AUTOMATED SEMANTICS TREATMENT OF SEQUENCE DIAGRAM DEFINING GRAMMAR RULES

Fahad Alhumaidan and Nazir Ahmad Zafar
College of Computer Sciences and IT
King Faisal University, Hofuf, Saudi Arabia
Emails: {nazafar, falhumaidan}@kfu.edu.sa

ABSTRACT
UML diagrams being graphical in nature have informal semantics and it is difficult to develop automated tools for conversion and transformation of the diagrams. Formal methods are proved to be effective for semantics analysis of software systems. However, usage of formal methods is not very welcomed at early stages of software development. Hence, linking UML and formal techniques is needed to address the deficiencies existing in both approaches. In this paper, an approach is developed for transformation of simple sequence diagram by defining grammar rules. Formal specification of the procedure is described using Z notation by capturing hidden semantics under the diagrams. The model is analyzed and validated using Z/Eves tool. We believe that resultant approach will be useful for developing automated tools for modeling and verification of software systems.

KEY WORDS
Automation, UML Sequence diagram, semantics analysis, grammar rules, Z notation

1. Introduction

Although UML is accepted as a de-facto standard for development of object oriented systems but its diagrams are graphical in nature and are prone to causing errors [1]. The hidden semantics of the diagrams allows ambiguities at design level. For example, model in UML may have multiple interpretations and someone may not be able to understand what is put under the diagrams. Formal methods having well-defined semantics are at the early stage of development. A linkage of UML diagrams and formal methods will enhance the modeling power by defining semantic rules over the diagrams [2].

There exits few work in this area because the hidden semantics under UML diagrams cannot be transformed easily into formal notations. In the most relevant work, a mechanism for verifying sequence diagram is proposed by describing event-based deterministic finite automata from UML interaction diagram [3]. This is an interesting piece of work which is taken as starting point. In [4], a solution is proposed by translating UML sequence diagram combining description logic and computation tree logic. Statics analysis of UML interaction diagram is provided in [5] to check the well-formed-ness of the diagram. Jackson et al. [6], have developed Alloy Constraint Analyzer tool for description of systems whose state space involves relational structures. A study is presented based on web-service composition technique for cooperative composition modeling language [7]. An approach is demonstrated in [8] using XML to visualize TCOZ models into UML diagrams. An algorithmic approach is developed to check a consistency between sequence and state diagrams [9]. A procedure of creating tables and SQL code for Z specifications to UML diagrams is described in [10]. Intelligent approach of fusion recognition is described using petri-nets and fuzzy logic in [11]. An integrated approach is developed by combining B and UML in [12]. Kim et al., present a framework by integrating Object-Z and UML for requirements elicitation by a case study [13]. A tool is developed which takes class diagram and produces a list of comments on the diagrams in [14]. Few other relevant works can be found in [15-20]

In this paper, systematic procedure for formalizing and verifying sequence diagram is presented by defining grammar rules. The preliminary result of this research were presented in [21]. Advanced concepts, for example, loops, options, alternatives and reference are not considered. Cash withdraw from an ATM system is taken as a case study. First of all, a model of the system is presented using sequence diagram. Then state diagram is created by identifying states and transitions based on the objects and messages considering the time sequence same as in [21]. It is noted that many states of an object may exit in the life of an object. In the next, a mapping is defined to develop grammar for the diagram. Formal analysis of the transformation procedure is generalized based on the case study using Z notation. Z is used because it is a model oriented specification language used at an abstract level. The Z/Eves tool is used for model analysis because it is powerful one for analyzing the specification. Rest of the paper is organized as:

In section 2, transformation procedure from sequence to state diagram is presented. Formal specification of the procedure is described in section 3. Model analysis is given in section 4. The work is concluded in section 5.

2. Transformation of Sequence Diagram

In this section, critical analysis of sequence diagram is provided. Then formal procedure from sequence diagram to state diagram is presented by taking a case study of ATM cash withdraw system. Finally, grammar is developed to be used for further transformation.

2.1 ATM Cash Withdraw Case Study

The UML sequence diagram is used to realize details under the use cases and shows the interaction between objects by the roles. Sequence diagrams model messages for analysis and design for behavior interaction. The diagram represents messages and interactions in two dimensions. The interaction is in horizontal dimension whereas time is defined in the vertical line by resulting a two dimensional model as shown in Figure 1.

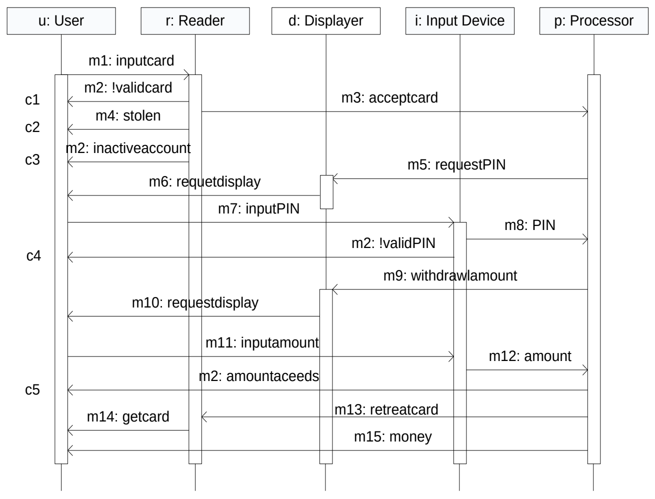

Figure 1. Sequence diagram for cash withdraw

UML sequence diagram is good modeling tool because it provides a dynamic view showing behavior which is not possible to extract from static system. Another important feature is its capability to represent parallelism between the complex components. The sequence diagram helps to discover architectural view and logical statements needed to define the system. Because of good modeling approach, sequence diagrams can be integrated easily because of the time dimension. In sequence diagram, object interaction, sequence order, responsibilities, functionalities and timings issues can be easily addressed. The diagram also facilitates the documentation at various levels of abstraction which is not easy when it is required to create from the static part of the system. Sequence diagram of ATM system as in the figure for cash withdraw is presented. At first the card is verified then PIN is entered for authentication. Finally, the cash is withdrawn if requested amount is less than the current balance of the customer.

2.2 Transformation Procedure

Sequence diagram in Figure 1 is transformed to state diagram as shown in Figure 2.

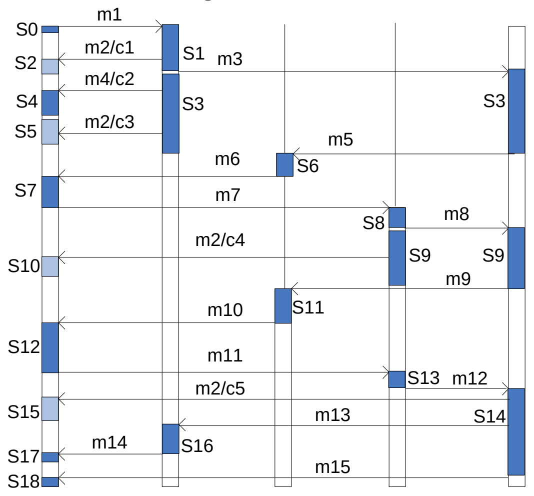

Figure 2. State diagram based on sequence diagram

In the transformation, each object may have many states. For example, the object user has ten states and the object reader has three states. It is noted that same message can be executed from two different pairs of states. For example, the message m2 is same for all the pairs of states (s1, s2), (s1, s4), (s3, s5), (s9, s10) and (s14, s15) which is repeated in case of failure of the transaction. A message may have execution condition. For example, c1, c2, c3, c4 and c5 are the message conditions.

Table 1
Mapping defining grammar for sequence diagram

#	Message	Production
1	(S0, m1, S1, null)	S0⇒m1S1, null
2	(S1, m2, S2, c1)	S1⇒m2S2, c1
3	(S1, m3, S3, null)	S1⇒m3S3, null
4	(S1, m4, S4, c2)	S1⇒m4S4, c2
5	(S3, m2, S5, c3)	S3⇒m2S5, c3
6	(S3, m5, S6, null)	S3⇒m5S6, null
7	(S6, m6, S7, null)	S6⇒m6S7, null
8	(S7, m7, S8, null)	S7⇒m7S8, null
9	(S8, m8, S9, null)	S8⇒m8S9, null
10	(S9, m2, S10, c4)	S9⇒m2S10, c4
11	(S9, m9, S11, null)	S9⇒m9S11, null
12	(S11, m10, S12, null)	S11⇒m10S12, null
13	(S12, m11, S13, null)	S12⇒m11S13, null
14	(S13, m12, S14, null)	S13⇒m12S14, null
15	(S14, m2, S15, c5)	S14⇒m2S15, c5
16	(S14, m13, S16, null)	S14⇒m13S16, null
17	(S16, m14, S17, null)	S16⇒m14S17, null
18	(S14, m15, S18, null)	S14⇒m15S18, null

The transformation procedure from state diagram to grammar development is listed in Table 1. In the table, the tuple (Si, mk, Sj, cp) represents that the message mk is executed from state Si to state Sj under the condition cp. For every message between two different states, a production rule is created. If there is no condition before the execution of a message then null condition is supposed. It is noted that S2, S4, S5, S10, S15and S18 are final states, however, S18 is the final state after successful execution of the procedure. Rest of all states, are failure of the operation.

Grammar Rules
After deriving rules from the messages, as in the table, whole set of productions is listed below. The null productions are added for termination of the process. The same sequence of derivations can be represented by the derivation tree for parsing of a scenario.

$S0 \Rightarrow m1S1$, null; $S1 \Rightarrow m2S2$, c1| m3S3, null| m4S4, c2; $S2 \Rightarrow \in$; $S3 \Rightarrow m2S5$, c3| m5S6, null; $S4 \Rightarrow \in$; $S5 \Rightarrow \in$; $S6 \Rightarrow m6S7$, null; $S7 \Rightarrow m7S8$, null; $S8 \Rightarrow m8S9$, null; $S9 \Rightarrow m2S10$, c4| m9S11, null; $S10 \Rightarrow \in$; $S11 \Rightarrow m10S12$, null; $S12 \Rightarrow m11S13$, null; $S13 \Rightarrow m12S14$, null; $S14 \Rightarrow m2S15$, c5|m13S16, null| m15S18, null; $S15 \Rightarrow \in$; $S16 \Rightarrow m14S17$, null; $S17 \Rightarrow \in$; $S18 \Rightarrow \in$

Derivation
Any possible scenario of the diagram can be derived for validation by the above grammar. For example, the scenario m1m3m5m6m7m8m9m10m11m12m15 can be validated by the sequence of derivations as below:

$S0 \Rightarrow m1S1$
$\Rightarrow m1m3S3$
$\Rightarrow m1m3m5S6$
$\Rightarrow m1m3m5m6S7$
$\Rightarrow m1m3m5m6m7S8$
$\Rightarrow m1m3m5m6m7m8S9$
$\Rightarrow m1m3m5m6m7m8m9m10S12$
$\Rightarrow m1m3m5m6m7m8m9m10m11S13$
$\Rightarrow m1m3m5m6m7m8m9m10m11m12S14$
$\Rightarrow m1m3m5m6m7m8m9m10m11m12m15S18$
$\Rightarrow m1m3m5m6m7m8m9m10m11m12m15$.

3. Formal Analysis

In this section, formal analysis of transformation procedure is described using Z notation. At first, the sequence diagram consisting of objects and messages is specified. The time sequence is given primary importance in specification of the diagram. Then state diagram is created based on the sequence diagram. Finally, grammar is developed to be useful for derivation of all possible scenarios based on the diagram.

There can be many states of an object of sequence diagram. Hence state is defined before specification of an object. The state is defined by the schema, State, which consists of three variables that is state name, start time and end time. To declare types of name, start and end times SName and Time are used at an abstract level of specification in Z. A schema consists of two parts namely definition and predicate parts. In definition part of the schema, variables are defined whereas invariants are defined in the predicate part.

$[SName]; Time == \mathbb{N}$

$$
\begin{array}{|l}
\textit{State} \\
\hline
sname: Sname;\ stime, etime: Time \\
\hline
stime \leqslant etime \\
\hline
\end{array}
$$

An object is represented by the schema Object which consists of six components namely object name, start time, end time, sequence of states, attributes and methods in the diagram. It is stated that the life line of an object is described by the start and end times variables. The object name and attributes are declared as a set type as specified above. The methods is defined as a partial function between object attributes.

$[OName]$

$[Attribute]$

$$
\begin{array}{|l}
\textit{Object} \\
\hline
oname: OName \\
ostart, oend: Time \\
states: \mathrm{seq}\ State \\
attributes: \mathbb{F}\ Attribute \\
methods: Attribute \rightarrow\!\!\!\!\!+\ Attribute \\
\hline
states \neq \langle\rangle \\
\#\, states \geqslant 1 \\
\Rightarrow (\exists s1, s2: State \mid s1 \in \mathrm{ran}\ states \wedge s2 \in \mathrm{ran}\ states \\
\quad \bullet\ states\ 1 = s1 \wedge states\ (\#\, states) = s2 \\
\quad\quad \Rightarrow ostart \leqslant s1\ .\ stime \wedge s2\ .\ etime \leqslant oend) \\
\forall i: \mathbb{N} \mid \#\, states \geqslant 1 \wedge i \in 1\ ..\ \#\, states - 1 \\
\quad \bullet\ \exists s1, s2: State \\
\quad\quad \bullet\ states\ i = s1 \wedge states\ (i + 1) = s2 \Rightarrow s1\ .\ etime \leqslant s2\ .\ stime \\
\forall input, output: Attribute \mid (input, output) \in methods \\
\quad \bullet\ input \in attributes \wedge output \in attributes \\
\hline
\end{array}
$$

The message in sequence diagram is defined by the schema Message, which consists of activation time, condition of execution, source and target objects. The activation time of a message is specified by the schema ActivationTime. It is stated that the start time is less than the finishing time of any message in the diagram. The next variable is condition that must be true before execution of a message. The condition has three values, i.e., true, false or null. The value null is used to represent that there is no triggering condition for the message. In predicate part of the schema, time ordering of the message is defined as an invariant.

Condition ::= *NULL* | *TRUE* | *FALSE*

```
__ActivationTime ________________________________
| starttime, endtime: ℕ
|______________________
| starttime < endtime
|_________________________________________________
```

```
__Message _______________________________________
| ActivationTime
| condition: Condition
| from, to: State
|______________________
| from . stime ⩽ starttime ∧ endtime ⩽ to . etime
|_________________________________________________
```

Formal specification of the sequence diagram is provided by the schema SequenceModel as given below. The schema contains two components, communicating objects and messages used in the sequence diagram. In predicate part, it is stated that for every message there exist two objects in the sequence diagram and vice versa. In sequence diagram, it is less focused on messages itself and more on the order in which these are executed. The first message starts from the left-top and subsequent messages are then followed following order of execution. The message sent to the receiving object is implemented by the receiving object.

```
__SequenceModel _________________________________
| objects: 𝔽 Object
| messages: 𝔽 Message
|______________________
| ∀o1, o2: Object | o1 ∈ objects ∧ o2 ∈ objects
|   • ∃s1, s2: State | s1 ∈ ran o1 . states ∧ s2 ∈ ran o2 . states
|   • ∃m: Message | m ∈ messages • m . from = s1 ∧ m . to = s2
| ∀m: Message | m ∈ messages
|   • ∃o1, o2: Object | o1 ∈ objects ∧ o2 ∈ objects
|   • ∃s1, s2: State | s1 ∈ ran o1 . states ∧ s2 ∈ ran o2 . states
|        • s1 = m . from ∧ s2 = m . to
|_________________________________________________
```

The state diagram was created from the sequence diagram as in Figure 2. Formal specification of the state diagram is described below by using the schema StateDiagram which consists of five components, that is, start state, all possible states of the diagram, messages, transformation function and set of final states. The definitions are given in first part and constraints are defined in the second part of the schema.

In the predicate part of the schema, it is stated that start state is an element of the total states of the sequence diagram. For any message there exist two states reachable after execution of the message. The transition function takes a state, checks guard condition and triggers the message by moving to the next state of the object. The set of final states is represented by final which is subset of the set of total states.

```
__StateDiagram __________________________________
| SequenceModel
| start: State
| states: 𝔽 State
| messages: 𝔽 Message
| delta: State × (Condition × Message) → State
| final: 𝔽 State
|______________________
| start ∈ states
| ∀s1, s2: State | s1 ∈ states ∧ s2 ∈ states
|   • ∃message: Message | message ∈ messages
|       • message . from = s1 ∧ message . to = s2
| ∀message: Message | message ∈ messages
|   • ∃s1, s2: State | s1 ∈ states ∧ s2 ∈ states
|       • s1 = message . from ∧ s2 = message . to
| ∀s1: State | s1 ∈ states
|   • ∃message: Message; cd: Condition; s2: State
|       | message ∈ messages ∧ s2 ∈ states ∧ (s1, (cd, message))
| ∈ dom delta
|       • delta (s1, (cd, message)) = s2
| ∀s: State | s ∈ final • s ∈ states
|_________________________________________________
```

proof of *StateDiagram$domainCheck*
prove by reduce

4. Model Analysis

Even formal specification of a complex system is written in any of the formal language, it may cause potential errors. This is because, for a moment, we don't have any computer tool which may guarantee about complete correctness of model of a complex system. The Z/Eves is a powerful tool used for analyzing formal specification of the model. The tool is integrated with various model analysis facilities providing rigorous checking of the system to be developed and has an automated deduction capability.

The syntax checking, type checking and theorem proving facilities of the Z/Eves tool are used for analysis of the model. It is noted that syntax and type checking do not require any interaction with the theorem proving facility of the tool. The domain checking facility allowed us to write meaningful properties of the system. It is observed that domain checking of model is much harder than the syntax and type checking of the model. Further, the syntax and type checking are performed automatically whereas one has to interact with the theorem proving facility to perform the domain checking. Furthermore, we observed that proof 'by reduce' was sufficient for formal specification of this transformation procedure for domain checking.

The schema expansion facility was used to unravel the specification of the diagrams and procedures which simplified the model results that were not easy otherwise to understand the specification. Prove by reduce is used for analyzing the formal specification. Some of the results of the model analysis are shown in the Table 2. In the Table, the first column shows name of the schema to be analyzed and evaluated, the second column is for syntax and type check, third for domain checking, fourth for reduction facility and the last one for the proof by reduction. The symbol Y in the table shows that all schemas are well written by syntax and domain checking. However the * symbol, after Y, shows that proof is done by the reduction technique.

Table 2. Results of model analysis

Schema Name	**Syntax Type Check**	**Domain Check**	**Reduction**	**Proof**
State	Y	Y	Y	Y
Object	Y	Y	Y	Y
ActivationTime	Y	Y	Y	Y
Message	Y	Y	Y*	Y
SequenceModel	Y	Y	Y*	Y
StateDiagram	Y	Y	Y*	Y

5. Conclusion

An exhaustive survey of existing work was performed before starting this work. Some interesting work was found as discussed in section I but our work is different from others because of capturing hidden semantics under the graphical notations. A comparison to most relevant work is presented. For example, in [3] a transformation mechanism from sequence diagram to event deterministic finite automata is provided. There were two major drawbacks in that work. Firstly, the resultant automaton is not deterministic because there is no state for some transitions in the automata. Secondly, the verification mechanism does not provide full support for correctness.

This work is part of our project on formalization of UML diagrams to be useful for software development of complex systems [21-24]. In this paper, an approach is developed for transformation of UML sequence to state diagrams by removing flaws existing in the diagram. Then grammar is developed based on the state diagram for verifying messages and scenarios. The resultant approach will be useful in development of automated tools for construction and verification of software systems. Although we have taken a simple case study but the advantage of our approach is that a formal procedure of transformation from UML notations to mathematical model is described. Then algorithm is specified using Z notation and verification is provided using Z/Eves tool. The Z notation is used because of its abstract and expressive power [25]. The rich mathematical notations in Z made it possible to reason about behavior of graphical notations. The Z/Eves is a powerful tool used to analyze the specification [26].

In future work, the advanced concepts of sequence diagram will be considered and complete transformation algorithm from the diagram to formal models will be designed. It is noted that conversion of UML diagrams to mathematical models by synthesis of suitable notations is our major objective. Transition diagrams, graphs, grammar, etc. are the tools for developing the integrated approach.

Acknowledgement

We would like to thank Deanship of Scientific Research, King Faisal University, Saudi Arabia for their funding support to our project on formalization of UML diagrams for automating design and development processes in software systems.

References

[1] Yeung, W. L., Leung, K. R. P. H., Wang, J., Dong, W.: Improvements Towards Formalizing UML State Diagrams in CSP, Proceedings of 12th Asia Pacific Software Engineering Conference, Taiwan, 2005.

[2] Shroff, M., France, R. B.: Towards Formalization of UML Class Structures in Z, 21st International Conference on Computer Software and Applications, pp. 646-51, 1997.

[3] Chen, Z., Zhenhua, D.: Specification and Verification of UML2.0 Sequence Diagrams using Event Deterministic Finite Automata, 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement – Companion, pp. 41-46, 2011.

[4] Li, M., Ruan, Y.: Approach to Formalizing UML Sequence Diagrams, 3rd International Workshop on Intelligent Systems and Applications (ISA), pp. 1-4, 2011.

[5] Li, X., Liu, Z., Jifeng H.: A Formal Semantics of UML Sequence Diagram, Proceedings of the 2004 Australian Software Engineering Conference, 2004.

[6] Jackson, D., Schechter, I., Shlyakhter, I.: Alcoa: The Alloy Constraint Analyzer, Proceedings of International Conference on Software Engineering, 2000.

[7] Xiuguo, Z., Liu, H.: Formal Verification for CCML Based Web Service Composition, Information Technology Journal, 2011.

[8] Sun, J., Dong, J. S., Liu, J., Wang, H.: A XML/XSL Approach to Visualize and Animate TCOZ, Proc. of 8th Asia-Pacific Software Engineering Conference, pp. 453-60, 2001.

[9] Litvak, B.: Behavioral Consistency Validation of UML Diagrams, First International Conference on Software Engineering and Formal Methods, 2003.

[10] Moeini, A., Mesbah, R. O.: Specification and Development of Database Applications based on Z and SQL, Proceedings of 2009 International Conference on Information Management and Engineering, pp. 399-405, 2009.

[11] Shi, Z.: Intelligent Target Fusion Recognition Based on Fuzzy Petri Nets, Information Technology Journal, 11, pp. 500-03, 2012.

[12] Leading, H., Souquieres, J.: Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B, Proceedings of 9th Asia-Pacific Software Engineering Conference, 2002.

[13] Kim, S. K., Carrington, D. A.: An Integrated Framework with UML and Object-Z for Developing a Precise and Understandable Specification: The Light Control Case Study. Proceedings of Seventh Asia-Pacific Software Engineering Conference, pp. 240-48, 2000.

[14] Ali, N. H., Shukur, Z., Idris, S.: A Design of an Assessment System for UML Class Diagram, Int'l Conference on Computational Science and Applications, pp. 539–46, 2007.

[15] Miao, H., Liu, L., Li, L.: Formalizing UML Models with Object-Z, Proceedings of 4th International Conference on Formal Methods and Software Engineering, Springer, 2002.

[16] Mostafa, A. M., Manal, A. I., Hatem, E. B., Saad, E. M.: Toward a Formalization of UML2.0 Meta-model using Z Specifications, Proc. of 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/ Distributed Computing, 3, pp. 694-701, 2007.

[17] Sengupta, S., Bhattacharya, S.: Formalization of UML Diagrams and Consistency Verification: A Z Notation Based Approach. Proceedings of India Software Engineering Conference, pp. 151-52, 2008.

[18] Zafar, N. A.: Modeling and Formal Specification of Automated Train Control System using Z Notation, IEEE Multi-topic Conference (INMIC'06), pp. 438-43, 2006.

[19] Zafar, N. A., Khan, S. A., Araki, A.: Towards Safety Properties of Moving Block Railway Interlocking System, Int'l Journal of Innovative Computing, Information & Control, 2012.

[20] Sohail, F., Zubairi, F., Sabir, N. Zafar, N. A.: Designing Verifiable and Reusable Data Access Layer Using Formal Methods and Design Patterns, International Conference on Computer Modeling and Simulation, 2009.

[21] Zafar, N. A., Alhumaidan, F.: Scenarios Verification in Sequence Diagram, International Conference on Computer and Engineering Technology, Canada, 2103.

[22] Zafar, N. A.: Event-Action Based Model for Identification and Formalization of Relations in UML State Diagrams, Archives Des Sciences Journal, 65(4), 2012.

[23] Zafar, N. A., Alhumaidan, F.: Transformation of Class Diagrams into Formal Specification, International Journal Computer Science and Network Security, 11, 289-95, 2011.

[24] Alhumaidan, F.:A Critical Analysis and Treatment of Important UML Diagrams Enhancing Modeling Power, Intelligent Information Management, 4(5), pp. 231-37, 2012.

[25] Spivey, J. M.: The Z Notation: A Reference Manual. Englewood Cliffs NJ, Prentice-Hall, 1989.

[26] Meisels, I., Saaltink, M.: The Z/Eves Reference Manual, Version 1.5, TR-97-5493-03d, ORA Canada, 1997.

Dr. Fahad M. Alhumaidan graduated from University of Newcastle Upon Tyne, UK. Currently, he is working as Assistant Professor in Information System Department at CCSIT. He is Vice Dean at College of Computer Sciences and Information Technology (CCSIT), at King Faisal University, Saudi Arabia. He is also Chairman of Computer Science Department. He is responsible for chairing various technical and administrative committees at the college. His research areas include Software Engineering, Object-oriented Paradigm, Integration of UML and Formal Methods, Business Process Management, Workflow Systems, Soft aspects of Information System, E-Business, Network & Communication. He has contributed for various funded research projects and completed successfully by publishing the results produced in international journals and conferences proceedings.

Nazir A. Zafar was born in 1969 in Pakistan. He received his M.Sc. (Math. in 1991), M. Phil (Math. in 1993), and M.Sc. (Nucl. Engg. in 1994) from Quaid-i-Azam University, Pakistan. He was awarded PhD degree in computer science from Kyushu University, Japan in 2004. Currently, he is working as Associate Professor at the College of Computer Sciences and Information Technology (CCSIT), King Faisal University (KF), Saudi Arabia. He is the founder of various research groups in the area of software engineering and formal methods. His current research interests are modelling of systems using formal approaches, integration of approaches, safety and security critical systems, etc. He is an active member of Pakistan Mathematical Society. Dr. Zafar has lectured at national and international level promoting use and applications of formal methods at academic as well as at industrial level. He has also administrative experience and qualities. For example, he has worked as Dean, Faculty of Information Technology, University of Central Punjab, Pakistan. He has leaded various scientific committees related to research and academic activities.