSUAM [13] to generate static slices for UML Architecture models. Later on, in another approach [14] they proposed a DSUAM algorithm which uses the MDG representation to compute dynamic slices. There slicing algorithm is based on traversing the edges in the MDG for any given slicing criterion. During MDG traversal, DSUAM identifies the relevant model parts from architecture. Algorithm has two phases. In first phase first phase, an MDG is constructed by extracting model information through static analysis of the UML structural and behavioral models. And in second phase MDG is traversed according to the given slicing criteria and produce desired chunk relative to condition.

Samuel and Mall [15] presented a scheme to generate slice and test cases with the help of edge marking dynamic slicing algorithm for activity diagrams. They used the flow dependency graph (FDG) which shows the dependencies among activities that arise during run time. The proposed technique uses the edge marking concept to mark the stable and unstable edges in FDG so that they can slice the graph according to slicing criteria from FDG correspond to conditional predicate present at each activity edge and can generate test cases according to them.

Samuel et al. [16] presented a methodology to generate dynamic slices and test cases with the help of UML sequence diagram. In which Message dependency graph (MDG) is constructed which represent every message as node. To identify the conditional predicates associated with messages in a sequence diagram, slicer can create dynamic slice according to the criteria.

As an extension of their previous work to generate automated test cases according to the functionality of the system at a designing part of the SDLC, they proposed an approach [17] to use slicing technique on the UML Sequence diagram. Sequence Diagram can capture time dependent sequence of interaction between different objects and components. By analyzing these relations a proper functionality of the system can be visualized and captured to generate test cases for better verification. The way to generate test data in their proposed approach was to select conditional predicate from sequence

diagram and make as a slicing criteria in the slicer while keeping all other variable constant while traversing the every node of sequence diagram until the solution is found.

Noda et.al [18] proposed a sequence diagram slicing method to visualize the object oriented program's behavior. In order to achieve this, a tool has been proposed that named as 'Reticella' which is implemented as eclipse plug-in. The proposed tool take java program as input and after analyzing, fetch the static information and draw B-model tree. The slicer extracts a slice according to user define slicer criteria from graph and Drawer convert the data sequence slice into sequence diagram with the help of Quick sequence diagram editor.

Swain et.al [19] proposed an approach to generate test cases from UML interaction diagram by using the condition slicing. In their approach they identify the message guard condition from interaction diagram and use the condition slicing to generate test cases. In the proposed approach, they first build a message dependency graph from UML Interaction diagram and then apply the conditional slicing on a predicate node of the graph by considering guard condition of message flow as a slicing criterion to compute slices and to generate test cases.

J. Kim et.al [20] proposed an approach to address the hierarchy and orthogonality problems while tracing the data dependency in slicing of UML State machine diagram. They first, constructed a control flow graph (CFG) to track every transaction and parallel flow, and then they created a hierarchy graph that represent a parent-child relationship among regions, states and behaviors of states. By utilizing CFG and hierarchy graph researchers have generated dependency graphs that represent the related functionality.

Yatapanage et al. [21] focused their work on Model Checking as fully automated technique to reduce the size of model with the help of slicing. They used Behavior Tree dependency graph (BTDG) to capture all functional requirements and dependency between components and attributes. After creating the BTDG they use the slicing criterion consists of all state-realization nodes that update the state of one of the components or attributes mentioned in the temporal logic property to reduce the parallelism and size of models in order to improve verification time in model checking techniques.

2.1.2 Using Control and Data Flow

Control and Data Flow are the important aspect of system modeling or UML models that describe the nature of every component, their behavior, and working with other components and sequential pattern of interaction. Many researchers dedicated their work to slice the models or architecture of the system into desirable small chunks. Korel et.al [22] dedicated their work on slicing the state based models, such as EFSMs (Extended Finite State Machines). As a result two types of slicing came to existence—deterministic and nondeterministic slicing. Their approach also includes a

slice reduction technique to reduce the size of a computed EFSM slice by isolating the parts of the model that may contribute to faulty behavior. Their research of slicing techniques for state machines is thoroughly based upon control and data-flow analysis. To automate the slice computation they proposed a tool that constitute of graphical editor, an EFSM executor and EFSM slicer.

Lano [23] defined that slicing can be carried out for UML state machines, using data and control flow analysis to remove elements of the machine that do not contribute to the value of a set of features in a selected state of the machine. The proposed technique of slicing by refactoring enables the models to be simplified and factored on the basis of features. Author also represents the pre and post condition relationship of the state during path predicate coverage.

Samuel et.al [24] proposed a schema known as 'CTest' that automatically generate test cases from UML communication diagram. According to schema the first step of the approach is to construct communication tree from communication diagram on the basis of data flow and control flow. After selecting the predicate from tree, tool named UTG (UML behavioral Test case Generator) transform the predicate according to 'CTest' schema to find the test data. Communication diagram has been taken as input by tool in xml format. In this approach Document Parser class parses the XML file for the message name, arguments, sequence numbers and constructs the communication tree, while Test Data Finder uses the parsed information and finds the test data in the form of a string

Julliard et.al [25] proposed an approach based on domain abstraction for generating test cases on the basis of syntactic abstraction and variable elimination with the help of model slicing. In the proposed approach source model is taken as input with set of abstract variable then reduced by syntactic abstraction followed by semantically abstraction to generate abstract model from which symbolic tests are extracted according to selection criteria. They proposed three methods for identifying the relevant variable and generating abstract model. The first one is to consider data flow dependency only. Second one uses both data-flow and control-flow dependency. Third method is to use data flow and partial control flow dependencies to find as much as possible strong relevant variables. Once the set of abstract variable 'Xa' is defined the next step is to define the Slice function that abstract the predicate P according to 'Xa'. The core idea of the work is to use the combination of syntactic and semantic abstraction or slicing to refine the result more precisely while generating the test cases.

2.1.3 Using UML/OCL Constraints

OCL allows the definition of expressions on UML models, an expression that evaluates the true or false of class invariant, or constraint. OCL expression involves several objects from one or more classes of the model. To get a starting object, we can use the keyword 'self', which denotes an object of the context

type or the method 'allInstances()', that can be used to access all objects of a given type, e.g. 'TT::allInstances()' returns a set of all objects of class TT. Given an object, OCL provides operators to read the values of its attributes and access the objects connected to it through associations (navigation). Combining these operators with arithmetic, logic and relational operators, iterators and user-defined query operations, it is possible to write complex constraints about UML models.

Shaikh et al [26] proposed a verification technique to check the correctness of model with the help of slicing. The proposed technique increases the scalability of verification by partitioning the original model into submodel. To define the binary association and inheritance relation, dependency graph and flow graph has been used in the proposed approach. To verify the satisfiability of instances in models independently structure of class diagram and OCL Constraints has been taken as input. By which slicing can be done easily to decompose the model into sub models with the help of dependency graph to extract instances of models and correct component relations.

In another approach [27][28] author proposed a tool (UOST) to enable the efficient verification of UML/OCL Class diagram with the help of model slicing technique. The tool can verify the properties of the diagram with disjoint and non-disjoint sets of slicing. Tool take the class diagram as input in XMI format with text specified OCL constraints and break the file into several slices with the help of model slicing technique after parsing the file. Researcher used the eclipse solver to translate the slices into CSP and check the existence of solution to generate the respective object diagram for satisfiable and unsatisfiable sub models with their specific invariants.

Sarna et al. [29] proposed an algorithm for automatic generation of test cases from sequence diagrams. They first transform UML sequence diagram into graphical representation named as SDG (Sequence diagram graph). They follow a graph based methodology with a depth first search algorithm to traverse the SDG and to generate test cases according to all message sequence path coverage criteria. To retrieve the information for a specification of input/output, pre and post conditions for test cases generation they use the use case template, class diagram and data dictionary and expressed in OCL.

2.1.4 Using Feature Based Criteria

Archer et al. [30] proposed a novel slicing technique on the feature model by taking cross-tree constraints into account with respect to set of features which are acting as slicing criteria. The core idea of proposed algorithm is to compute proposition formula representing the set of configuration and rules and to apply propositional logic reasoning techniques to construct an FM (representing its hierarchy, variability information, feature groups and cross-tree constraints). By extended the previous author [31] also proposed the concept

that how set of complementary set of operators like aggregate, merge and slice can provide practically and efficient support for separation of concerns from feature modeling. They defined that slicing process is both semantic and syntactic so they analyze the cross-cutting constraints to define the features that must be or cannot be sliced. In their proposed technique, the feature model and its cross-cutting constraints are first analyzed by transformation into predicates and then these predicates are transformed in a sliced feature model.

Hubaux et al. [32] proposed a slice feature diagram to design three different views of an input diagram to provide more flexibility to the configuration environment. The sliced diagram does not keep the same structure as the input diagram. The proposed approach does not consider cross-cutting constraints and is thus syntactic. The important property of the approach is that it should always lead to valid configurations but the problem can arise in the approach when features belong to more than one view or, more generally, when the selection of a feature in one view affect the selection of another feature in a concurrent view.

2.1.5 Using Model Languages

Kim [33][34] introduced the slicing technique called dynamic software architecture slicing (DSAS). In the situations where a large number of ports are present and their invocation can change the values of some variables, or the occurrence of certain events, Kim's work is very efficient there because it's able to generate a smaller number of components and connectors in each slice according to slicing criteria. In this approach software architecture is first designed by using ADL (Architecture description language) and later on mapped onto program statement as executable architecture. Dynamic slicer takes slicing criterion as input, and reads the ADL source code of the architecture to identify the information of component and connector along with the event names used in the ADL and parameter names combined with those events. The proposed algorithm filters out the events that are not relevant and passes only those which are relevant to slicing criterion and generate resulting software architecture slice as shown in Fig 2.1.

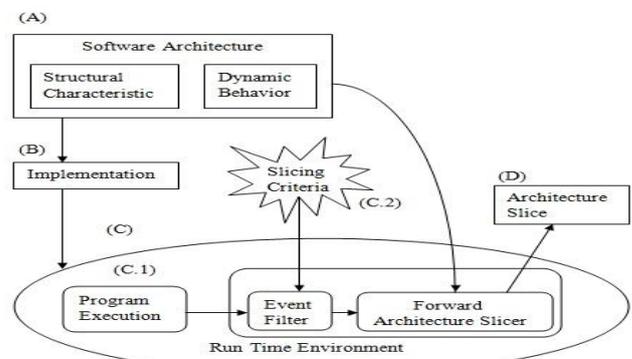


Fig 2.1 Dynamic Software Architecture Slicing Methodology Proposed by Kim [32]

Falessi et al. [35] used the concept and technique of model slicing to automate the safety inspection of system. In order to achieve this, a tool named “Safe Slicer” has been proposed that use model based slicing to enables automatic extraction of the safety-related slices (fragments) of design models. They proposed and elaborate a design methodology which ensures the traceability of links required for automated slicing. The methodology and the slicing algorithm proposed by these researchers are the basis for the Safe Slice tool. Evaluation conducted by this tool indicates that the use of design slices substantially reduces the amount of information that needs to be inspected.

Lano et al. [36] defined the technique for slicing of UML model using Model Transformation, particular for restriction of model to those parts which specifies the properties of subset within. The proposed technique use class diagrams, individual state machines and communicating sets of state machines to perform slicing of UML Model. Researcher used the client-supplier relation between different classes to form a tree structure. According to proposed technique slicing will be carried out upon class invariants and operation pre and post conditions by considering the predicates P. To slice the behavior and communicating state machines they define criteria's for a slice 'S' of a state machine 'M' are $S \prec_{syn} M$, if S has fewer elements than M. $S =_{sem} M$. This means that any analysis which concerns the value of the slice features V in the selected state s, over all paths to this state, can be performed on the slice S, and the result will also apply to state model M. their techniques focused on structure-preserving and amorphous slicing of class diagrams and state machines by using Model transformations to perform slicing, each transformation will satisfy the \prec_{syn} and $=_{sem}$ relations, resulting in a slice which also satisfies these relations compared to the original model.

Zoltán et.al [37][38] proposed dynamic backward slicing of model transformations technique with respect to program slicing. To slice the models they used model transformation language as a core of technique with the help of Dynamic Backward slicing by considering the Execution traces of program to generate final slice. The proposed technique take three inputs, the model transformation program, the model on which the MT program operates and the slicing criterion and generate the output as transformation slices and model slices. Transformation of models into MT Language is done by three consecutive processes, Graph pattern, Graph transformation rules and control language on VIATRA2 platform. The algorithm keep the record of execution traces during graph pattern calls and GT rules, to provide traceability information between source and target models and uses these traces to slice MT programs and models simultaneously according to slicing criteria.

Blouin et.al [39] [40] proposed a DSML (Domain Specific Model Language) 'Kompren' to model the model slicers for particular domain. Kompren refers to the selection of the set of classes and relations from the input metamodel expressed using an object-oriented meta-language. Using Kompren, the

MSM (Model Slicer Model) specifies the type of slicing criteria among the classes of input metamodel by domain expert that uses the 'Ecore' to describe the structure of metamodel and 'Kermeta', an action language to specify the behavior of slicer. Kompren's compiler processes the MSM and automatically generates an actual model slicer function (MSF). This MSF take as input the slicing criteria, options, constraints defined in MSMs by Domain user to produce the Model Slices.

3.TOOLS FOR MODEL BASED SLICING

Table 1: List of Tools

Year	Tool Name	Technique Used
2003	EFSM Slicing Tool	Control and Data flow analysis.
2007	UTG	Data Flow and Control Flow dependency, Communication Tree
2008	SSUAM	Model Dependency Graph.
2008	UML Slicer	MetaModel Diagram.
2009	Reticella	B-Model dependency Graph.
2011	Archlice	Model Dependency Graph.
2011	Safe Slicer	System Model Language, Traceability Links and Rules.
2012	UOST	UML + OCL Constraints.

4. CONCLUSION

From the given literature this has been listed out that for model based slicing techniques there is use of dependency relation, control and data flow, uml/ocl constraints, model language are present in literature with great emphasis on dependency relation. Hence there is a need for such technique that can reduce the effort of generation of dependency graph as intermediate state.

5. REFERENCES

- [1] Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language User Guide,” 2nd Edition, May 2005, Publisher. Addison Wesley.
- [2] Jianjun Zhao, "Slicing Software Architecture," Technical Report 97-SE-117, pp.85-92, Information Processing Society of Japan, Nov 1997.
- [3] Jianjun Zhao, “Applying slicing technique to software architectures,” In Fourth IEEE International Conference on Engineering of Complex Computer Systems, ICECCS'98, pp 87 –98, 1998.

- [4] J. Wang, Wei Dong, and Zhichang Qi, "Slicing Hierarchical Automata for Model Checking UML Statechart," Proc. Fourth Int'l Conf. Formal Eng. Methods: Formal Methods and Software Eng., pp. 435-446, Oct. 2002.
- [5] W. Fangjun and Y. Tong, "Dependence Analysis for UML Class Diagrams," J. Electronics (China), vol. 21, no. 3, pp. 249-254, May 2004, doi 10.1007/BF02687879.
- [6] H. Kagdi, J.I. Maletic, and A. Sutton, "Context-Free Slicing of UML Class Models," Proc. 21st IEEE Int'l Conf. Software Maintenance, pp. 635-638, 2005.
- [7] S. Van Langehove, "Internal Broadcasting to Slice UML State Charts: As Rich as Needed," Proc. Abstracts of the FNRS Contact Day: The Theory and Practice of Software Verification, Oct.2005.
- [8] J.H. Bae, K. Lee, and H.S. Chae, "Modularization of the UML Metamodel Using Model Slicing," Proc. Fifth Int'l Conf. Information Technology: New Generations, pp. 1253-1254, 2008.
- [9] Jung Ho Bae and Heung Seok Chae, "UMLSlicer: A tool for modularizing the UML metamodel using slicing," In 8th IEEE International Conference on Computer and Information Technology (CIT), pp.772-777, 2008.
- [10] Sagar Sen, Naouel Moha, Benoit Baudry, and Jean Marc Jézéquel, "Meta-model Pruning," In 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09), 2009.
- [11] J. Lallchandani and R. Mall, "Slicing UML Architectural Models," ACM SIGSOFT, vol.33, no.3, May 2008.
- [12] Jaiprakash T. Lallchandani, R. Mall, "Static Slicing of UML Architectural Models," Journal of Object Technology, vol. 8, no. 1, pp. 159-188, January-February 2009.
- [13] J. Lallchandani and R. Mall, "A Dynamic Slicing Technique for UML Architectural Models," IEEE Transaction on Software Engineering, Vol. 37, No. 6, NOV/DEC 2011.
- [14] Philip Samuel, Rajib Mall, "Slicing-Based Test Case Generation from UML Activity Diagrams," ACM SIGSOFT Software Engineering Notes, Vol. 34 No. 6, November 2009.
- [15] Philip Samuel, Rajib Mall, "A Novel Test Case Design Technique Using Dynamic Slicing of UML Sequence Diagrams," e-Informatics Software Engineering Journal, Vol. 2, Issue 1, 2008.
- [16] Philip Samuel, Rajib Mall and Sandeep Sahoo, "UML Sequence Diagram Based Testing Using Slicing," IEEE Indicon 2005 Conference, Chennai, 11-13 Dec. 2005.
- [17] Kunihiro Noda , Takashi Kobayashi, Kiyoshi Agusa, Shinichiro Yamamoto, "Sequence Diagram Slicing," 16th Asia-Pacific Software Engineering Conference, IEEE, 2009.
- [18] Ranjita Kumari Swain , Vikas Panthi, Prafulla Kumar Behera, "Test Case Design Using Slicing of UML Interaction Diagram," 2nd International Conference on communication, computing and security, vol.6 , pp.136-144, ELSEVIR, 2012.
- [19] Hyeon-Jeong Kim , Doo-Hwan Bae, Vidroha Debroy, W. Eric Wong, "Deriving Data Dependence from UML State Machine Diagrams," Fifth International Conference on Secure Software Integration and Reliability Improvement (IEEE), 2011.
- [20] Nisansala Yatapanage, KirstenWinter, and Saad Zafar, "Slicing behavior tree models for verification," In IFIP Advances in Information and Communication Technology, Vol. 323, pp. 125-139, 2010.
- [21] B. Korel, I. Singh, L. Tahat, and B. Vaysburg, "Slicing of State Based Models," Proc. Int'l Conf. Software Maintenance, pp. 34-43, 2003.
- [22] Kevin Lano Crest, "Slicing of UML State Machines," Proceedings of the 9th WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS (AIC '09), 2009.
- [23] Philip Samuel , Rajib Mall, Pratyush Kanth, "Automatic test case generation from UML communication diagrams," Information and Software Technology (ELSEVIER), 2007.
- [24] J. Julliard, N. Stouls, P-C. Bue, P-A. Masson, "B model slicing and predicate abstraction to generate tests," Software Quality Journal, vol. 21, pp.127-158, 2013.
- [25] Asadullah Shaikh, Robert Clarisó, Uffe Kock Wiil, and Nasrullah Memon, "Verification-driven slicing of UML/OCL models," In Proceedings of the IEEE/ACM international conference on Automated software engineering, pp. 185-194, ACM, 2010.
- [26] Asadullah Shaikh, Uffe Kock Wiil, and Nasrullah Memon, "UOST: UML/OCL aggressive slicing technique for efficient verification of models," In System Analysis and Modeling: About Models, 6th International Workshop SAM'10, pp. 173-192, 2010.
- [27] Asadullah Shaikh, Uffe Kock Wiil, and Nasrullah Memon, "Evaluation of tools and slicing techniques for efficient verification of UML/OCL class diagrams," Advances in Software Engineering, vol.18, pp 173-192, 2011.
- [28] Monalisa Sarma, Debasish Kundu, Rajib Mall, "Automatic Test Case Generation from UML Sequence Diagrams," 15th IEEE International Conference on Advanced Computing and Communications, 2007.
- [29] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France, "Slicing feature models," In 26th IEEE/ACM International Conference On Automated Software Engineering (ASE'11), IEEE/ACM, 2011.
- [30] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France, "Separation of Concerns in Feature Modeling: Support and Applications," In Aspect-Oriented Software Development(AOSD'12), ACM Press, 2012.
- [31] Arnaud Hubaux, Patrick Heymans, Pierre-Yves Schobbens, Ebrahim Khalil Abbasi, and Dirk Deridder, "Supporting multiple perspectives in feature-based configuration," Software and Systems Modeling, 2012.
- [32] T. Kim, Y.-T. Song, L. Chung, and D.T. Huynh, "Dynamic Software Architecture Slicing," Proc. 23rd Int'l Computer Software and Applications Conf., pp. 61-66, 1999.

- [33] T. Kim, Y.-T. Song, L. Chung, and D.T. Huynh, "Software Architecture Analysis: A Dynamic Slicing Approach," *J. Computer and Information Science*, vol. 1, no. 2, pp. 91-103, 2000.
- [34] Davide Falessi, Shiva Nejati, Mehrdad Sabetzadeh, Lionel Briand, and Antonio Messina, "SafeSlice: a model slicing and design safety inspection tool for SysML," In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering Conference (ESEC-13)*, ACM, 2011.
- [35] Kevin Lano and Shekoufeh K. Rahimi, "Slicing of UML Models Using Model Transformations," *Model Driven Engineering Languages and Systems, 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Lecture Notes of Computer Science, Vol. 6395*, pp. 228-242, Springer, 2010.
- [36] Zoltán Ujhelyi, Ákos Horváth, and Dániel Varró, "Towards dynamic backward slicing of model transformations," In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pp.404–407, IEEE Computer Society, 2011.
- [37] Zoltán Ujhelyi, Ákos Horváth, and Dániel Varró, "Dynamic Backward Slicing of Model Transformations," *IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012.
- [38] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, "Modeling model slicers," *Proceedings of the 14th international conference on Model driven engineering languages and systems*, 2011.
- [39] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, "Kompren Modeling and Generating Model Slicers," *Journal of Software and System Modeling*, Springer, 2012.