Data Flow Analysis of UML Models by ALF

S. Obaid, S. Asghar, and M. Naeem

Abstract—**Data flow analysis (DFA) technique is used to analyze program into data variables and identify data flow operation on these variables. This basic information can be used to identify data dependencies, test data, program execution paths and hence it is helpful in the testing process to verify expected behavior of the system. In terms of executable UML, the models are based on action languages which internally consist of variables and specify data flow operation on these variables. Therefore, DFA of action languages is essential to analyze UML models in term of data flow and use the information to verify formal correctness (expected behavior) of the system. In our proposed approach, we are using action language for fUML (ALF) to design and analyze executable UML model. Analytical results show that DFA of executable models provides the precise execution flow of variables which are used to identify data dependencies and verification of system expected behavior from its abstract model.**

Index Terms—**DFA, UML, testing, executable UML, ALF.**

I. Introduction

The model can formally describe a particular aspect of the system by following system's specification. It can present an abstract view of the system by graphical notations and complex operations by using natural or action language specification [1]. In this way model can help to represent and analyze that particular aspect of system to study feasibility of requirements, ambiguities in specifications and to gain significant knowledge about the system to be developed. An additional benefit of the model includes light weight or simple representation of a system which enables a modeler to capture knowledge about a system without indulging into code complexity.

Executable modeling is ability of model to be executed on the bases of execution semantics. UML superstructure [2] provides the precise action semantics and there are many action languages that are based on these action semantics such as ALF [1], java like action language (JAL) [3], Kennedy-Carter's action specific language (ASL) [4] etc. The use of these action languages can make an existing UML model to act as an executable entity and can be used as a prototype for the system to be developed. Among existing action languages, ALF is object management group (OMG) specified language for modeling executable systems. It consists of C or JAVA like syntax and provides the same way to define classes, object and method call. In the domain of executable UML models, actions can be specified in UML state machine [5], [6] and activity diagram. In the context of a state machine model, actions are written in the form of Entry, Do and Exit by using concrete action language. The state machine can be considered as flattened and actions are specified within states of state-machine. On the other hand, UML activity diagram can provide detailed descriptions of particular state's action or entire execution flow of the system. In any case, an action itself consists of variables and their values. Change in variable's value or attribute might affect the execution flow of the system and hence it can trigger the particular behavior of the system. In order to ensure accurate behavior of model, it is required that actions should be considered as data operation and they must be analyzed term of data flow [6].

Data flow analysis (DFA) technique is used to discover useful properties such as variable's value, point of initialization, data dependency and reachable definition etc. of variables within the program being analyzed. Data flow analysis techniques are initially found useful for compiler optimization [7] but it is also found to have effective uses in software testing such as finding test paths, test data selection [8] and anomalies detection e.g. Def-def, undefined-used within the program. In discipline of software testing and verification process, it is used to ensure soundness of program by identifying flow paths, variables anomalies which can be used to observe the program behavior when it executes through particular path. Rapps [8] have made a first attempt at bringing DFA technique for testing premises but code based DFA technique cannot be applied to action languages due to the abstract nature of actions. DFA of action semantics [6] is found helpful in determining flow information about action languages within the model. A variable's value can influence particular execution flow within the program which consequently executes particular behavior of the system. Thus DFA can play an import role to ensure accurate behavior of the system during testing.

This paper addresses the issue of DFA of UML models with the help of ALF. Studying existing DFA technique on UML models, we observed that the abstract nature of UML models hinders the ability of DFA technique in finding out detailed data flow information from the model. The use of action language to UML model can lower down the issue of abstraction and in this way the resulting DFA of the model can find out precise data flow information of the system. The rest of this paper is organized as follows: Section II discusses the existing literature on DFA of UML models: Section III describes proposed approach: results and discussions are mentioned in Section IV: Section V summarizes the work done and future direction.

The manuscript received June 15, 2013; revised August 17, 2013. This work is supported by University Institute of Information Technology, PMAS Arid Agriculture University Rawalpindi.

S. Obaid is with ARID Agriculture University, Rawalpindi (e-mail: samir_obaid@yahoo.com).

S. Asghar is with University Institute of Information Technology, PMAS-Arid Agriculture University, Rawalpindi (e-mail: sohail.asghar@uaar.edu.pk).

M. Naeem is with Muhammad Ali Jinnah University, Islamabad (e-mail: naeems.naeem@gmail.com).

II. Literature Review

Data flow analysis (DFA) is used to utilize definition and use of variables in program and to compute associations and relations such as dependencies among data objects [9]. The value of variable is responsible for program control flow and hence it represents particular behavior of the system [6]. In this way, system particular functionality can be depicted by identifying data occupied by variable. Therefore, data flow information must be considered when system is intended to be tested or analyzed. In this section, we describe existing approaches where DFA is used for program analysis and different testing purposes.

Kim [10] presents test cases generation from UML state diagram. The concurrent and hierarchical structure of model is flattened and state machine is transformed into intermediate models called extended finite state machine (EFSM). EFSM is then used to generate flow graph and test cases are generated on bases of data flow information of variables. In their technique, they describe how conventional DFA techniques can be used to generate test cases from UML state-machine diagram. Furthermore, they also describe definition-use associations of variable that either occurs due to hierarchical or concurrent structure of state machine model. Although they use to identify definition-use operation and definition-use (DU) pairs of variables by applying DFA on flattened state machine model, but the use of un-interpreted/natural language expressions to describe behavior of system can hinder the ability of DFA in finding detailed data flow information from model. Since natural language expressions are not based on formal semantics or grammar rules and therefore it is not possible to parse and tokenize these expressions into atomic expressions or variables.

Liuying and Zhichang [11] propose DFA of UML state-chart diagram. Like Kim [10] they use to flatten the hierarchical and concurrent structure of state machine to simplify the model. The sub states in existing state-chart model are specified as atomic states and methods are defined to select test paths from whole state machine including hierarchical execution region. The proposed approach is beneficial in generating reduced test suites from UML state machine model and they meet the objective by eliminating test paths if they are found as prefix of some other test paths.

Hong *et al*. [12] described test sequence selection method by applying DFA technique on state-chart. Like DFA on UML state machine [10], they transform state-chart to EFSM model which contains events, guards, actions and all possible runs (test paths) of state-chart. Actions are described at nodes and transition edges. Conventional DFA technique is applied on state machine by identifying predicate use (p-use), computational use (c-use) and definition points of variables in EFSM. Concurrent and hierarchical structure of state machine is flattened and test paths are generated from resulting flow graph by EFSM. Hong *et al*. [12] transform state machine model into flow graph and applies DFA on model. The benefit of flow graph is its ability to occupy variable and data information whether they exist on nodes or transition edges. Although the flow graph is used to occupy expressions that appears on transition edges and nodes but author does not describe parsing of these expressions into data variables in order to identify definition or use operation on these variable. UML consist of set of events and actions such as *callEvent*, *createObjectAction*, *callBehaviorAction* etc. [2] which need to be identified and associated with data variables, but the use of un-interpreted or natural language expressions can become hurdle to identify this information from model. Therefore, a formal semantic or grammar based language is required [13] which can precisely describe actions execution within model and it can also be parsed into variables with help of language parser.

Lnous and Honiden [14] propose components based DFA by using OCL expressions. The objective is to extract data flow information among different classes and to flow dependencies that occurs due to procedural calls. In order to meet the objective, they use OCL expressions to describe variables, constraints (in form of pre-conditions) and operation execution. They translate OCL post condition into set of operations in order to describe data variables and their associated actions.

Cavarra [15] proposes DFA of abstract state machine model. In their approach, they keep in consideration the effect of parallel execution procedures on global variables. Cavarra [15] describes the fact that program control flow path is not sufficient if there are parallel paths and each of them can modify the variable value. Cavarra [15] introduces the concept of multi-agent (independent execution regions) and elaborate data flow information from multi-agent abstract state machine model but does not specify specific language such as OCL or action languages to support their methodology by automated tool.

Briand [5] performs DFA on UML state chart and describes the effectiveness of DFA technique in MBT. In their approach, they transform state machine to event action flow graph (EAFG) which is a directed graph where nodes represent post condition of actions and edges represent guards and precondition for successor nodes. EAFG is augmented with object constraint language (OCL) expressions. DFA is performed on data variables that appear in OCL expressions. Definition clear path and definition use path are identified on bases of definition use operation on variables and builds transition tree to specify execution of operations. In their proposed approach, they make use of DFA information to identify flow paths from state machine model through EAFG. The flow paths which consist of large number of definition use pair of variable is considered more effective in fault detection because such paths have potential to accommodate large number of anomalies. Briand [5] also defines set of rules which are used to identify definition-use operations on variables and collection operations. The collection operations are commonly used by OCL and UML based action languages and therefore, the defined rules are equally applicable on both of them. Since OCL itself is a declarative language and it can't be used to define computation logic and algorithmic details in model [13].

Waheed [6] uses action specific language (ALS) and analyze data flow information from UML state machine model. ASL is based on UML action semantics and its augmentation with UML model can become a better technique to address following important issues of existing approaches.

A. Use of Informal/un-Interpreted Language Expressions

Waheed [6] describes internal details of state machine with ASL. Unlike informal language expressions, ASL is based on formal semantics and grammar rules and therefore its expressions can be easily parsed into atomic expressions or data variables.

B. Precise Action Specification

Action language such as ASL, ALF etc is based on UML action semantics and they can precisely describe executable behavior of model. These action languages can also be used to specify algorithms and add computational details which are required for execution.

Like existing approaches [6], [7], [9], Waheed [6] also flattens the hierarchical and concurrent structure of UML state machine. Fig. 6 refers case study from Waheed [6] approach where states represents actions execution and transition edges represent communication among states. From Fig. 6, we can observe that ASL expressions are used to describe action/operation within state such as **"status=idel"** in state **1.1**. Similarly OCL expressions are used to describe expressions on transition edges such as **"call(cf) (cf<ef)"** on transition edge 1-6". ASL parser and mapping rules are used in order to tokenize expressions into variables and identify definition-use operations on these variables. But the use of parser and mapping rules can only identify definition-use operation on those variables that exist inside a state(s) whereas it completely neglect definition-use operation on variables that exist on transition edges. Consequently, the resulting DFA of model will not have data flow information for all those variables that are defined within state but used on transition edges e.g. **cf** and **ef** are defined in state 1 and used on transition edge 1-9 but this information is not acquired by Waheed [6] approach. The situation become more critical if *use* of variable occurs only on transition edge such as "**direction**" defined in state 6.4, 9.4 and used on edge 7-13, 10-13. Hence variable "**direction**" will always be considered as unused although it is used on two transition edges (7-13, 10-13).

Above discussion lead us to the conclusion that use of formal languages such as OCL or action languages can become better approach of DFA of UML models. Since OCL is descriptive language and lack execution ability therefore action languages such as ASL, ALF, and SMALL etc can become better choice for modeling and analysis. Existing approaches [6] uses UML state machine to identify variables' flow information but neglect identification of those variables that appears on transition edges or predicates (p-use). Therefore, it is necessary to propose a technique which could acquire complete coverage of code and data flow operations on variables in order to ensure the correctness of system.

III. Proposed Approach

In this section we describe our approach towards DFA of UML models by using semantic based action language ALF. We have developed state-based ALF model analysis tool (SAMAT) for analysis of (ALF based) executable UML models. Fig. 1 describes the architecture of SAMAT where ellipses represent activities and boxes shows input to or output from activities. Sequence of steps and process flow of SAMAT is described in following subsection.

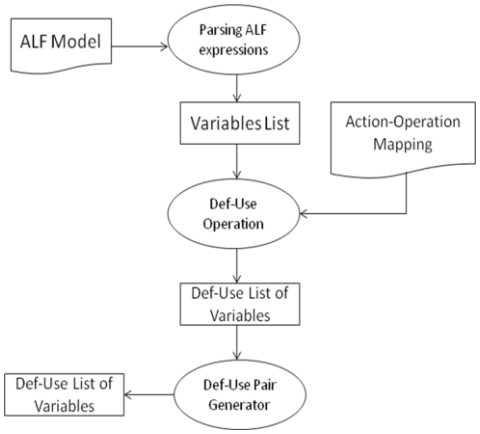

Fig. 1. Process flow of SAMAT.

A. ALF Parser

SAMAT takes ALF model as an input and tokenize expressions by ALF Parser. The parser identifies atomic expressions from statements and specifies UML action associated with these expressions.

ALF itself does not contain concept like atomic expression. ALF set of expressions includes primary, increment and decrement, unary, binary, conditional assignment expression. In our case we use the term atomic expression to classify all those expressions that represent individual elements such as variables, class object, tuples and that can be directly mapped to UML actions.

B. Def-Use Operations

ALF parser provides atomic expressions/variables along with action associated with these expressions. Waheed [6] categorizes UML actions into define and use classes and we are also using this classification in form of action-operation mapping in SAMAT to identify data flow operation on variables. Table I describes some UML action along with data flow operations. Classification of actions into definition-use can consequently determine DU operations on associated data variables.

TABLE I: Action-Operation Mapping

Action	Operation
Add structural feature value action	Def
Create object action	Def
Read structural feature action	Def

C. Definition-Use Pairs

DU pairs of variable can be identified by finding use and then corresponding definition(s) of used variable. Fig. 2 describes an algorithm to find out DU pairs of variables from a program being analyzed. After identifying data flow operation on variables, the resulting model appears in form of acyclic flow graph (AFG) which is taken as an input and for each used variable the graph is traversed to find out definition in all possible parents nodes of variable. There can be a single event or multiple non-deterministic (external/internal) events

that can cause definition-use operation on variables. In our proposed approach, we use to handle these cases by graph traversing algorithm and feasible path matrix (FPM).

```
function find DU Pairs(list)
{
for (i=0 to list. size ())
{
used Variables=list. Get Used Variables ()
for Each(use Var in used Variables)
{
find Definition Point (use Var)
}
If (list. subList!=null)
{
find DU Pairs (list. subList)
}
}
}
```

Fig. 2. Algorithm to identify DU Pairs.

IV. Case Study and Results

We have discussed in Section I that ALF concrete syntax resembles with C of Java and provides same way to define classes, object and method calls. In this way ALF act as a bridge between abstract UML models and complex programming languages and hence its DFA can provide necessary data flow information which is abstract but precise enough to verify correctness of the system. Following section refers elevator control system (ECS) from Waheed [6]. Fig. 6 depicts the state machine model of ECS where we apply DFA by augmenting state machine with ALF and compare results on the bases of DFA information.

In this case study our objective is to describe the advantages of DFA in finding out the formal correctness (expected behavior) of a system from its model. We are mapping flattened state machine of ECS to ALF and resulting ALF model is parsed by the ALF parser to find out data variables and DU pairs. As mentioned in previous sections that feasible path matrix can be used to find out data dependencies if state consists of un-deterministic events and each of them can trigger the variable's value. Keeping in view the fact, we are also using feasible path matrix [6] to find out data dependencies and DU pairs of variables.

A. Def-Use Operations on Variables

```
Active Class ECS
{
 Public Floor cf;
 Public Floor ef;
 Public Floor targetFloor;
 Public String status, direction;
}
do
{
accept( callFloor:Floor)
cf=callFloor;
if(cf<ef)
        {
        startMovingDown();

}
```

```
Activity startMovingDown()
{
  1. Status="moving"
  2. direction="down"
  3. p.enqueue(cf)
  4. targetFloor=p.front()
}
```

Fig. 3. ALF active class for start moving down state.

We are taking flattened state machine model of ECS and transform it into ALF by specifying mapping rules. We are performing this transformation by mapping state into ALF activity class and incoming transition into corresponding event and predicate in ALF active class. Fig. 3 depicts an example of mapping "StartMovingDown" state of ECS to ALF activity class. From Fig. 3 it can be observed that by mapping "StartMovingDown" state to ALF we have also acquired variables that exist on the incoming transition edge of the state. Now the resulting ALF model can be parsed by the ALF parser to find out variables and identify DU operations on these variables.

The p-use in Table II describes ***use*** operation on variable that appear on state transition edge or active class predicate. In other words, it shows that *use* operation is performed on "**ef**" at incoming transition edge of *startMovingDown* state. By using above mentioned approach, we have found existing (transition and state) variables from every state of ECS and identified data flow (definition, p-use, c-use) operations on them.

TABLE II: Def-Use Operation on Variables

Variable	State	Operation
cf	---	def
ef	---	p-use
ef	---	p-use
status	startMovingDown	def
direction	startMovingDown	def
cf	startMovingDown	c-use
targetFloor	startMovingDown	def

B. DU Pairs

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	1	1	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	0	1	1	0	1	1	1	1
3	0	1	1	1	1	0	1	1	0	1	1	1	1
4	0	1	1	1	1	0	1	1	0	1	1	1	1
5	0	1	1	1	1	0	1	1	0	1	1	1	1
6	0	1	1	1	1	0	1	1	0	1	1	1	1
7	0	1	1	1	1	0	1	1	0	1	1	1	1
8	0	1	1	1	1	0	1	1	0	1	1	1	1
9	0	1	1	1	1	0	1	1	0	1	1	1	1
10	0	1	1	1	1	0	1	1	0	1	1	1	1
11	0	1	1	1	1	0	1	1	0	1	1	1	1
12	0	1	1	1	1	0	1	1	0	1	1	1	1
13	0	1	1	1	1	0	1	1	0	1	1	1	1

Fig. 4. Feasible path matrix of ECS from waheed [6].

We use to find DU pairs of each variable within state by finding variable's use point and then corresponding definition point of that variable. In order to find intrastate DU pairs, we are consulting feasible path matrix (FPM) of giving state machine model. The FPM is described in Fig. 4 where it marks the entry between two states as 1 if there is a direct or indirect connection between two states. From each state, we acquire defined variables and then look for their use points in states that are marked feasible by FPM. The process is continued until we acquire DU pairs of all defined variables within the model. Table III presents the results of interstate

and intrastate DFA of ECS. The distinction of our results is the identification of *definition* and *p-use* of variables such as "direction" and "targetFloor" etc. and these distinct variable pairs are availed by identifying variables on transition edges of a state machine model.

Table III describes data flow information from ECS that exists in states and state transitions. Precise DFA information also includes those variables that exist on states and state transition such as "direction" and "target floor" etc. On the other hand, the resulting variables also include invalid DU pairs such as "targetfloor" defined in state 4 and 5 and used on transition edge 3-1, 7-13. These DU pairs can never be availed by test paths because their definition is killed by state 3. The use of adjacency matrix can avoid identification of such DU pairs. Another kind of invalid DU pairs includes a definition of "targetFloor" at state 2 and 6 and their use of edge 12-11 and 12-8. Such invalid DU pairs can appear even by using adjacency matrix and there can be (invalid) test path that avail these variables.

TABLE III: DU PAIRS OF VARIABLES

Variable	State	Line	Use State	Line
cf	1	3	2	2
cf	1	3	5	2
cf	1	3	6	3
direction	6	4	Edge 7-13	
direction	9	4	Edge 10-13	
floorNo	1	3	7	1
floorNo	7	1	10	1
targetFloor	6	5	Edge 7-13	
targetFloor	6	5	Edge 2-11	

In subsequent portion of this chapter, we further extends experimental results of ECS to identify data flow information and their benefit in formal accuracy of system through testing.

C. Adding Exceptional State to ECS

TABLE IV: MEASURING ACCURACY BY INITIAL HYPOTHESIS

Variables	Identified variables by Waheed [6]	Identified variables by DFA of ALF Model	Adding MailFunctioned State	Adding Emergency DoorOpen state
Declared variables	8	8	8	9
Definition operation	22	22	23	25
Def: State Use: State	17	17	17	18
Def: State Use: edge	N/A	37	51	55
	0	16	20	25

We have added some exceptional states in ECS which includes "Malfunctioned" and "Emergency Door Open". The purpose of these states is to describe the behavior of the system against the unacceptable system event. Table IV describes the resulting data flow information after adding exceptional states to models.

We have compared our result with Waheed [6] on the bases of DFA results of Table IV. From comparison graph we can note that both approaches cover the same number of definitions and def: State & Use: State operation. Fig. 5 also shows that Waheed [6] neglect variables information that appears on state transition edges but on the same hand they do not have invalid DU pairs in their analysis model.

The latter two bars in graph describe analysis results by adding new states to the system. We use the compute the results of additional states to find out impact of each state in improving accuracy and overall cost (unreachable DU pairs) by analysis of model.

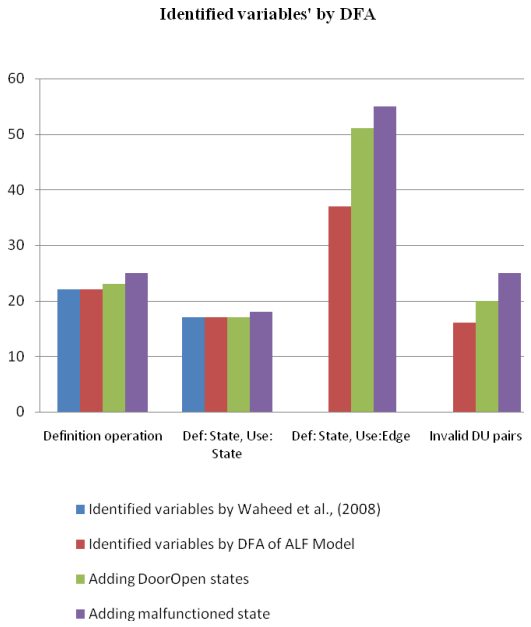

Fig. 5. Comparison graph by DFA of ALF.

D. Measuring Cost and Accuracy

By adding new states to ECS, we acquired additional data flow operations from both computer and predicate part of the scheme. Furthermore, the process also identified untraceable DU pairs from ECS which are called invalid DU pairs and describes as cost in Table V.

TABLE V: AVERAGE INCREASE IN COST AND DU PAIRS

variables	DU operations on state machine	Adding MailFunctioned state	Adding Emergency DoorOpen state	Average
c-use	17	0.00	5.88	2.94
p-use	37	37.84	48.65	43.24
cost	16	25	56.25	40.63

E. Average Improvement on Bases of Hypothesis Value

DFA is helpful in verifying accuracy [10] through the selection of improved test paths from the model. By testing system on the basis of initial data flow information, we assumed that there is a 40 % probability of system's accurate functionality whereas 60 % chances exist for system failure. By adding exceptional states to the system the probability of

successful execution increased by (55.14 - 40) 15.14% and then 30.07% which adds to average improvement of 22.60 %.

Table VI describes a gradual and average improvement after initial assumption value based on hypothesis.

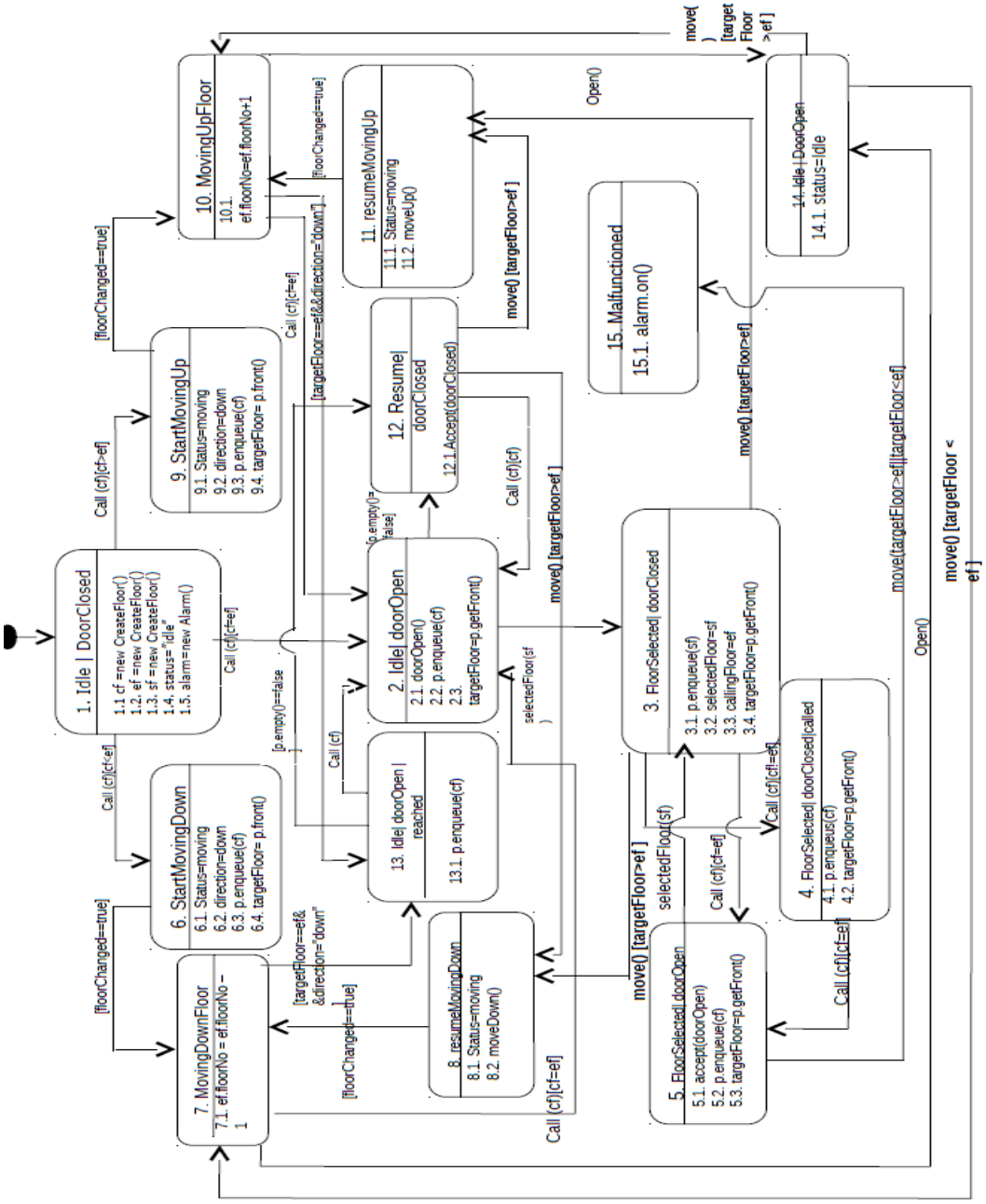

Fig. 6. Impact on accuracy by adding new states.

TABLE VI: MEASURING ACCURACY BY INITIAL HYPOTHESIS

Variables	assumption value based on hypothesis	Adding Mail Functioned State	Adding Emergency Door Open state
Probability	40	55.14	70.07
Difference		15.14	30.07
Average improvement		22.60	

We can plot the results of table VI with the help of the graph. From Fig. 7, it can be observed that by taking initial hypothesis of 40% (number 40 in a vertical column) at initial state the accuracy increased to 55 and then 70 percent by the addition of two new states to the system. It can be observed that the accuracy line deviates toward the y-axis after adding second state which shows a higher contribution of 2nd state in an average improvement of system behavior.

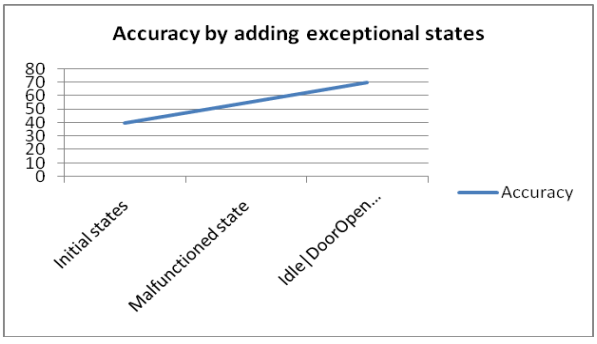

Fig. 7. Impact on accuracy by adding new states

F. Observation

In the above case study we have used DFA information to

determine the impact of each exceptional state in improving accuracy or protecting system against failure. We also observed some additional cost of DFA that exist in form of invalid or unreachable DU pairs of variables and there also exist invalid test paths that occupy these DU pairs. Yet the user input data is the only way to identify these invalid test paths and eliminate them from test suite. By applying DFA on different case studies we observed a weak relation among the cases which means that cost and accuracy varies from one particular situation to another which depends upon the behavior of state, state execution and evaluation of variables within model being analyzed.

V. Conclusion and Future Work

In this research we have used executable language ALF for DFA of UML models. We also describe the complete DFA process by SAMAT and identification of variable flow information from both hierarchical and flattened UML models. From our experimental work, we also described how DFA technique is helpful in identifying impacts of system states in improving the accuracy of the system. We have found an ALF suitable language for modeling and analysis because its concrete syntax can be mapped to UML modeling notations and programming language constructs such as class, object and events etc. From our experimental work, we have observed that mapping UML state machine to ALF has enabled us to find precise data flow information from the model. Since DFA has numerous application and we have just used it to find a DU pair of variables. We have aimed some feasible future work in domain of executable models. Our instant future task can be identified of refined test cases by DFA information. Similarly DFA information can be used in fault seeding and thus it can be useful for system verification by mutation operation.

References

[1] Action Language for Foundational UML (ALF): Beta 1. [Online]. Available: http://www.omg.org

[2] Unified Modeling Language: Superstructure. Version 2.3. (2007). Object Management Group (OMG) document ptc/06-04-02. [Online]. Available: http://www.omg.org

[3] T. D. Trong, S. Ghosh, and D. France "Java like action language (JAL)," Specification 1.1-Beta version. CS Technical Report 06-102, Department of Computer Science, Colorado State University, Fort Collins, USA.

[4] K. Carter. Abstract Solutions. [Online]. Available: http://www.kc.com

[5] L. C. Briand, Y. Labiche, and Q. Lin, "Improving state chart testing criteria using data flow information," in *Proc. 16th IEEE International Symposium on Software Reliability Engineering*, 2005, pp. 95-104.

[6] T. Waheed, Z. Z. Iqbal, and Z. I. Malik, "Data flow analysis of UML action semantics for executable models," in *Model Driven Architecture - Foundations and Applications, Volume 5095 of Lecture Notes in Computer Science*, I. Schieferdecker and A. Hartman, Eds. Springer Berlin / Heidelberg, 2008, pp. 79–93..

[7] Y. Bertot, B. Gregoire, and X. Leroy, "A structured approach to proving compiler optimizations based on dataflow analysis," in *Proc. of TYPES'04*, Springer LNCS, 2006, vol. 3839, pp. 66-81

[8] S. Rapps and E. J. Weyuker, "Data flow analysis techniques for test data selection," in *Proc. 6th International Conf. on Software Engineering*, Tokyo, Japan, 1982, pp. 367-374.

[9] L. Moonen, "A generic architecture for data flow analysis to support reverse engineering," in *Proc. Second International Workshop on the Theory and Practice of Algebraic Pecifications, Electronic Workshops in Computing, Amsterdam*, Springer, Heidelberg, 1997.

[10] Y. G. Kim, H. S. Hong, S. M. Cho, D. H. Bae, and S. D. Cha, "Test case generation from UML state diagrams," in *IEEE Proc.-Software*, vol. 146, 1999, pp. 187-192.

[11] L. Liuying and Q. ZhiChang, "Test selection from UML statecharts," *Technology of Object-Oriented Languages and Systems*, 1999, pp. 273-279.

[12] S. H. Hong, Y. G. Kim, S. D. Cha, D. H. Bae, and H. Ural, "A test sequence selection method for state charts," *Journal of Software Testing, Verification and Reliabilit*, vol. 10, pp. 203–227, 2000.

[13] S. J. Mellor, S. Tockey, R. Arthaud, and P. LeBlanc, "Software-platform-independent precise action specifications for UML," in *Proc. Unified Modeling Language, UML'98—Beyond the Notation. First International Workshop*, J. Bézivin and P.-A. Muller, Eds. Mulhouse, France. LNCS, vol. 1618, 1998, pp. 281–286.

[14] A. Cavarra, "Inter-agent data flow analysis of business components," in *Proc. Australian Software Engineering Conference*, 2009, pp. 237-245.

[15] T. Inous and S. Honiden, "A method for data flow analysis of business components," in *Proc. 14th International ACM Sigsoft Symposium on Component Based Software Engineering*, ACM New York, NY, USA, 2011, pp. 51-60.

S. Obaid is a MS student at university of Arid Agriculture, Rawalpindi, Pakistan. He got his BS degree in software engineering from International Islamic University, Islamabad, Pakistan in 2010. His major field of studies includes modeling, testing and quality assurance.

He is working as a quality assurance engineer at Alachisoft Pakistan from 2012. The major responsibilities are testing NCache and JvCache application on Linux and Windows platform where the tasks includes test case specification, sanity design and execution, benchmarking, stress and load testing. His recent work in Alachisoft is Network Virtualization by Hyper-V and testing distributed cache application on network.

S. Asghar is an associate professor/director at University Institute of Information Technology, PMAS-Arid Agriculture University, Rawalpindi, Pakistan. He graduated with honors in Computer Science from the University of Wales, United Kingdom in 1994. He obtained his Ph.D. degree from Monash University, Melbourne, Australia in 2006. His major fields of interest include Data Mining and Business Intelligence, Decision Support Systems, Model Management and Disaster Management Systems.

He was serving as an associate professor of Computer Science, Department of Computer Sciences, Faculty of Engineering and Applied Sciences, Mohammad Ali Jinnah University, Islamabad, Pakistan. He also worked as an assistant professor of Computer Sciences and head of R&D at Shaheed Zulfikar Ali Bhutto Institute of Science and Technology, Islamabad, Pakistan. Previously, he was a research associate and assistant lecturer in Clayton School of Information Technology, Faculty of Information Technology at Monash University, Melbourne, Australia. From 1994 to 2002, he worked as a senior software engineer in a software company in Islamabad.

Dr. Asghar is a member of the Australian Computer Society (ACS), IEEE and also a higher education commission approved supervisor. He is in the Editorial Team of well reputed Scientific Journals. He has also served as a program committee member of many International Conferences.

M. Naeem is a Ph.D. scholar at department of computer science, Mohammad Ali Jinnah University Islamabad Pakistan. His research area includes machine learning, software engineering, semantic computing, text retrieval, graph mining, classification and data mining.