2014 IEEE International Conference on Software Testing, Verification, and Validation Workshops

Extracting the Combinatorial Test Parameters and Values from UML Sequence Diagrams

Preeti Satish, Arinjita Paul, Krishnan Rangarajan
Department of Computer Science and Engineering
Dayananda Sagar College of Engineering
Bangalore, India
preetisatish8@gmail.com, arinjita.paul@gmail.com, krishnanr1234@gmail.com

Abstract— **In the current practice, the Combinatorial Test Design Model (CTDM) is designed by the test designers manually leveraging their experience in testing. Their involvement, perception, domain knowledge and testing proficiency are needed to analyze the requirements document and design the test model. Till date we know of no automated method that has eased the process of deriving the Combinatorial Test Design Model.**

Requirements document and analysis artifacts like UML activity diagrams and sequence diagrams hold information on parameters, values and constraints of the underlying CTDM. Our research focus is to develop a tool that assists test designers in coming up with the CTDM. This paper presents an approach to extract CTDM related information such as parameters and values from sequence diagrams. Our key contribution in this paper includes proposing a rule-based method for identifying the model elements from the sequence diagrams with the supporting rules and extraction algorithms. The rules have been applied onto individual sequence diagrams and results qualitatively discussed based on the general understanding of the requirements.

Keywords— combinatorial testing; sequence diagram; test design model;

I. INTRODUCTION

The key to test design of an application is identifying the input parameters and the associated input space from where their values are drawn. Interaction between the values of the different parameters can trigger faults and cause an application to fail. Combinatorial Testing (CT) is a testing method that focuses on covering combinations of values of these parameters. In the entire CT process, deriving the Combinatorial Test design model (CTDM) is a very fundamental and an important prerequisite [16]. CTDM consists of test parameters, parameter values and constraints involved between the parameters and their values. Further the parameters can be classified as input and output parameters and values as assumed values, invalid value, computed values, out of range values etc. In the contemporary, the CTDM is modeled by test designers without much tool support. Their involvement, perception and testing proficiency is needed to analyze the requirements documents and design manuscripts. Till date we know of no automated method that has eased the process of deriving the CTDM. The extensive survey done on CT [16] reveals that modeling the SUT for CT is still an open research issue. Finding parameter and values is a creative process [8]. The key ideas on how to identify parameters and values from requirements are given in [6]. Deriving the CTDM elements from activity diagrams is explored and reported in our previous work [26]. Recognizing further need for increased automation and improved results, our interest lies in investigating the use of analysis artefacts like UML sequence diagrams as additional source of inputs for deriving the CTDM. Our investigation reveals that though automation is very desirable, the nature of the input sequence diagrams is such that complete derivation of CTDM from sequence diagrams alone may not be possible. In our initial study, we did not find constraints showing up directly in the sequence diagrams. Identifying CTDM elements from sequence diagrams is also quite complex because the sequence diagram can be drawn in diverse ways, from an abstract level to most detailed level, based on the style the designer adopts. The same sequence can be expressed in multiple ways using multiple sequence diagram constructs like guard condition, combined fragments and interaction use. Automated analysis of sequence diagrams can help to identify many of the parameters, values of the underlying CTDM and this input on CTDM can be fused with similar information derived from analyzing requirements document and other analysis artefacts like activity diagrams[26] etc and incorporated in a tool that assists test designers in coming up with the CTDM.

This paper presents a novel approach to extract CTDM related information from sequence diagrams. Our key contribution in this paper includes proposing a rule-based method for identifying the model elements from the sequence diagrams with the accompanying analyzer tool (UML Sequence diagram Analyzer & Modeler) that extracts the design elements. These rules have been applied independently onto individual sequence diagrams and results verified based on the general understanding of the requirements.

The rest of the paper is organized as follows: Section 2 briefly covers the related work. Section 3 sheds light on our approach, section 4 explains the implementation details, along with the observations that are amenable for automated analysis and Section 6 includes concluding comments and possible future work.

978-0-7695-5194-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICSTW.2014.11

II. Related Work

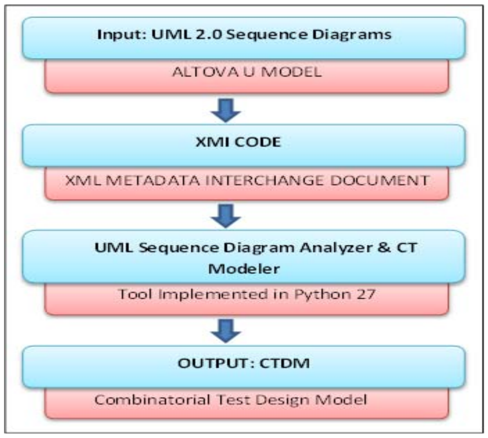
We briefly discuss the related work from two perspectives, namely: usage of sequence diagram for test case generation and combinatorial test model identification.

Zoltan et al. [27] has surveyed and have given a detailed picture on 13 semantics of UML 2.0 sequence diagrams and how they differ. The Object Management Group (OMG) specification [31] gives a basic awareness on how sequence diagram semantics work. The author explains the usage of formal semantics to interpret the sequence diagrams in practice. Many published papers address generation of test cases from sequence diagrams as applicable to general software testing. Samuel et al. [7] create Message Dependency Graph (MDG), an intermediate form from sequence diagrams for further analysis. For each condition on the sequence diagram, slices are created from MDG using edge marking dynamic slicing method. Based on these slices, test cases are generated for cluster level testing. Nayak et al. [2] present an automatic approach to synthesize the test data with the information rooted from sequence diagram, class diagram and OCL constraints and map it onto an intermediate form called as Structured Composite Graph (SCG). Test specifications are then generated from SCG and in turn test data is generated for each specification using a constraint solving system. Cartaxo et al. [15] propose a method to transform a sequence diagram into a Labelled Transition System (LTS) and obtain test cases by traversing the LTS using Depth First Search (DFS).

A detailed widespread survey on CT is done by Nie et al. [16] covering all aspects of CT, from test modeling to the applications of CT. Heuristics to find the factors, levels and constraints are given by Krishnan et.al. [6]. The basis of category partition method (CPM) introduced by Ostrand et al. [28] is to split the input domain into categories and choices. Grochtmann and Grimm [8] used the classification Tree Method (CTM) to segregate the input domain into classifications and classes and further model it in tree structure. Borazjany et al. [17] propose an input space modeling methodology using two steps, Input Structure Modeling (ISM) and Input parameter Modeling (IPM). Grindal & Offutt et al. [20] presents a method for CT modeling known as Input Parameter Model (IPM). Segall et al. [3] enumerate the frequently occurring correctness, completeness and redundancy issues and hence guiding the testers.

III. Our Approach

The paper presents a rule based semi-automated approach [26] to derive the information related to CTDM from the UML 2.0 sequence diagrams. This reported work has been implemented by us in the in-house tool UML sequence diagram analyzer and CT modeler. Fig.1 shows the sequence of steps in our approach, which takes UML 2.0 sequence diagrams as input in Step-1 and outputs information relating to CTDM like parameters and associated values in the final step. We have experimented this approach on individual sequence diagrams created by us from analyzing few requirements documents and few others sourced from internet. The sequence diagrams [10] [22] are used mainly to show the interactions between objects in time sequential order to achieve the functional requirements. The diagram conveys this information along the horizontal and vertical dimensions. The vertical dimension is from top to down, and shows the time sequence of messages/calls as they occur, and the horizontal dimension is from left to right and shows the object instances that the messages are sent to. The messages exchanged can be synchronous, asynchronous or return messages. UML 2.0 sequence diagrams supports a collection of fragments, such as, choice operator's alt, opt and break, repetition operator loop and concurrencies par which are executed under specific named conditions. The relevant sequence diagrams are created in Altova UModel [14] tool. Altova UModel [14] is an UML tool for software modeling and application development. In Step-2 of our approach, the export feature of Altova U Model is used and the information in the diagram is converted into an XMI code. XML Metadata Interchange (XMI) [19] is an open standard file format for storing and exchanging the metadata information. The exported XMI file contains the UML sequence diagram information in document form which forms the input for our tool. In Step-3 of our approach, the UML sequence diagram analyzer and CT modeler tool is used to execute the formulated rules on the XMI code and extract the CTDM elements, which is the desired output.


Input: UML 2.0 Sequence Diagrams
ALTOVA U MODEL
XMI CODE
XML METADATA INTERCHANGE DOCUMENT
UML Sequence Diagram Analyzer & CT Modeler
Tool Implemented in Python 27
OUTPUT: CTDM
Combinatorial Test Design Model
Fig. 1. Steps in deriving CTDM from sequence diagram

As in our previous work of deriving the CTDM from activity diagram [26] we have used a rule based approach in this work as well. We analyzed the sequence diagram constructs set [10] [22] and found that the synchronous message calls and combined fragments alt, opt, break and loop with guard conditions are the constructs that are useful in finding CTDM related information. We also noticed that other sequence diagram constructs did not reveal the CTDM related information directly. We have identified rules specific to synchronous message calls and combined fragments alt, opt, break and loop with guard conditions constructs. For each construct, we have identified three rules, one rule each for parameter identification, value identification and for linking the parameters and values. We have devised algorithms for parsing the XMI file to apply these rules and extract relevant information. We classify sequence diagrams for rule applications as follows:

A. *Simple sequence diagrams.* Simple sequence diagrams contain just the lifelines and messages. It shows a series of interactions between the objects or between the user and the system [32].

1) *Simple Sequence diagrams with synchronous messages.*
2) *Sequence diagrams with customer console*
3) *Synchronous messages with argument passing*.

B. *Sequence diagrams with alt combined fragment.*

1) *Alt with guard conditions without relational operator*
2) *Alt with guard conditions with relational operators*

C. *Sequence diagrams with opt, break and loop.*

1) *Sequence Diagram with opt Combined Fragment*
2) *Sequence Diagram with break Combined Fragment*
3) *Sequence Diagram with loop Combined Fragment*

We discuss the rules and the corresponding algorithms for the above types in section IV.

IV. Rules and Algorithms

A. Simple Sequence Diagrams

1) Simple Sequence Diagrams with synchronous messages: Synchronous messages are complete messages passed between objects. They are denoted by a solid arrowhead as shown in Fig.2. There will be a reply message for every sent message. The caller object cannot continue processing further until it receives a reply. Hence this classification helps in identifying the parameters and their respective values directly.

a) Rule and Algorithm for parameter extraction:

Rule name:Syncmsg_param.

Rule Narration: The synchronous message passed by the caller as shown in Fig.2 is likely to form the parameter and the return message the value. The algorithm for syncmsg_param rule is shown in Fig.3.

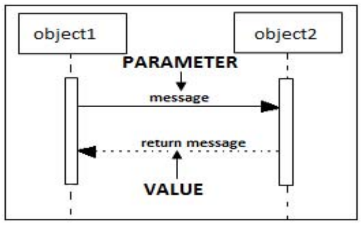


Fig. 2. Generic depiction of a sequence diagram showing synchronous message.

```
Step 1: Begin
Step 2: Open the XMI file in read mode
Step 3: Search a message of type reply,
        a) Find the object lifeline from where it was
           sent. This forms the receiver lifeline.
        b) Identify the synchronous message received
           immediately above the reply message at this
           receiver lifeline.
Step 4: For the same reply message,
        a) Determine the object lifeline where it was
           received. This forms the sender lifeline.
        b) Verify if the synchronous message was sent
           from the same.
Step 5: If true, display the value of the attribute “name" of
        the synchronous message node as parameter.
Step 6: END
```

Fig. 3. Algorithm “Syncmsg_param”.

b) Rule and Algorithm for value extraction:

Rule name: Syncmsg_value.

Rule Narration: In synchronous message passing, the return message sent back to the sender within the same activation bar as shown in Fig.2 is likely to provide the value.

Algorithm: The Syncmsg_val algorithm is similar to the syncmsg_param algorithm. In essence repeat the first four steps, and if found true, display the value of the attribute "name" of the reply message node as the value.

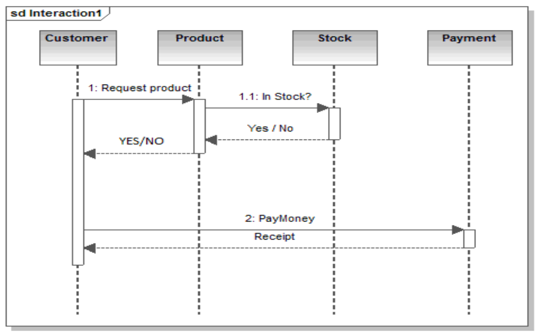
c) Rule and Algorithm for Linking the parameters and their values .

Rule name: Syncmsg_LinkPV

Rule Narration: The synchronus message from the sender and its corresponding reply from the reciever within the same activation bar are likely to form the parameter and its associated value respectively.

Algorithm:The first 4 steps of the algorithm Syncmsg_LinkPV is similar to the syncmsg_param algorithm. In essence we need to repeat the first four steps, and if found true display the value of the attribute "name" of the synchronous and its reply message node as the parameter and its value respectively.

d) An illustration of the approach: Fig.4(a) shows the sequence diagram relating to Stock Payment,where a customer enquires if a product is in stock and makes payment. Fig.4(b) shows the suggested CTDM elements obtained from applying our approach. Fig.4(c) shows the manually interpreted refinement of Fig.4(b) on manual analysis of the sequence diagram in Fig.4(a), where in the paymoney parameter is ignored.



(a)

Parameter	Request product	In stock?	Paymoney
Value 1	YES	yes	reciept
Value 2	NO	no	

(b)

Parameter	Product Name	Stock Status
Value 1	Name of product available	available
Value 2	Name of product not available	Not available

(c)

Fig. 4. Stock Payment case study (a) Sequence diagram (b) Suggested automated results (c) Manually refined and verified CTDM elements.

2) Simple Sequence Diagrams with Synchronous Messages Involving Customer Console: The reason a Software Under Test (SUT) exists is to process its inputs and inputs are the factors that has an influence on the test run. Hence, the SUT input variables should be considered as test parameters [30].Our observation shows that since customer console is the user object external to the system and provides input to the system as shown in Fig.5, it increases the degree of confidence of identifying the CTDM elements. As there is no standard naming convention to recognize a customer console object from sequence diagrams, we provide the customer console object's name as an input from the test designer to our tool.

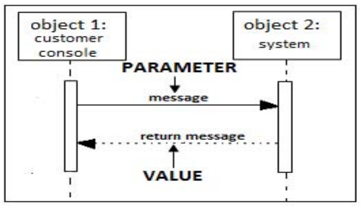


Fig. 5. Generic fragment of a sequence diagram showing customer console.

a) Rule and algorithm for parameter extraction :

Rule name: CustomerConsole_param

Rule narration:The outgoing message from the customer console object towards the system object lifeline are parameters. The algorithm is shown in Fig.6.

```
Step 1: Begin
Step 2: Open the XMI file in read mode
Step 3: Obtain the object name of the customer
        console from the test designer.
Step 4: Find a message passed from the customer
        console lifeline to system object lifeline.
Step 5: Display the value of the attribute "name" of
        the message as a parameter.
Step 6: End
```

Fig. 6. Algorithm "CustomerConsole_param".

b) Rule and algorithm for value extraction:

Rule name: CustomerConsole_value

Rule Narration: The incoming messages towards the customer console object from the system object lifeline are values. The algorithm is shown in Fig.7.

```
Step 1: Begin
Step 2: Open the XMI file in read mode
Step 3: Obtain the object name of the customer
        console from the test designer.
Step 4: Find a message passed to the customer console
        lifeline from the system object lifeline.
Step 5: Display the value of the attribute
        "name" of the message as a value.
Step 6: End
```

Fig. 7. Algorithm "CustomerConsole_value".

c) Rule and algorithm for Linking the parameters and their values:

Rule name: CustomerConsole_LinkPV

Rule Narration: The outgoing messages from the customer console object to the system are parameters, and its reply message from the system back to console are the associated values. The algorithm is shown in Fig.8.

```
Step 1:  Begin
Step 2:  Open the XMI file in read mode
Step 3:  Obtain the object name of the customer
         console from the test designer.
Step 4:  Find a message passed from the customer
         console object lifeline to the system lifeline.
Step 5:  If found, find just the next sibling message
         node of this message which is passed back to
         the customer console lifeline.
Step 6:  If found, display the value of the attributes
         "name" of the messages obtained in step 4 and
         5 as parameters and their values respectively.
Step 7:  End.
```

Fig. 8. Algorithm "CustomerConsole_LinkPV".

d) An Illustration of the Approach: Fig.9 shows the sequence diagram relating to logging into a Library system, searching for a specific book and getting it issued. In the diagram, the user object is external to the system providing inputs to the system and hencs acts as customer console. Fig.10(a) shows the suggested results obtained from applying our approach to this sequence diagram. Fig.10(b) shows the manually interpreted and verified CTDM parameters and values of Fig.10(a) on manual analysis of Fig.9. In Fig.10(b), parameter "Issue status" is shown to have a single value "Issued". Our observation is that, there may be other values like "Not issued" coming in through other sequence diagrams to complete the model. Similar is the case with "Logout Status" and other parameters.

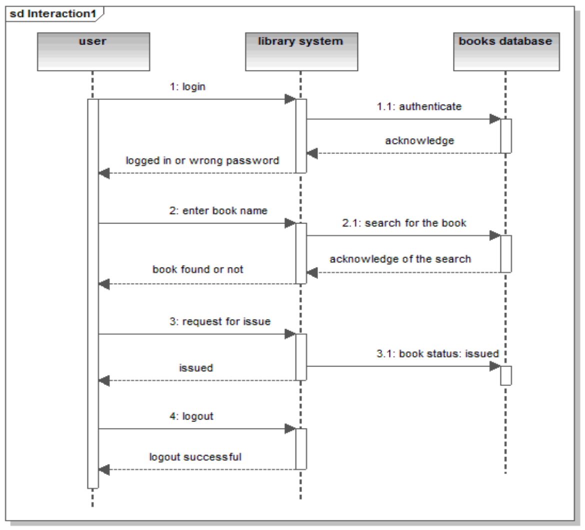


Fig. 9. Sequence diagram of Library system search transaction

Parameters	Login	Enter book name	Request for issue	Logout
Value 1	Logged in	Book found	Issued	success
Value 2	Wrong password	Bo not found		

(a)

Parameters	Login	Enter book name	Issue Status	Logout Status
Value 1	Correct password	Name of book found	Issued	success
Value 2	Wrong password	Name of book not found	Not Issued	fail

(b)

Fig. 10. (a) Suggested automated results and (b) Manually refined and verified CTDM elements of Fig.9.

3) Argument Passing: Our observation shows that, when synchronous messages are passed with arguments, it can be inferred that the arguments are the input parameters and the reply message forms the output parameter. In the example shown in Fig.11, quantity and rate are input parameters, reply message price forms the output parameter.

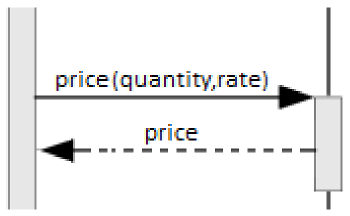


Fig. 11. Generic component of a sequence diagram with argument passing.

B. Sequence Diagrams with alt Combined Fragment

Alt combined fragment are used to designate a mutually exclusive option between two or more message sequences based on a guard condition [10]. Here we discuss two variations of guard conditions within alt, without relational operators and with relational operators.

1) Alt with Guard conditions without relational operators: In conditional fragments, the guards are used to denote the various conditions depending on which the execution flow alters. In this case, the guard conditions do not contain any relational operators as shown in Fig.12.

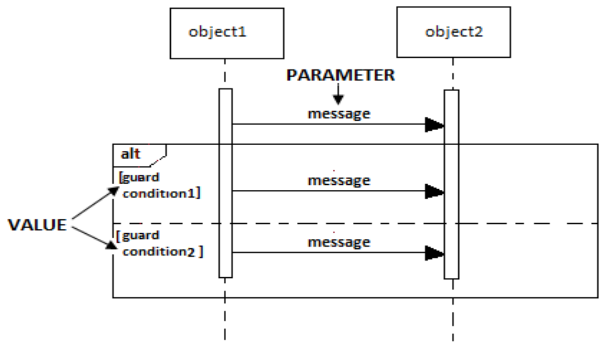


Fig. 12. Generic component of a sequence diagram showing alt combined fragment without relational operator.

a) Rule and Algorithm for parameter extraction:

Rule name: Alt_NoRlnOp_param.

Rule Narration: The message immediately preceding an alt combined fragment is extracted and the value of the attribute "name" of the message node is displayed as a parameter. The algorithm for Alt_NoRlnOp_param is shown in Fig.13.

```
Step 1: Begin
Step 2: Open the XMI file in read mode
Step 3: For a combined fragment node, the value of
        the attribute "name" of the message node,
        immediately preceding it is the parameter.
Step 4: End.
```

Fig. 13. Algorithm "*Alt_NoRlnOp_param*".

b) Rule and Algorithm for value extraction:

Rule name: Alt_NoRlnOp_value.

Rule Narration: The guard condition of the alt combined fragment is extracted and displayed as the value.The algorithm for Alt_NoRlnOp_value is shown in Fig.14.

```
Step 1: Begin
Step 2: Open the XMI file in read mode
Step 3: The guard condition is obtained by traversing
        down the combined fragment node to its last-
        level child node from the value of the "value"
        attribute.
Step 4: End.
```

Fig. 14. Algorithm "*Alt_NoRlnOp_value*".

c) Rule and Algorithm for linking the parameters and their values:

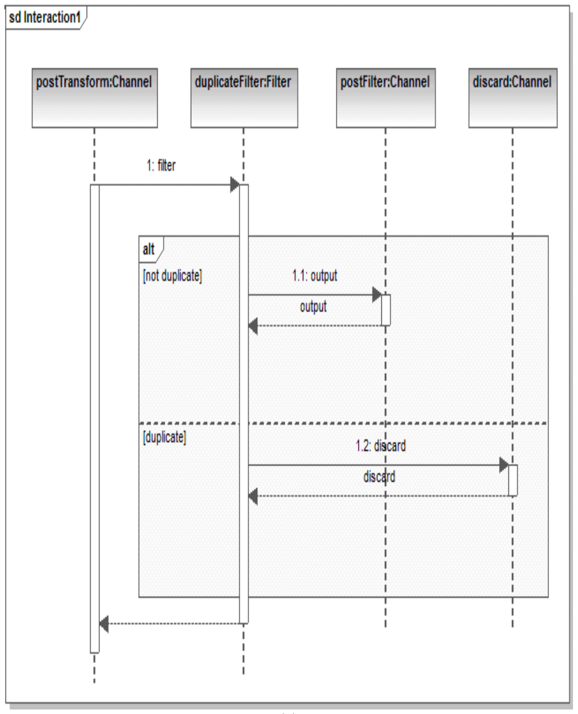
Rule name: Alt_NoRlnOp_LinkPV.

Rule Narration: The message immediately preceding an alt combined fragment is extracted and the value of the attribute "name" of the message node is displayed as a parameter. The guard conditions of this alt combined fragment is obtained and displayed as the values linked to the parameter.The algorithm for Alt_NoRlnOp_LinkPV is shown in Fig.15.

```
Step 1: Begin
Step 2: Open the XMI file in read mode
Step 3: For a combined fragment node, the "name"
        attribute's value of the message node
        immediately preceding it is the parameter.
Step 4: The guard condition is obtained from the
        "value" attribute of the last-level child node
        of the combined fragment node considered in
        step 3,which are the required values.
Step 5: End.
```

Fig. 15. Algorithm "*Alt_NoRlnOp_LinkPV*".

d) An Illustration of the Approach: Fig.16(a) shows a sequence diagram for a Duplicate Filter case study where the filter information received needs to be determined as "duplicate" or "not duplicate" before being handled by subsequent tasks. Fig.16(b) shows the suggested results obtained on application of alt rule as explained above. Fig.16(c) shows the suggested results obtained on application of synchronous message rule. The naming convention the sequence diagram designer has used with reference to message 1.1 and 1.2 in Fig.16(a) is output and discard respectively. On manual refinement and combining both the rules we arrive at Fig.16(d). The manual analysis shows that it is a single parameter outcome with two values output and discard.



(a)

Parameters	Filter
Value 1	Not Duplicate
Value 2	Duplicate

(b)

parameters	output	discard
Value1	output	discard

(c)

Parameters	Message Type	outcome
Value 1	Not Duplicate message	output
Value 2	Duplicate message	discard

(d)

Fig. 16. Duplicate case study (a) Sequence diagram (b) Suggested automated results on application of Alt rule (c) Suggested automated results on application of synchronous message rule (d) Manually refined and verified CTDM elements

2) Alt with Guard conditions with relational operators:

In conditional fragments, the guards are used to denote the various conditions depending on which the execution flow varies. In this case, the guard conditions have a relational operator as shown in Fig.17.

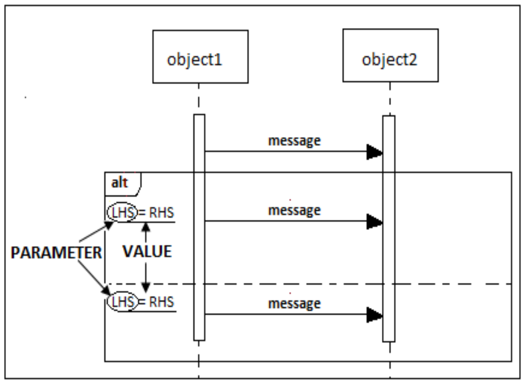


Fig. 17. Generic component of a sequence diagram showing alt combined fragment with relational operator.

a) Rule and Algorithm for parameter extraction:

Rule name: Alt_RlnOp_param.

Rule Narration: The first guard condition of the alt combined fragment is extracted. The LHS of the guard condition is displayed as the parameter.

The algorithm for Alt_RlnOp_param is shown in Fig.18.

```
Step 1: Begin
Step 2: Open the XMI file in read mode
Step 3: The guard conditions, obtained from the
            value of the "value" attribute of the combined
            fragment node's last-level child node, are
            obtained.
Step 4: For the combined fragment node's first child,
            display the L.H.S value of this
            attribute "value" until a relational condition is
            met, as the parameter.
Step 5 : End.
```

Fig. 18. Algorithm "*Alt_RlnOp_param*".

b) Rule and Algorithm for value extraction:

Rule name:*Alt*_RlnOp_*value*

Rule Narration: The guard condition of the alt combined fragment is extracted and displayed as the value.

Algorithm: The first 3 steps algorithm for value extraction is similar to the ALT_RlnOp_param algorithm. In essence we need to repeat the first three steps, and display the value of the attribute "value" found, as the value.

c) Rule and Algorithm for Linking the parameters and their values:

Rule name: ALT_RlnOp_LinkPV:

Rule Narration: The guard condition of the alt combined fragment is obtained. The L.H.S of the first extracted guard condition is displayed as the parameter while all the guard conditions are displayed as its linked values.

Algorithm: The first *4* steps algorithm for value extraction is similar to the RlnOp_param algorithm. In essence we need to repeat the first four steps, and display the value of the attribute "value" obtained for all the child nodes of the fragment node, as the value.

d) An Illustration of the Approach: Fig.20(a) shows an ATM withdrawal transaction sequence diagram resluting from the ATM requirements document [29] considered in our activity paper [26]. Fig.20(b) shows the suggested results of our approach which is inline with manually derived CTDM in [26].

C. Sequence Diagrams with opt,break and Loop Combined Fragment.

We classify the combined fragments opt, break and loop different from alt combined fragment because in these guard conditions may or may not exist.

1) Sequence Diagram with opt Combined Fragment: An opt is used to denote an "if-then" condition in a sequence diagram. It doesn't contain any else part and hence denotes an optional fragment. In essence, given a certain condition, the sequence occurs, else the sequence does not occur The flow to the optional fragment is denoted by a guard which is however not a required element [10].

If the guard condition is present in the frames content area as shown in Fig.19 ,then it is likely to contribute to the CTDM elements. The guard condition is interpreted in the same way as that of alt combined fragment algorithm for extracting the parameters and values.

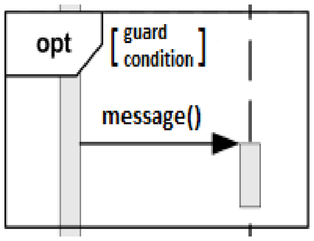
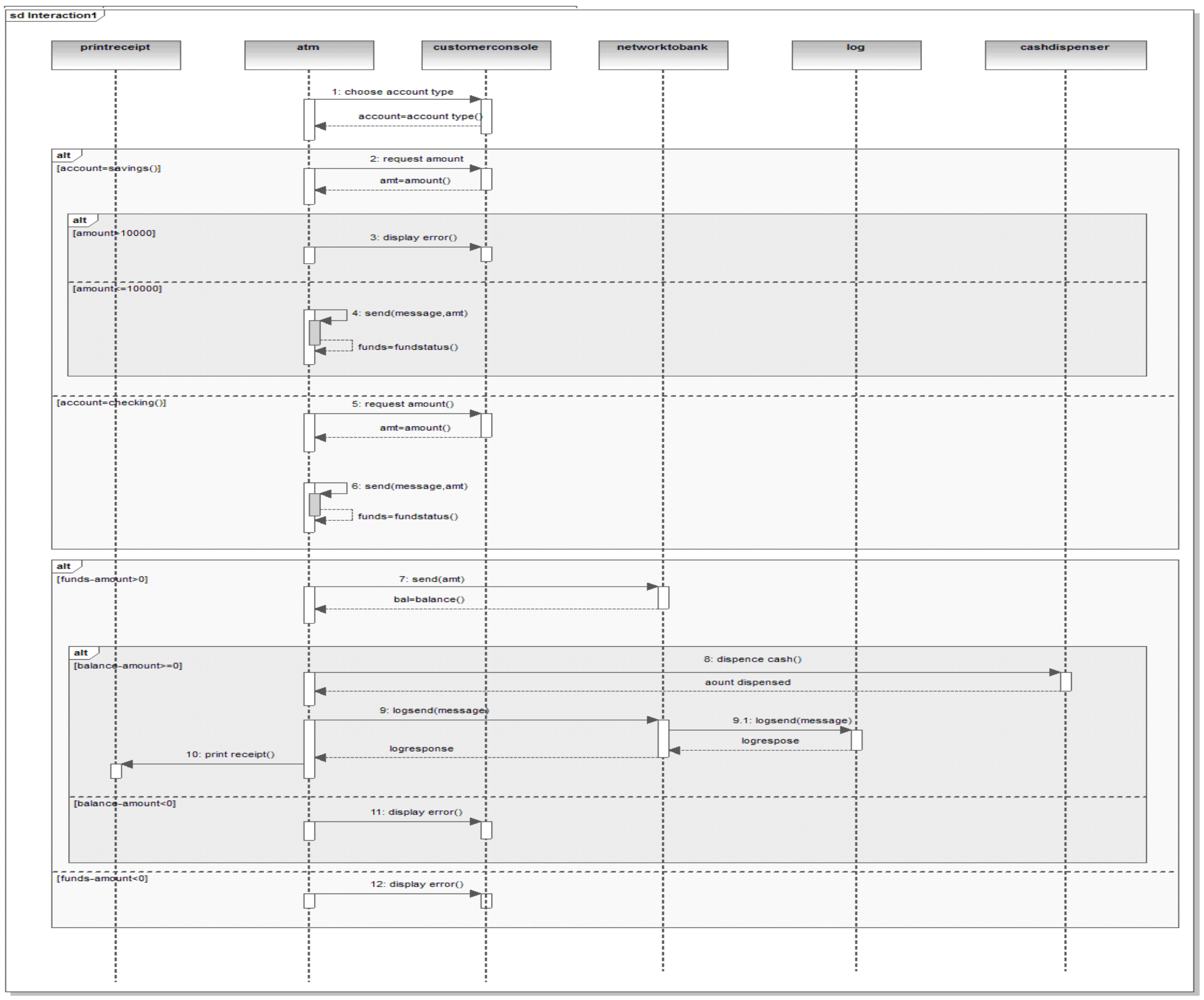


Fig. 19. Generic component of a sequence diagram showing opt combined fragment.



(a)

Parameters	Account	Amount	Balance-amount	Funds-amount
Value 1	Savings	>10000	Balance-amount>0	Funds-amount<=0
Value 2	Checking	<=10000	Balance-amount<=0	Funds-amount>0

(b)

Fig. 20. ATM withdraw transaction case study (a) Sequence diagram (b) Suggested automated results

2) Sequence Diagrams with Break combined fragment: Break combined fragment is similar to the break keyword used in the programming languages. If the guard condition within the break fragment evaluates to true, the remaining of the directly enclosing interaction fragment is overlooked. If the guard condition within the break fragment evaluates to false, the break operand is overlooked and the rest of the enclosing interaction fragment proceeds[10]. Again the guard condition within the break combined fragment as shown in the Fig.21 contributes to the CTDM elements parameter and value which can be determined in a similar fashion as that of alt combined fragment algorithm.

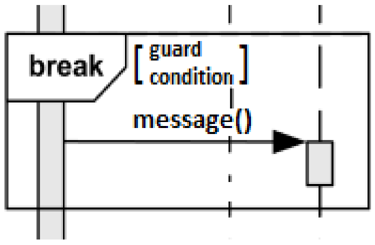


Fig. 21. Generic component of a sequence diagram showing break combined fragment.

3) Loop Combined Fragment: Loop fragment is an iteration operator and is used when there is a need to repeat a sequence. There are 3 ways to use the loop fragment as shown in Fig .22.

If guard condition becomes false, the loop terminates regardless of the minimum number of iterations specified. Our observation shows that the guard condition contributes to the identification of CTDM elements as in Fig.22(c).

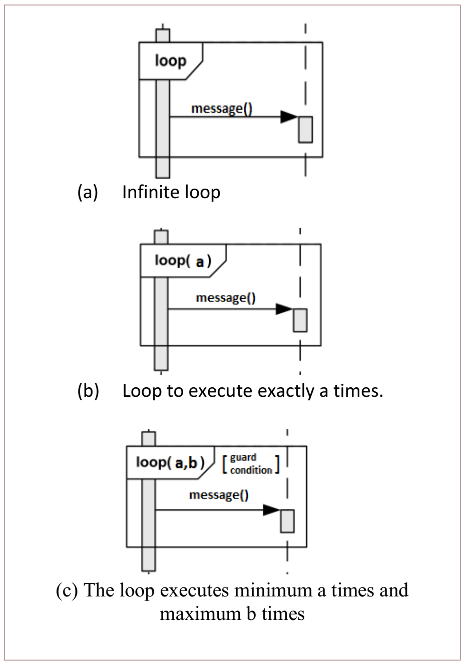

loop
message()
(a) Infinite loop
loop(a)
message()
(b) Loop to execute exactly a times.
loop(a,b)
guard condition
message()
(c) The loop executes minimum a times and maximum b times

Fig. 22. Generic component of a sequence diagram showing (a) Infinite loop (b) loop exact number of times (c) loop with guard condition.

V. Conclusion and Future Work

This paper presents a rule based approach to extract CTDM related information from UML 2.0 sequence diagrams. The UML 2.0, sequence diagram has been augmented with many new semantic constructs making it more expressive and powerful. They can be drawn in diverse ways, from an abstract level to most detailed level. In this paper, we have come up with a set of rules relating to specific constructs of the sequence diagram to derive information on parameters, values of the underlying CTDM. The guard conditions of various combined fragments and the messages in synchronous message calls are found to hold the test parameters and values. The formulated rules are applied independently on individual sequence diagrams and the results output can be of assistance to the test designer in building CTDM. Our vision is to develop an integrated tool that automates the process of CTDM derivation by analyzing various sources of information like requirements document, activity diagrams and sequence diagrams. Derivation of CTDM from activity diagrams is reported in our earlier work [26]. Currently we have implemented all the rules discussed in this paper and as an immediate future work, we would like to design and implement appropriate rules for synchronous message calls with arguments, constructs opt, break and loop without guard as discussed in section IV. In our initial study, we did not find constraints showing up directly in the sequence diagrams. However we may have to investigate this further and add additional rules as required.

Acknowledgment

We would like to sincerely thank Dr. Ramesh Babu D R our head of the department, for his continuous support and motivation throughout the pursuit of this research. We would also like to express our heartfelt gratitude to our students Ankita, Prateek and Abhinandan for helping us in implementing our prototype.

References

[1] D. M. Cohen, S. R. Dalal, J. Parelius and G. C. Patton, “The combinatorial design approach to automatic test generation,” IEEE software, vol. 13, No 5, pp. 83–89, September 1996.

[2] Ashalata Nayak and DebashishSamanta, “Automatic Test Data Synthesis using UML Sequence Diagrams,” Journal of Object Technology, Vol. 09, No. 2, pp.75-104, March-April 2010.

[3] Itai Segall, R. Tzoref-Brill, and A. Zlotnick, “Simplified modeling of combinatorial test spaces,” IEEE 5th International Conference on Software Testing,Verification and Validation (ICTS) , 2012, pp. 573-579.

[4] Sergiy A. Vilkomir, Khalid A. Abdelfattah and Sudha Gummadi, “MIST: Modeling input space for testing tool,” Proceddings of the 13th IASTED International Conference Software Engineering and Applications (SEA 2009), Cambridge, MA, USA, 2009, pp. 210–217.

[5] Sergiy A. Vilkomir, W. Thomas Swain and Jesse H. Poore, “Software input space modeling with constraints among parameters,” 33rd Annual IEEE International Computer Software and Applications Conference, Seattle, Washington, USA, 2009, Vol. 1, pp. 136–141.

[6] R. Krishnan, S. M. Krishna and P. S. Nandhan, “Combinatorial testing: Learnings from our experience,” ACM SIGSOFT Software Engineering Notes, 2007, vol. 3, pp. 1–8.

[7] P.Samuel and R.Mall, “A novel test case design technique using dynamic slicing of UML sequence diagrams,” in e-Informatica, 2008,2(1) , pp. 71-92.

[8] M.Grochtmann and K.Grimm., “Classification Trees for Partition Testing,” Journal of Software Testing, Verification, and Reliability, 3(2):63–82, 1993.

[9] Mario Brcic and Damir Kalpic, “Combinatorial testing in software projects,” MIPRO, 2012 Proceedings of the 35th International Convention, 2012, pp. 1508–1513.

[10] “UML basics: The sequence diagram.” [online]. Available: http://www.ibm.com/developerworks/rational/library/3101.html

[11] Itai Segall, R. Tzoref-Brill, and A. Zlotnick, “Commom patterns in combinatorial models,” IEEE 5th International Conference on Software Testing,Verification and Validation (ICTS), 2012, pp. 624-629.

[12] Grady Booch, James Rumbaugh and Ivar Jacobson, “The unified modeling language ,” Reference Manual, Addison Wesley, 2001.

[13] Robert V. Binder, “Testing object-oriented systems: models, patterns and tools ,” Addison Wesley, 2000.

[14] UML modeling tool [online]. Available: http://www.altova.com/

[15] Emanuela G. Cartaxo, Francisco G. O. Neto and Patr´ıcia D. L. Machado, “Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems”. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, SMC 2007, pp. 1292–1297. IEEE (2007)

[16] Changhai Nie and Hareton Leung, “A survey of combinatorial testing,” ACM Computing Survey, 2011, vol. 43, No.2.

[17] Mehra N. Borazjany, Linbin Yu, Yu Lie, Raghu Kacker, and Rick Kuhn, “Combinatorial testing of ACTS: A case study,” IEEE 5th International Conference on Software Testing, Verification and Validation (ICTS), 2012, pp. 591–600.

[18] D. Richard Kuhn, Raghu N. Kacker, and Yu Lie.(2010,October) “Practical combinatorial testing” [Online]. Available:

http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf

[19] XML Metadata Interchange (XMI). [online]. Available: http://www.omg.org

[20] Mats Grindal and Jeff Offutt, “Input parameter modeling for combination strategies,” In Proceedings of the 25th Conference on IASTED International Multi-Conference, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[21] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, “The AETG system: An approach to testing based on combinatorial design,” IEEE Transactions On Software Engineering, vol. 23, No 7, pp. 437–444, July 1997.

[22] ”UML basics tutorial”. [online]. Available: http://www.uml-diagrams.org/

[23] “The python tutorial”. [online].

Available: http://docs.python.org/2/tutorial

[24] “Stock Payment example”. [online]. Available: http://www.codeproject.com/Articles/28445/UML-Interview-Questions-Part-1

[25] “Duplicate Filter Example.” [online]. Available: https://www.ibm.com/developerworks/community/blogs/workflowusingsi/?lang=en

[26] Satish P.,Sheeba K. and Rangarajan. K.,"Deriving Combinatorial Test Design Model from UML Activity Diagram",IEEE 6th International Conference on Software Testing, Verification and Validation (ICTS),18-22 March 2013, pp. 331 - 337.

[27] Zoltan Micskei and Helene Waeselynck.,"The many meanings of UML 2 Sequence Diagrams: a survey," Software and Systems Modeling,2011, 10(4), pp. 489–514.

[28] T. J. Ostrand and M. J. Balcer.,"The category-partition method for specifying and generating functional tests,"Communications of the ACM,June 1988, 31(6), pp. 676–686.

[29] “ATM simulation example”. [online]. Available: http://wwwagse.informatik.unikl.de/teaching/gse/ws2012/example/ESExample_Document.pdf.

[30] D. Richard Kuhn, Raghu N. Kacker, Yu Lei," Introduction to Combinatorial Testing; CRC Press", 2013, ISBN 978-1-4665-5229-6.

[31] “OMG Formal Specifications”. [online]. Available: http://www.omg.org/.

[32] “UML Sequence Diagrams: Guidelines”. [online]. Available: http://msdn.microsoft.com/en-us/library/dd409389.aspx#Simple