# Generation of Test Cases from Sliced Sequence Diagram

Manpreet Kaur
Department of CSE
CGC, Gharuan
Mohali, India

Rupinder Singh
Assistant professor of CSE
CGC, Gharuan
Mohali, India

## ABSTRACT

UML diagrams are vital design and modeling artifacts. These UML models can also be used to create test cases. In this approach, condition slicing is used and creates test cases from UML sequence diagrams. Test cases can be planned at design level of software development life cycle. But to visualize the system model or architecture is hard due to its bulky and complex structure. This methodology derives test cases of the computed slice using conditional predicate and it beneficial for sequence diagram containing number of messages. The proposed methodology also use the notion of model based slicing to compute the slice of the sequence diagram by extracting the desired chunk.

## Keywords
Software Testing, Sequence diagram, Model based slicing.

## 1. INTRODUCTION

Testing is an important failure detection technique whose major aim is to identify all defects existing in software product. Software testing is done to expose possible failures of the software. So testing becomes difficult due to raise in product sizes and complexities. Due to raised product size and complexity, UML models for that product tend to become large and complex and also have thousands of interactions between hundreds of objects. For such huge system architectures, it becomes extremely complicated to understand and analyze these models. For huge system architectures, it is very difficult to test the entire system in one pass. So there is need for some ways to reduce that effort.

One capable way to hold software architecture development is to use slicing technique. Program slicing concept was originally established by Weiser [1], is a decomposition technique which removes irrelevant program elements from the source program. A program slice contain only that content of program which may directly or indirectly affect the computed part of program at some point of interest, called a slicing criterion. The procedure required to compute program slices is known as program slicing [2]. In this context, they have proposed to use program slicing techniques to decompose large architectures into manageable portions [3].

However, in considering software architectures, the structural models (e.g. class diagrams) explain various relationships exist among classes, such as aggregation, association, composition, and generalization/specialization. In contrast, the behavioral models (e.g., communication and sequence diagrams) are used to represent a sequence of actions in an interaction which explain how the objects are interacting to complete their individual action [4]. The conventional slicing is generally executed using data and control dependency relationships present among program statements. On the other hand, to execute model based slicing, it is necessary to convert model into an appropriate intermediate representation which represents different dependence relations that may be present among classes and their sub-classes, methods, and attributes, and call sequences. This intermediate representation represents elements in an xml document. DOM parser parse the xml file for Object name, identifier, message name, message to & fro information. DOM parser uses the function DocumentBuilderFactory ( ) to generate the object of the class to parse the file. The entire information generated by parser will be stored in .txt file. Then apply slicing criteria and produce a particular slice and generate test cases corresponding to that slice. Test cases are mainly created based on the guard condition. This makes the test case generation at the early level of software development life cycle that is at design level. The main idea of model based slicing technique is to disintegrate the structure of system model into sub-models without affecting their original structure and functionality. It assists the developer to take the perfect view of system software as per their requirement.

## 2. RELATED WORK

In this section, related work is concisely explained. Today's scenario tells that researchers are interested to use UML models for test generation. However, the works of slicing UML models have been reported in the literature.
Zhao [5] introduced the concept of architectural slicing by using architectural description language ADL (e.g. Acme). In this context, there are three types of dependencies like component-connector dependency, connector component dependency, and additional dependency. They proposed a two phase algorithm to compute architectural slice which is based on SADG (software architectural dependency graph). As an extension of his previous work Zhao [2] introduced Architectural Information Flow Graph with three types of information flow arcs: Component-connector, Connector-component, internal flow arcs and in this work different ADL (e.g. Wright) is used.

J. Kim et al. [6] proposed an approach whose target is the hierarchy and orthogonality problems occur in slicing of UML State machine diagram. In their approach, first they constructed a control flow graph (CFG) and hierarchy graph. Then, they make dependency graph by using CFG and hierarchy graph which is required to show the related functionality.
Korel et al. [7] presented the slice reduction technique of slicing the EFSMs (Extended Finite State Machines) by isolating the parts of the model that may participate to faulty behavior. They present slicing such as deterministic and nondeterministic. Developer proposed a tool to automate the computation of slice that constitute of graphical editor, an EFSM executor and slicer which is based upon control and data-flow analysis.

Samuel et al. [8] presented a method that is "Ctest" to generate test cases automatically from UML communication model. They developed a tool named UTG (UML behavioral Test case Generator) transform the predicate to find the test data. This tool use communication diagram as input in xml format and DocumentParser class parses that XML file. And then constructs the communication tree, while TestDataFinder uses the parsed information and displaying the list of test cases for communication diagram.

Kagdi et al. [9] proposed method of context-free model slicing to compute slices on class models. Concrete applications of model slicing such as design understanding, fault location, and metric relevance etc. are used to address particular software maintenance questions which support the usefulness and validity of the method. The disadvantage of their approach is that, they extract slice in a very general manner. That is why class models are lacking of representing precise behavioral information and represent structural behavior only.

Lallchandani et al. [3] propose a technique for generating static and dynamic slices of UML models. They proposed an algorithm AMSMT (Architectural Model Slicing through MDG Traversal) to produce the static and dynamic architectural model slices. They proposed and implement another tool named Static Slicer for UML Architectural Models (SSUAM) [26] to implement similar algorithm. Moreover, developer proposed a DSUAM algorithm [27] in which edges of MDG is traversed according to slicing criterion. In this work, they present a tool Archlice, which computes a dynamic slice for UML architectural models by using MDG and DSUAM algorithm. This tool supports analysis of Extensible Markup Language (XML) in static and dynamic manner.

Samuel and Mall [10] presented a novel approach to generate slice and test cases by using edge marking dynamic slicing algorithm of UML activity diagrams. In this work, they used the flow dependency graph (FDG) which shows the dependencies arise during run time among activities. They mark the stable and unstable edges in FDG and they may generate dynamic slice based on slicing criteria and test cases are automatically generated with respect to each slice.

Yatapanage et al. [11] developed an approach for automatically reducing Behavior Tree models by using slicing. In their approach, they draw Behavior Tree dependency graph (BTDG) then they use the slicing criterion consisting of all state-realization nodes. Their results illustrate the improvements in execution time and memory usage that allows the verification of models for which model checking was earlier infeasible.

Lano [12] proposed a method for slicing UML state machines by refactoring the models to be simplified and factored on the origin of features. The concept of data and control dependency between states is simpler than the Korel’s concepts of transition post-domination. Shaikh et al. [13] proposed an innovative slicing technique for UML/OCL class diagrams to improve their scalability. The satisfiability of the original model can be predicted by examining if at least one sub model or all sub-models are satisfiabile. [14] [15] Later on, they proposed a tool (UOST) to allow the efficient verification of UML/OCL models by using aggressive slicing technique. The tool can verify the properties of models with disjoint and non-disjoint solutions.

Noda et al. [16] proposed a technique of sequence diagram slicing which is enable accurate slice calculation based on high-precision data dependency and may support various programs as well as exceptions and multithreading. In order to achieve this, they develop a named as “Reticella” that envision object-oriented program’s behavior and calculation slice on the Eclipse platform.

Swain et al. [17] proposed a technique which is used condition slicing and create test cases from interaction diagrams. In the proposed technique, they first build a message flow dependence graph from an ordinary sequence diagram and then apply conditioned slicing on a predicate node of the graph to compute slices and to generate test cases.

Sarma et al. [18] proposed a method to generate automatic test cases as of sequence diagrams with the help of SDG (Sequence diagram graph). They traverse the SDG and to generate test cases based on sequence path coverage criteria of all message. Archer et al. [19] proposed a novel technique of slicing on the feature model by using cross-tree constraints with respect to slicing criteria. [20]They also proposed the idea that how set of complementary set of operators (like aggregate, merge and slice) can provide practicality.

Zoltan et al. [21] [22] explain a dynamic backward slicing approach for model transformation programs and their transformed models. They focuses on the simultaneously assess data and control dependencies with the help of program slicing for model transformations. Later on, researcher presented that conversion of models into MT Language is completed by three consecutive processes (Graph pattern, Graph transformation and control language on VIATRA2 transformation language platform).

Blouin et al. [23] [24] developed the DSML Kompren language to model and generate model slicers for some DSL. They proposed a two-level generative approach where Kompren’s compiler processes Kompren models to generate an actual model without human intervention. In Kompren, the Model Slicer Model uses the ‘Ecore’ to explain the structure of metamodel and ‘Kermeta’ an action language to identify the behavior of slicer.

Falessi et al. [25] presented a technique of model slicing to automate the safety inspection of system. They developed a tool named “Safe Slicer”, which allow automatic extraction of the safety slices of design models. The methodology and the slicing algorithm of the Safe Slice tool that ensures the traceability of links required for automated slicing.

## 3. OVERVIEW OF PROPOSED METHODOLOGY

The proposed work involves the slicing of sequence diagram and generates test cases of that slice to ease the software visualization by using conditional predicate. In the proposed methodology, following steps has been followed:

3.1 Creation of Sequence diagram from a user requirement specification. There are several software tools such as Visual paradigm for UML, Rational rose and Magic-draw etc. to generate diagram.

3.2 Produce XML code from the specified sequence diagram. Visual paradigm for UML 10.2 version has in-built functionality to export XML.

3.3 Java API Document Object Model (DOM) parser used for parsing XML code and generating an output file containing Object name, identifier, message name, message to & fro information with .txt extension.

3.4 Apply slicing criteria which is java program act as slicer to the file obtained from step 3 and producing the slice in a separate .txt file.

3.5 Converting object id with relative object name among which message is passing so that information can be recover easily (it only deal with sliced part).

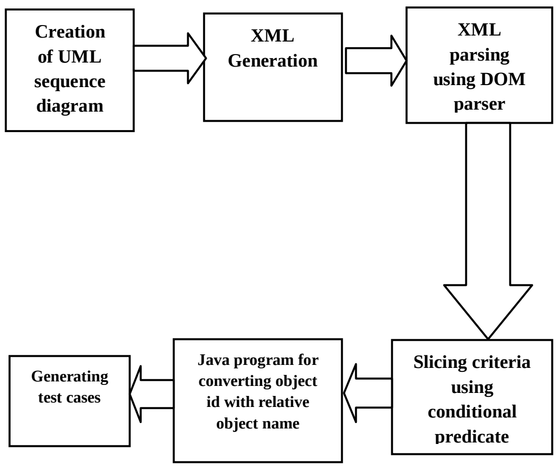

Creation of UML sequence diagram
XML Generation
XML parsing using DOM parser
Slicing criteria using conditional predicate
Java program for converting object id with relative object name
Generating test cases

**Fig 1: Overview of Proposed Methodology**

3.6 Test cases are generated of the computed slice of information obtain in step5. Test cases are generated corresponding to the conditional coverage.

## 4. IMPLEMENTATION

After evaluating the literature survey of software testing, program and architectural slicing techniques, software visualization and UML (unified modeling language), the result says that slicing UML diagrams is one of the most important area in which work can be extended. And also analyze that the slicing sequence diagram has no consolidate technique to extract the point of interest from architecture of software to ease the software visualization that uses conditional predicate for finding out a relative slice.

Consider an example UML sequence diagram as shown in fig 2. In the example there are ten objects that are interacting with each other by message passing using guard condition. There are some variable (such as x, y, z etc) and constants (such as 1,2,3 etc) are used in guard condition. To illustrate this methodology, explaining the generation of chunk or information with respect to slicing criteria.

To extract relative chunk and information from UML Sequence diagram following technique has been proposed:

4.1 Creation of Sequence diagram from a user requirement specification. There are several software tools such as Visual paradigm for UML, Rational rose and Magic-draw etc. to generate diagram.

4.2 Next step is to create XML from the specified UML Sequence diagram. Visual paradigm for UML 10.2 version provides the in-built functionality to export the diagrams into XML format.

As shown in Fig 3, XML document represents all the information regarding sequence diagram like the object name with distinct ids, the messages which they are using to transfer the data or to call the object of other classes, their attributes, etc. The purpose of converting the UML diagram into XML file is yield platform independency.

4.3 Document Object Model (DOM) parser parse the XML code and generate an output file with .txt extension containing Object name, identifier, message name, message to & fro information.

4.4 Apply slicing criteria that is a .java program which act as slicer on output of step 3 for getting the relative/required chunk of information in a separate .txt file. Slicer will ask user to define the slicing criteria at run time to generate the chunk/slice as per specified requirements. In this example user define the ‘a’ variable as slicing criteria.

4.5 Converting object id with relative object name among which message is passing so that information can be recover easily.

4.6 Test cases are generated of the computed slice of information obtain in step5 as shown in figure 4. Test cases are generated corresponding to the conditional coverage.

## 5. CONCLUSION AND FUTURE WORK

This work presented a way to generate test cases of the sliced sequence diagram. This methodology predominantly uses the visual paradigm for UML 10.2 for generating diagram. And discover the conditional predicates linked with messages in the sequence diagram and create slice with respect to each conditional predicate. In this work, test cases are generated manually with respect to each constructed slice by satisfying slice condition. The slicing approach was beneficial, when the number of messages in the sequence diagram is large. One needs to consider only the slice for finding test cases instead of entire sequence diagram. If one can knows the location of error then it becomes a huge simplification and saves lots of time and resources. In this technique, test cases are manually created that can be explored further by making it automatically and generate test cases by slicing of combination of any two models.

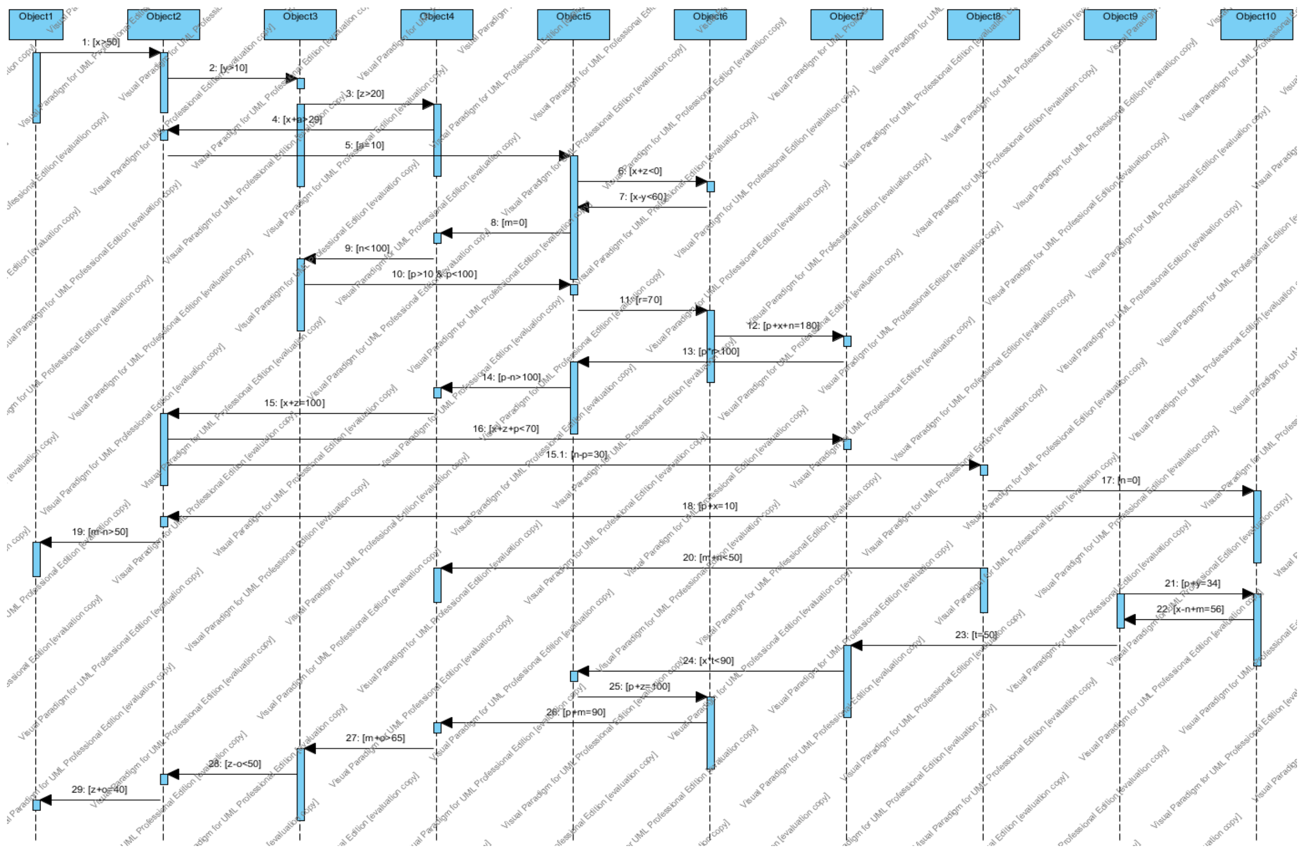
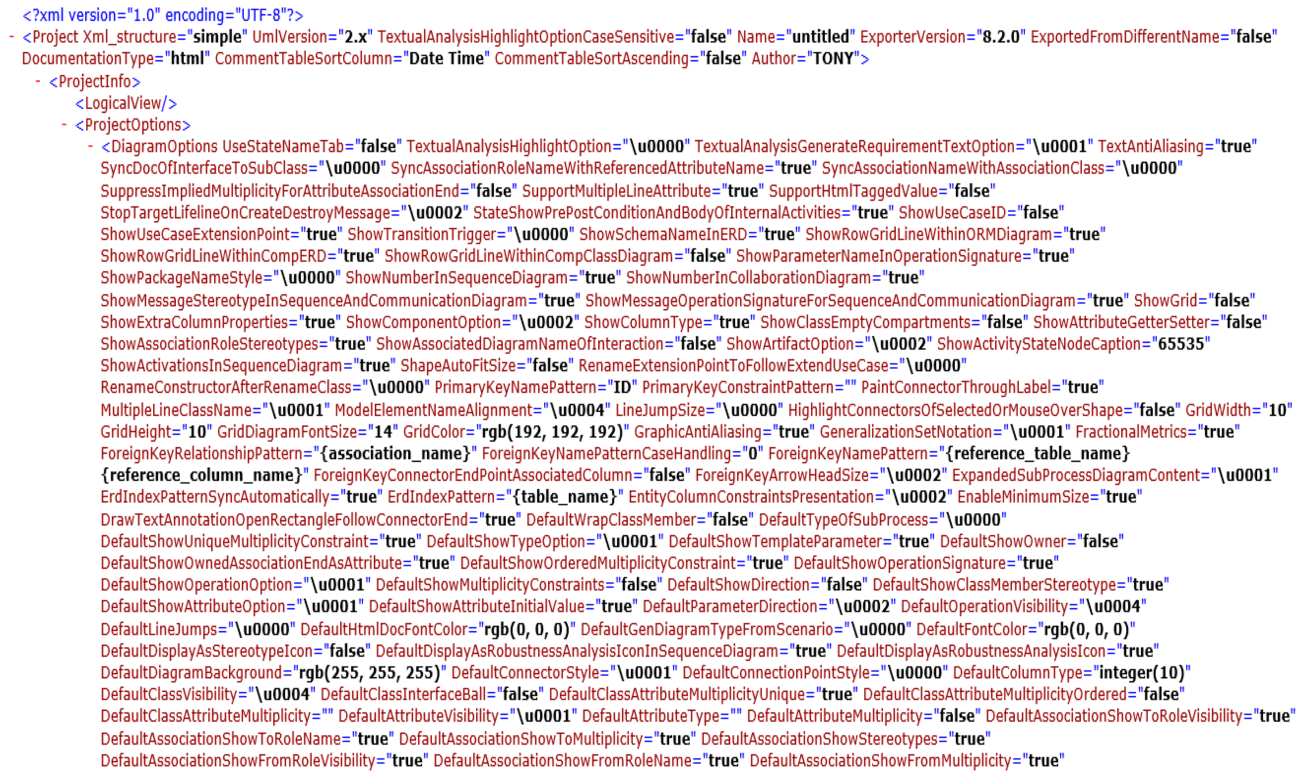
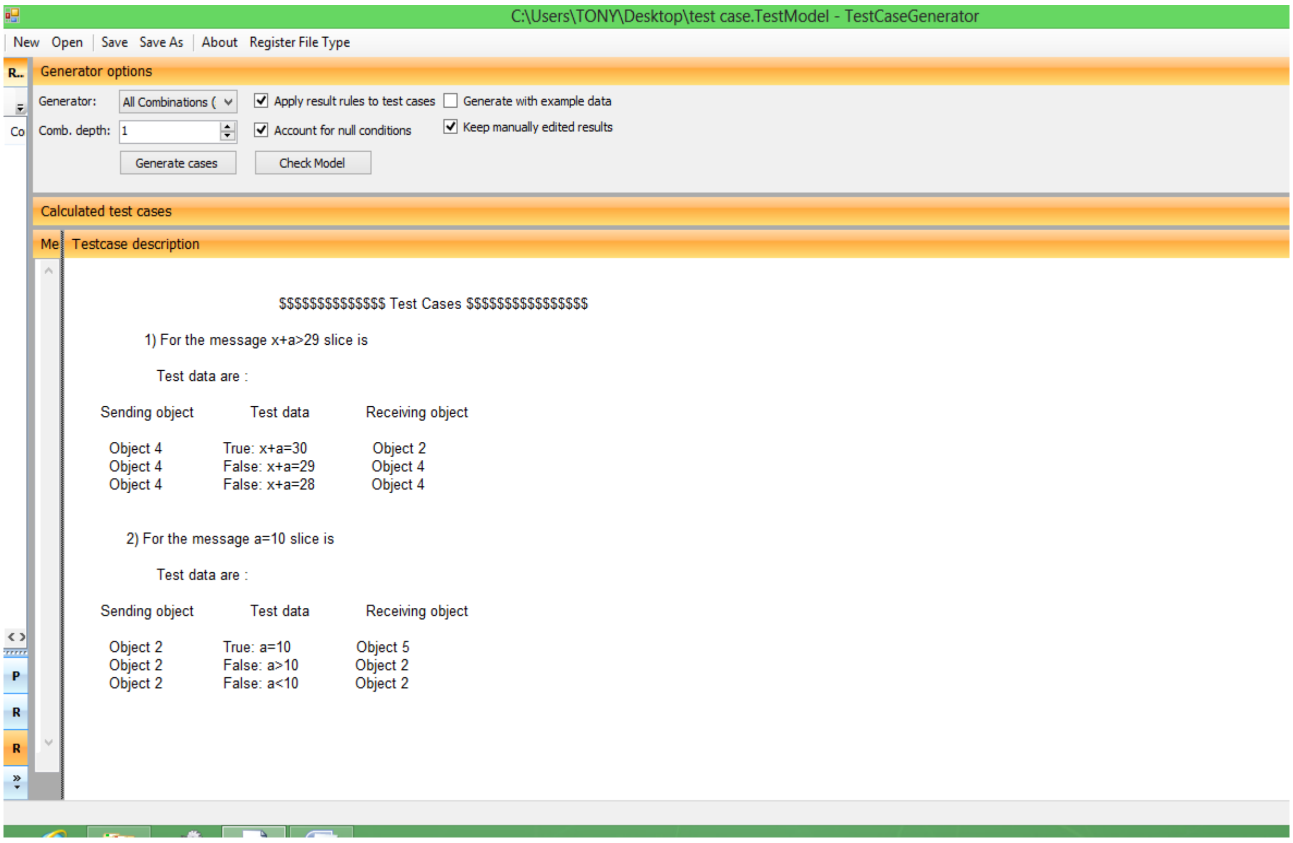


**Fig 2: Example Sequence diagram**



```
<?xml version="1.0" encoding="UTF-8"?>
- <Project Xml_structure="simple" UmlVersion="2.x" TextualAnalysisHighlightOptionCaseSensitive="false" Name="untitled" ExporterVersion="8.2.0" ExportedFromDifferentName="false"
  DocumentationType="html" CommentTableSortColumn="Date Time" CommentTableSortAscending="false" Author="TONY">
  - <ProjectInfo>
      <LogicalView/>
    - <ProjectOptions>
      - <DiagramOptions UseStateNameTab="false" TextualAnalysisHighlightOption="\u0000" TextualAnalysisGenerateRequirementTextOption="\u0001" TextAntiAliasing="true"
          SyncDocOfInterfaceToSubClass="\u0000" SyncAssociationRoleNameWithReferencedAttributeName="true" SyncAssociationNameWithAssociationClass="\u0000"
          SuppressImpliedMultiplicityForAttributeAssociationEnd="false" SupportMultipleLineAttribute="true" SupportHtmlTaggedValue="false"
          StopTargetLifelineOnCreateDestroyMessage="\u0002" StateShowPrePostConditionAndBodyOfInternalActivities="true" ShowUseCaseID="false"
          ShowUseCaseExtensionPoint="true" ShowTransitionTrigger="\u0000" ShowSchemaNameInERD="true" ShowRowGridLineWithinORMDiagram="true"
          ShowRowGridLineWithinCompERD="true" ShowRowGridLineWithinClassDiagram="false" ShowParameterNameInOperationSignature="true"
          ShowPackageNameStyle="\u0000" ShowNumberInSequenceDiagram="true" ShowNumberInCollaborationDiagram="true"
          ShowMessageStereotypeInSequenceAndCommunicationDiagram="true" ShowMessageOperationSignatureForSequenceAndCommunicationDiagram="true" ShowGrid="false"
          ShowExtraColumnProperties="true" ShowComponentOption="\u0002" ShowColumnType="true" ShowClassEmptyCompartments="false" ShowAttributeGetterSetter="false"
          ShowAssociationRoleStereotypes="true" ShowAssociatedDiagramNameOfInteraction="false" ShowArtifactOption="\u0002" ShowActivityStateNodeCaption="65535"
          ShowActivationsInSequenceDiagram="true" ShapeAutoFitSize="false" RenameExtensionPointToFollowExtendUseCase="\u0000"
          RenameConstructorAfterRenameClass="\u0000" PrimaryKeyNamePattern="ID" PrimaryKeyConstraintPattern="" PaintConnectorThroughLabel="true"
          MultipleLineClassName="\u0001" ModelElementNameAlignment="\u0004" LineJumpSize="\u0000" HighlightConnectorsOfSelectedOrMouseOverShape="false" GridWidth="10"
          GridHeight="10" GridDiagramFontSize="14" GridColor="rgb(192, 192, 192)" GraphicAntiAliasing="true" GeneralizationSetNotation="\u0001" FractionalMetrics="true"
          ForeignKeyRelationshipPattern="{association_name}" ForeignKeyNamePatternCaseHandling="0" ForeignKeyNamePattern="{reference_table_name}
          {reference_column_name}" ForeignKeyConnectorEndPointAssociatedColumn="false" ForeignKeyArrowHeadSize="\u0002" ExpandedSubProcessDiagramContent="\u0001"
          ErdIndexPatternSyncAutomatically="true" ErdIndexPattern="{table_name}" EntityColumnConstraintsPresentation="\u0002" EnableMinimumSize="true"
          DrawTextAnnotationOpenRectangleFollowConnectorEnd="true" DefaultWrapClassMember="false" DefaultTypeOfSubProcess="\u0000"
          DefaultShowUniqueMultiplicityConstraint="true" DefaultShowTypeOption="\u0001" DefaultShowTemplateParameter="true" DefaultShowOwner="false"
          DefaultShowOwnedAssociationEndAsAttribute="true" DefaultShowOrderedMultiplicityConstraint="true" DefaultShowOperationSignature="true"
          DefaultShowOperationOption="\u0001" DefaultShowMultiplicityConstraints="false" DefaultShowDirection="false" DefaultShowClassMemberStereotype="true"
          DefaultShowAttributeOption="\u0001" DefaultShowAttributeInitialValue="true" DefaultParameterDirection="\u0002" DefaultOperationVisibility="\u0004"
          DefaultLineJumps="\u0000" DefaultHtmlDocFontColor="rgb(0, 0, 0)" DefaultGenDiagramTypeFromScenario="\u0000" DefaultFontColor="rgb(0, 0, 0)"
          DefaultDisplayAsStereotypeIcon="false" DefaultDisplayAsRobustnessAnalysisIconInSequenceDiagram="true" DefaultDisplayAsRobustnessAnalysisIcon="true"
          DefaultDiagramBackground="rgb(255, 255, 255)" DefaultConnectorStyle="\u0001" DefaultConnectionPointStyle="\u0000" DefaultColumnType="integer(10)"
          DefaultClassVisibility="\u0004" DefaultClassInterfaceBall="false" DefaultClassAttributeMultiplicityUnique="true" DefaultClassAttributeMultiplicityOrdered="false"
          DefaultClassAttributeMultiplicity="" DefaultAttributeVisibility="\u0001" DefaultAttributeType="" DefaultAttributeMultiplicity="false" DefaultAssociationShowToRoleVisibility="true"
          DefaultAssociationShowToRoleName="true" DefaultAssociationShowToMultiplicity="true" DefaultAssociationShowStereotypes="true"
          DefaultAssociationShowFromRoleVisibility="true" DefaultAssociationShowFromRoleName="true" DefaultAssociationShowFromMultiplicity="true"
```

**Fig 3: XML file of Sequence diagram**


C:\Users\TONY\Desktop\test case.TestModel - TestCaseGenerator
New Open Save Save As About Register File Type
Generator options
Generate cases
Check Model
Calculated test cases
Testcase description

Generator: All Combinations (
Comb. depth: 1
☑ Apply result rules to test cases
☐ Generate with example data
☑ Account for null conditions
☑ Keep manually edited results

```
$$$$$$$$$$$$$$ Test Cases $$$$$$$$$$$$$$$$

1) For the message x+a>29 slice is

Test data are :

Sending object     Test data          Receiving object

Object 4           True: x+a=30       Object 2
Object 4           False: x+a=29      Object 4
Object 4           False: x+a=28      Object 4

2) For the message a=10 slice is

Test data are :

Sending object     Test data          Receiving object

Object 2           True: a=10         Object 5
Object 2           False: a>10        Object 2
Object 2           False: a<10        Object 2
```

**Figure 4: Generation of Test Cases corresponding to the computed slice**

## 6. REFERENCES

[1] Mark Weiser, "Program slicing",1981. Proceedings of the 5th International Conference on Software Engineering, IEEE Computer Society Press, pages 439–449, March 1981.

[2] Jianjun Zhao, “Applying slicing technique to software architectures,” 1998. In Fourth IEEE International Conference on Engineering of Complex Computer Systems, pp.87 –98.

[3] J.T. Lallchandani and R. Mall, “Slicing UML architectural models,” 2008. ACM SIGSOFT Software Engineering Notes, vol.33, No.3, pp. 1–9.

[4] Grady Booch, Ivar Jacobson & James Rumbaugh, “OMG Unified Modeling Language Specification”, 1998. Publisher: Addison Wesley, Version 1.3, First Edition: October 20, 1998.

[5] Jianjun Zhao, "Slicing Software Architecture," Nov 1997. Technical Report 97-SE-117, pp.85-92, Information Processing Society of Japan.

[6] Hyeon-Jeong Kim , Doo-Hwan Bae, Vidroha Debroy, W. Eric Wong, “Deriving Data Dependence from UML State Machine Diagrams,”2011. In Proceeding 5th IEEE International Conference on Secure Software Integration and Reliability Improvement, pp.118-126.

[7] B. Korel, I. Singh, L. Tahat, and B. Vaysburg, “Slicing of State Based Models”, 2003. In Proceeding of International Conference of Software Maintenance, pp.34-43.

[8] [8] “Automatic test case generation from UML communication diagrams”, 2007. Information and Software Technology (ELSEVIER), vol.44, No. 2, pp.158-171.

[9] H. Kagdi, J.I. Maletic, and A. Sutton, “Context-Free Slicing of UML Class Models”, 2005. In Proceeding of 21st IEEE International Conference on Software Maintenance, pp. 635-638.

[10] Philip Samuel, Rajib Mall, “Slicing-Based Test Case Generation from UML Activity Diagrams,” 2009. ACM SIGSOFT Software Engineering Notes, vol. 34, No. 6.

[11] Nisansala Yatapanage, Kirsten Winter and Saad Zafar, “Slicing behavior tree models for verification”, 2010. In IFIP Advances in Information and Communication Technology, pp.125–139.

[12] Kevin Lano Crest, “Slicing of UML State Machines”, 2009. In Proceedings of the 9th WSEAS International Conference on Applied Informatics and Communications, pp.63-69.

[13] A. Shaikh, R. Clarisó, U.K. Wiil, and N. Memon., “Verification-driven slicing of UML/OCL models”, 2010. In Proceedings of the IEEE/ACM International Conference on Automated software engineering, pages 185–194.

[14] Asadullah Shaikh, Uffe Kock Wiil, and Nasrullah Memon, "UOST: UML/OCL aggressive slicing technique for efficient verification of models", 2010. In 6th International Workshop on System Analysis and Modeling, pp. 173–192.

[15] Asadullah Shaikh, Uffe Kock Wiil, and Nasrullah Memon, "Evaluation of tools and slicing techniques for efficient verification of UML/OCL class diagrams", 2011. Advances in Software Engineering, vol.18, pp 173-192.

[16] Kunihiro Noda, Takashi Kobayashi, Kiyoshi Agusa, Shinichiro Yamamoto, "Sequence Diagram Slicing", 2009. In Proceeding of 16th Asia-Pacific Software Engineering Conference, IEEE, pp.291-298.

[17] Ranjita Kumari Swain , Vikas Panthi, Prafulla Kumar Behera, "Test Case Design Using Slicing of UML Interaction Diagram", 2012. In Proceeding 2nd International Conference on communication, computing and security, Elsevier, pp.136-144.

[18] Monalisa Sarma, Debasish Kundu, Rajib Mall, "Automatic Test Case Generation from UML Sequence Diagrams,"2007. 15th IEEE International Conference on Advanced Computing and Communications, pp. 60-65.

[19] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France, "Slicing feature models," 2011. In Proceeding 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 424-427.

[20] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France, "Separation of Concerns in Feature Modeling: Support and Applications," 2012. In Proceedings of the Aspect-Oriented Software Development (AOSD'12), pp.1-12, ACM, March 2012.

[21] Zoltán Ujhelyi, Ákos Horváth, and Dániel Varró, "Towards dynamic backward slicing of model transformations," 2011. In Proceeding 26th IEEE/ACM International Conference on Automated Software Engineering, pp.404–407.

[22] Zoltán Ujhelyi, Ákos Horváth, and Dániel Varró, "Dynamic Backward Slicing of Model Transformations", 2012. In Proceeding IEEE 5th International Conference on Software Testing, Verification and Validation, pp. 1-10.

[23] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, "Modeling model slicers," 2011. In Proceedings of the 14th IEEE/ACM International conference on Model driven engineering languages and systems, pp.62-76.

[24] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, "Kompren Modeling and Generating Model Slicers," 2012. Journal of Software and System Modeling, Springer.

[25] Davide Falessi, Shiva Nejati, Mehrdad Sabetzadeh, Lionel Briand, and Antonio Messina, "SafeSlice: A model slicing and design safety inspection tool for SysML", 2011. In Proceeding 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference.

[26] Jaiprakash T. Lallchandani, R. Mall, "Static Slicing of UML Architectural Models", 2009. Journal of Object Technology, vol. 8, No. 1, pp.159-188.

[27] J. Lallchandani and R. Mall, "A Dynamic Slicing Technique for UML Architectural Models", 2011. IEEE Transaction on Software Engineering, vol. 37, No. 6.