Scenarios Verification in Sequence Diagram

Nazir Ahmad Zafar and Fahad Alhumaidan

College of Computer Sciences and Information Technology, King Faisal University, Hofuf, Saudi Arabia
nazafar@kfu.edu.sa; falhumaidan@kfu.edu.sa

Abstract: The Unified Modeling Language (UML) has become a de-facto standard for analysis, design models and specification of object oriented software systems. UML structures being graphical in nature have informal semantics and, hence, it is difficult to develop verification tools for UML specification. Formal methods are proved to be useful at requirements analysis, specification and design level. Hence linking of UML and formal notations is required to overcome the deficiencies existing in the UML diagrams. In this paper, an approach is developed by transformation of UML sequence diagram to transition graph using Z notation. Then formal specification is described by capturing the hidden semantics by focusing on the syntax and semantics. Finally, scenarios are generated from the transition graph and verified to show correctness of the diagram. We claim that this approach will be effective and useful for developing automated tools for verification of UML sequence diagrams. The resultant formal models are analyzed and validated using Z/Eves tool.

[Nazir Ahmad Zafar and Fahad Alhumaidan. **Scenarios Verification in Sequence Diagram.** *J Am Sci* 2013;9(11):287-293]. (ISSN: 1545-1003). http://www.jofamericanscience.org. 38

Keywords: UML; Sequence diagram; Z notation; Integration; Verification

1. Introduction

Although UML has become a de-facto standard for development of object oriented systems but its semantics is semi-formal allowing ambiguities in design of systems (Borges and Mota, 2003) and (Yeung et al., 2005). The same system can be described by multiple notations which may cause inconsistencies. Formal methods have a well-defined syntax and semantics but are not welcomed at industrial level. To get full benefits, UML diagrams and formal methods can be linked for enhancing the modeling power of these approaches (Shroff and France, 1997).

Although there exits a lot of work on integration of approaches but there does not exists much work on linking UML diagrams with formal approaches. This is because the hidden semantics under the UML diagrams cannot be transformed easily into formal notations. In the existing work, a mechanism for verifying sequence diagram is proposed by creating an event deterministic finite automata model from UML interaction diagram Z (Chen and Zhenhua, 2011). This work is interesting and starting point for us. In (Li and Ruan, 2011), an effort is done to propose a solution by translating UML sequence diagrams by combining the features of the description logics and computation tree logic. Static semantics of UML interaction diagrams is provided in (Li et al., 2004) to check the well-formed-ness of the diagram. A study is presented by formal verification method for Cooperative Composition Modeling Language based on web-service composition technique (Xiuguo and Liu, 2011). An approach is demonstrated in (Sun et al. 2001) using XML to visualize TCOZ models into various UML diagrams. An algorithmic approach is developed to check a consistency between UML sequence and state diagrams (Litvak, 2003). In (Moeini and Mesbah, 2009), it is described a way of creating tables and SQL code for Z specifications according to UML diagrams. In (Leading and Souquieres, 2002), an integrated approach is developed by combining B and UML. Kim et al., 2000, present a framework by integrating UML and Object-Z to support requirements elicitation supported by a case study. A tool is developed in (Ali et al., 2007) which takes class diagram and produces a list of comments on the diagrams. Few other relevant works can be found in (Miao et al. 2002), (Mostafa et al., 2007), (Sengupta and Bhattacharya, 2008), (Sarma et al., 2007), (Yang et al., 2010), (Ameedeen and Bordbar, 2008), (Zafar, 2006), (Zafar et al., 2012), (Sohail et al., 2009).

Main contribution of the work is to provide an effective and systematic mechanism for formalizing and verifying sequence diagram. The diagram is assumed as a simple one in which advanced concepts, for example, loops, options, alternatives are not considered. First of all, formal specification of sequence models is described. In the next, the sequence model is transformed to transition graph by capturing semantics hidden under the diagram. The order of messages and time sequence are given primary importance. Further, scenarios are defined based on the transition graph and are validated based on the transition function. For an effectiveness of the approach, transformation procedure is explained by taking a case study of

ATM cash withdraw system. Formal analysis is provided by Z which is a model oriented specification language. Z/Eves tool is used for model analysis because it is a powerful one for analyzing the specification. The tool provides various exploration techniques to prove correctness of the properties. Rest of the paper is organized as: In section 2, transformation mechanism from sequence diagram to transition graph is provided. Formal specification is described in section 3. Conclusion and future work are discussed in section 4.

2. Sequence Diagram to Transition Graph

The mechanism of transformation from UML sequence diagram to transition graph is presented by a case study in this section.

A. Case Study

Sequence diagram is important and good modeling tool because it provides a dynamic view showing behavior which is not possible to extract from statics of the system. The sequence diagram helps to discover architectural view and logical statements needed to define the system at early stages of the design. Separate sequence diagrams can be integrated easily because of the time dimension.

Sequence diagram represents flow of events, messages and interactions between objects in two dimensions. The interaction is in horizontal dimension and time is defined in the vertical line by making a two dimensional model as in Figure 1. In the figure, sequence diagram of ATM 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 balance.

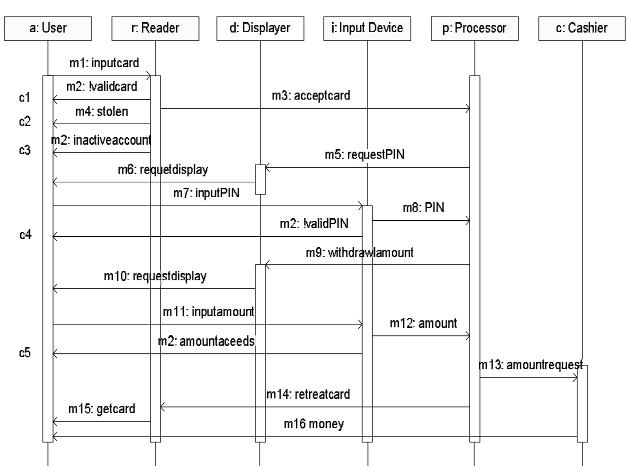

Figure 1. Sequence diagram for cash withdraw

B. Transformation Procedure

The transition procedure from sequence diagram to transition graph is explained in Table 1. In the table, a, r, d, i, p and c represent the objects: user, reader, displayer, input device, processor and cashier respectively. The state of message initiating object is termed as initial state. If an object a sends a message m to another object b under the guard condition c then the next state is represented by s = (m, a, b, c). It is noted that for two different next states, the triggering messages might be same. For example, in the table, s2 = (m2, r, a, c1) and s5 = (m2, r, a, c3), the messages are same but the states are different. In case of s2 = (m2, r, a, c1), the card is rejected because it is invalid card whereas for s5 = (m2, r, a, c3) it is rejected because the account is inactive. If there are more than one guard conditions in the diagram then at least one must be true. If both conditions are false then the last one option "PIN request" is triggered for validity of the diagram.

Table 1. Relationship of states and messages

State	Event	Action	State	Event
s0	-		s10	(m2, i, a, c4)
s1	(m1, a, r, -)	card inserted	s11	(m9, p, d, -)
s2	(m2, r, a, c1)	card rejected	s12	(m10, d, a, -)
s3	(m3, r, d, -)	card accepted	s13	(m11, a, i, -)
s4	(m4, r, a, c2)	card retained	s14	(m12, i, p, -)
s5	(m2, r, a, c3)	card rejected	s15	(m2, i, a, c5)
s6	(m5, p, d, -)	PIN request	s16	(m13, p, c, -)
s7	(m6, d, a, -)	display PIN	s17	(m14, p, r, -)
s8	(m7, a, i, -)	PIN entered	s18	(m15, r, a, -)
s9	(m8, i, p, -)	PIN process	s19	(m16, c, a, -)

Figure 2 shows the resultant graph consisting of set of states and transitions. There are four types of states, that is, initial state, rejecting, internal and accepting states. The initial state is represented by minus sign inside the circle. It is in fact first state of the object (User) before inserting the card into the machine. The rejecting states are represented by the light shaded circles in the transition graph. In the figure, the set {s2, s4, s5, s10, s15} is a collection of rejecting states. If any of these states is reached, cash withdraw operation is terminated resulting a failure of operation. The only accepting state is s19 which is represented by the dark color. It is noted that objects communication is represented from top to down and time sequence is captured by left to right by traversing the Figure 2. There are only six possible scenarios which can be generated by traversing the graph using top-left approach. The set of possible scenarios is generated from transition graph as: S1 = <s0, s1, s2>; S2 = <s0,

s1, s3, s4>; S3 = <s0, s1, s3, s5>; S4 = <s0, s1, s3, s6, s7, s8, s9, s10>; S5 = <s0, s1, s3, s6, s7, s8, s9, s11, s12, s13, s14, s15>; S6 = <s0, s1, s3, s6, s7, s8, s9, s11, s12, s13, s14, s16, s17, s18, s19>.

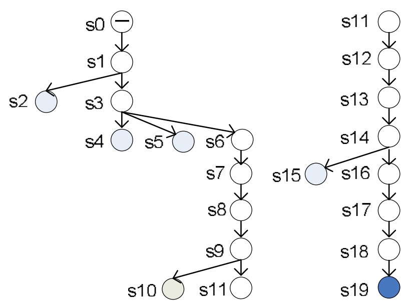

Figure 2. Transition graph based on sequence diagram

3. Formal Analysis

In this section, a generic formal approach is described based on the sequence diagram in Figure 1 and transition graph in Figure 2. Formal definition of class diagram, the hidden semantics under the sequence diagram and interaction among the objects is defined. The diagram is transformed to transition diagram. Finally, all possible scenarios of the diagram are generated and verified.

A. Static Model

The class diagram is represented as: Class Diagram = <classes: P(C); relationship: C x Relation x C>, where C is class and P(C) stands for power set classes. The relation among classes can be association, generalization, aggregation, composition. Formal specification of class is presented using Class schema which consists of three variables namely, cname, attributes and methods. The attributes variable is used for storing class attributes and methods is to define class operations which is a partial function between the classes. The Name and Attribute are used at an abstract level of specification. The schema consists of two parts namely definition and predicate parts. In definition part, variables are defined and invariants are defined in the predicate part. It is stated that input and output of the methods are attributes of class.

[*Name*]; [*Attribute*]

Class
cname: Name; attributes: *Attribute*
methods: Attribute *Attribute*
input, output: Attribute *input* *output* *methods*
input *attributes* *output* *attributes*

An object is created from the class by the notation object: Class consisting of same attributes but different values. The invariants object are same as in the Class schema.

Object
object: Class

A class diagram is defined by the schema CM containing its main components. The first components *classes* is used to define set of all classes. The next three variables, *sclasses, whole,* and *parts* are used to define sets of subclasses, whole and part classes respectively. Finally, four types of relations, *association, generalization, aggregation* and *composition* are considered. The first one relationship *association* shows how object are connected to each other. The generalization relationship is between a child and parent where a child receives all the attributes and operations that are defined in the parent. An aggregation relationship describes a group of objects and the way of interaction with each other. For example, relationship between college and university can be defined by the aggregation relationship showing that the college is a part of the university. The composition relationship is a special type of aggregation which is stronger than aggregation. In aggregation if whole is destroyed the part may exist whereas in case of composition part cannot exist without the whole class.

CM
classes: *Class; classes, whole, parts:* *Class*
association, generalization: Class *Class*
aggregation, composition: Class *Class*
cl1, cl2: Class *cl1* *cl2* *association* *cl1* *classes*
cl2 *classes*
cl: Class; sc: Class *sc* *cl* *generalization* *cl* *classes*
sc *sclasses*
c1, c2, wc: Class; pc: Class *c1* *pc* *aggregation* *c2*
pc *aggregation* *c1 = c2*
wc1, wc2, wc: Class; pc: Class *wc1* *pc* *composition*
wc2 *pc* *composition* *wc1 = wc2*

B. Dynamic Model

In the sequence diagram, vertical dimension represents life line of objects and messages and the horizontal dimension shows change in states of the objects. For specification, event is defined by the schema *Event* which includes four variables. The first one *name* is used to define name of the event. The next two variables, *first* and *second*, represent initial and next states of communicating objects. The last one variable *condition* must be true before execution of the event. The *condition* has three values, i.e., null, true or false. The value null is used to represent that there is no triggering condition.

[*EName*]; *Condition* ::= *NULL* | *TRUE* | *FALSE;* [*State*]

```
Event
name: EName; first, second: State; condition: Condition
```

Lifeline of an object is defined by the schema *LifeLine* given below. The *min* and *max* are used to represent creation and destruction times. In the predicate part, it is stated that creation time is less than the destruction time.

```
LifeLine
min, max:
min  max
```

```
SObject
Object; states: seq State; lline: LifeLine; olife:
lline . min  olife  olife  lline . max
```

The object used in sequence diagram needs all of its possible states and is called sequence object which is defined by the schema *SObject*. In the predicate part, it is stated that life line of the object is within the time limits. 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 the message.

The *Event* schema is reused in the definition of *Message* to define change in states of objects. The *ActivationTime* is included to describe activation time of a message in the sequence diagram. Finally *from* and *to* variables are used to represent communicating objects in the message. In predicate part of the schema, time ordering is defined.

```
ActivationTime
starttime, endtime:
starttime  endtime
```

```
Message
Event; ActivationTime; from, to: SObject
from.lline.min  starttime  starttime  endtime
endtime  to.lline.max  to . olife  from . olife
```

Formal specification of validating messages in sequence diagram is provided by the *SSM* model given below. The schema contains class model, communicating objects and messages used in the diagram. In predicate part of the schema, it is stated that every object in the diagram is an object of some class diagram. For every message in the sequence diagram, there exist two objects of some classes in the diagram such that there is a relation, association, generalization aggregation or composition among these objects. For every two objects in sequence diagram, there is a sequence of messages among the objects in the diagram. And for every message in the diagram, there exist two objects which can communicate.

```
SSM
CM; name: Name; objects:  SObject; messages:  Message
 o: SObject  o  objects   c:Class  c  classes  c = o . object
 m: Message  m  messages
  c1,c2:Class;o1,o2: SObject  c1  classes  c2  classes
 o1 objects  o2 objects  c1=o1 . object  c2 = o2 .object
  c1 c2  association  c1 c2  generalization
  c1 c2  aggregation  c1 c2  composition
 o1, o2: SObject  o1  objects  o2  objects
 m: Message  m  messages  m . from = o1  m . to = o2
 m: Message  m  messages
  o1, o2: SObject  o1  objects  o2  objects  o1 = m .
from  o2 = m . to
```

Dynamic behavior of the sequence diagram is represented using *DSM* which contains *SSM* as defined above. In the schema, *start* variable is used to define the start state of the object triggering the first event. The *states* variable represents sequence of all possible states of the objects. The set of events in the diagram is represented by the *events* variable. The most important is the transition function *next* which takes a state, checks guard condition and triggers the event by moving to the next state of an object. The set of final states is represented by *final*.

```
DSM
SSM; start: State; states: seq State; events: seq Event
next: State  Condition  Event  State; final:  State
start  ran states
 s1, s2: State  s1  ran states  s2  ran states
  event: Event  event  ran events  event . first = s1
event . second = s2
 event: Event  event  ran events   s1, s2: State  s1  ran
states  s2  ran states
  s1 = event . first  s2 = event . second
 s1: State; cd: Condition  s1  ran states   ev: Event; s2:
State  ev  ran events  s2  ran states  s1 cd ev
dom next  next s1 cd ev = s2
 s: State  s  final  s  ran states
```

In the predicate part, it is stated that start state belongs to the set of possible states. For any two states, there is an event which triggers for moving control from one state to the other. For any event, there are two states completing the execution of the event after guard condition is true. For every state, there exists an event and new state in the domain of transition function. Every final state is in the set of possible states.

The sequence diagram is transformed to TG *STGraph* in which *start*, *states* and *events* have the same meaning as explained in *DSM*. The transition function *transition* takes a state and triggering event as input and moves to the next state of an object of the diagram. The last one is the set of accepting states of the transition graph.

```
┌ STGraph ─────────────────────────
│ initial: State; states: ℙ State; events: ℙ Event
│ transition: State × Event ⇸ State; accepted: ℙ State
├─────────────────
│ initial ∈ states ∧ (∀ s: State • s ∈ accepted ⇒ s ∈ states)
│ ∀ s1: State • s1 ∈ states
│  ⇒ (∃ event: Event; s2: State • event ∈ events ∧ s2 ∈ states
│  ∧ transition (s1, event) = s2)
└─────────────────────────────────
```

A scenario is a sequence of events which may execute after following the order of the messages. For validation, transition graph and scenario? are inputs to the schema given below and validation process is described in the predicate part. It is stated that the first and last elements of the scenario are in the set of events of the transition graph. Every element, other than first and last events, in the scenario is also in the transition graph.

Scenario == seq Event

```
┌ ScenarioValidation ──────────────────
│ STGraph; scenario?: Scenario
├─────────────────
│ # scenario? ≥ 1
│ ∃ event: Event • event ∈ events ∧ event ∈ ran scenario?
│   ∧ scenario? 1 = event ∧ event . first = initial
│ ∃ event: Event • event ∈ events ∧ event ∈ ran scenario?
│   ∧ scenario? (# scenario?) = event ∧ event . second ∈
│ accepted
│ ∃ sss: seq State • ran sss ⊆ states ∧ # sss = # scenario? + 1
│   ∧ (∀ i: ℕ • i ∈ 1 .. # scenario? - 1
│      ⇒ (∃ event1, event2: Event • event1 ∈ ran scenario?
│         ∧ event2 ∈ ran scenario? ∧ event1 . second ∈ states
│         ∧ event2 . first ∈ states ∧ scenario? i = event1 ∧
│ scenario? (i + 1) = event2 ∧ event1 . second = event2 . first))
└─────────────────────────────────
```

The sequence diagram is validated using the schema *SDValidation*. Transition graph and set of scenarios are given as inputs to the schema and validation process is described as same as in case of above schema.

```
┌ SDValidation ─────────────────────
│ STGraph; scenarios?: ℙ Scenario
├─────────────────
│ ∀ sc: Scenario • sc ∈ scenarios? ⇒ # sc ≥ 1
│ ∀ sc: Scenario • sc ∈ scenarios? ∧ # sc ≥ 1 ⇒ (∃ event: Event
│ • event ∈ events ∧ event ∈ ran sc ∧ sc 1 = event ∧ event . first
│ = initial)
│ ∀ sc: Scenario • sc ∈ scenarios? ∧ # sc ≥ 1 ⇒ (∃ event: Event
│ • event ∈ events ∧ event ∈ ran sc ∧ sc (# sc) = event ∧ event .
│ second ∈ accepted)
│ ∀ sc: Scenario • # sc ≥ 1 ∧ sc ∈ scenarios? ⇒ (∃ sss: seq State
│ • ran sss ⊆ states ∧ # sss = # sc + 1 ∧ (∀ i: ℕ • i ∈ 1 .. # sc - 1
│ ⇒ (∃ e1, e2: Event • e1 ∈ ran sc ∧ e2 ∈ ran sc ∧ e1 . second ∈
│ states ∧ e2 . first ∈ states ∧ sc i = e1 ∧ sc (i + 1) = e2 ∧ e1 .
│ second = e2 . first)))
└─────────────────────────────────
```

4. Conclusion

This work is part of our ongoing project on digging semantics of UML diagram (Zafar, 2012), (Zafar and Alhumaidan, 2011) and (Alhumaidan, 2012). In this paper, an approach is developed by identifying ambiguities and removing flaws by verifying all the possible scenarios existing in the diagram. Further, consistency between sequence and class diagram is checked by verifying messages existing in both the diagrams. Automatic test cases can be generated from our transition graph, is another significance of this approach. The resultant approach can be useful in development of automated tools for generation and verification of the system's specification. Although we have taken a simple sequence diagram in which advanced concepts, for example, loops, options, alternative, etc. are not considered but the advantage of our approach is that the diagram is fully transformed to transition graph. Then graph is specified using Z notation and formal verification is provided using Z/Eves tool. Z is used because of its abstract and expressive power (Spivey, 1989). The rich mathematical notations in Z made it possible to reason about behavior of sequence diagram rigorously. Several type checking tools exist to support the specification. The Z/Eves is a powerful tool used here to analyze the specification (Meisels and Saaltink, 1997).

Acknowledgements:

We would like to thank Deanship of Scientific Research, King Faisal University, Saudi Arabia for their funding support to our project.

Corresponding Author:
Dr. Nazir Ahmad Zafar
Department of Computer Science
College of Computer Sciences and IT
King Faisal University
Alahsa, 31982, Saudi Arabia
E-mail: nazafar@kfu.edu.sa

References

1. R. Borges and A. Mota, Integrating UML and Formal Methods, Electronic Notes in Theoretical Computer Science, 184, pp. 97-112, 2003.
2. W. L.Yeung, K. R. P. H. Leung, J. Wang and W. Dong, Improvements Towards Formalizing UML State Diagrams in CSP, Proc. of 12th Asia Pacific Software Engineering Conference, Taiwan, 2005.
3. M. Shroff and R. B. France, Towards Formalization of UML Class Structures in Z, 21st Int'l Conference on Computer Software and Applications, pp. 646-51, 1997.
4. Z. Chen and D. Zhenhua, Specification and Verification of UML2.0 Sequence Diagrams using Event Deterministic Finite Automata, 2011 Fifth Int'l Conference on Secure Software Integration and Reliability Improvement–Companion, pp. 41-46, 2011.
5. M. Li and Y. Ruan, Approach to Formalizing UML Sequence Diagrams, 3rd Int'l Workshop on Intelligent Systems and Applications (ISA), pp. 1-4, 2011.
6. X. Li, Z. Liu and H. Jifeng, A Formal Semantics of UML Sequence Diagram, Proc. of the 2004 Australian Software Engineering Conference, 2004.
7. Z. Xiuguo and H. Liu, Formal Verification for CCML Based Web Service Composition, Information Technology Journal, 2011.
8. J. Sun, J. S. Dong, J. Liu and H. Wang, A XML/XSL Approach to Visualize and Animate TCOZ, Proc. of 8th Asia-Pacific Software Engineering Conference, pp. 453-60, 2001.
9. B. Litvak, Behavioral Consistency Validation of UML Diagrams, First Int'l Conference on Software Engineering and Formal Methods, 2003.
10. A. Moeini and R. O. Mesbah, Specification and Development of Database Applications based on Z and SQL, Proc. of Int'l Conference on Information Management and Engineering, pp. 399-405, 2009.
11. H. Leading and J. Souquieres, Integration of UML and B Specification Techniques, Systematic Transformation from OCL Expressions into B, Proc. of 9th Asia-Pacific Software Engineering Conference, 2002.
12. S. K. Kim and D. A. Carrington, An Integrated Framework with UML and Object-Z for Developing a Precise and Understandable Specification: The Light Control Case Study," Proc. of Seventh Asia-Pacific Software Engineering Conference, pp. 240-48, 2000.
13. N. H. Ali, Z. Shukur and S. Idris, A Design of an Assessment System for UML Class Diagram, Int'l Conference on Computational Science and Applications, pp. 539–46. 2007.
14. H. Miao, L. Liu and L. Li, Formalizing UML Models with Object-Z, Proc. of 4th Int'l Conference on Formal Methods and Software Engineering, Springer, 2002.
15. A. M. Mostafa, A. I. Manal, E. B. Hatem and E. M. Saad, Toward a Formalization of UML2.0 Meta-model using Z Specifications, Proc. of 8th ACIS Int'l Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 3, pp. 694-701, 2007.
16. S. Sengupta and S. Bhattacharya, Formalization of UML Diagrams and Consistency Verification: A Z Notation Based Approach, Proc. of India Software Engineering Conference, pp. 151-52, 2008.
17. M. Sarma, D. Kundu and R. Mall, Automatic Test Case Generation from UML Sequence Diagrams, 15th Int'l Conference on Advanced Computing and Communications, pp. 61-65, 2007.
18. N. Yang, H. Yu, H. Sun and Z. Qian, Modeling UML Sequence Diagrams using Extended Petri Nets, Int'l Conference on Information Science & Application, pp. 1-8, 2010.
19. M. A. Ameedeen and B. Bordbar, A Model Driven Approach to Represent Sequence Diagrams as Free Choice Petri Nets, Int'l 12th IEEE Enterprise Distributed Object Computing Conference, pp. 213-21, 2008.
20. N. A. Zafar, Modeling and Formal Specification of Automated Train Control System using Z Notation, IEEE Multi-topic Conference (INMIC'06), pp. 438-43, 2006.
21. N. A. Zafar, S. A. Khan and K. Araki, Towards Safety Properties of Moving Block Railway Interlocking System, Int'l Journal of Innovative Computing, Information & Control, 2012.
22. F. Sohail, F. Zubairi, N. Sabir and N. A. Zafar, Designing Verifiable and Reusable Data Access Layer Using Formal Methods and Design Patterns, Int'l Conference on Computer Modeling and Simulation, 2009.
23. N. A. Zafar, Event-Action Based Model for Identification and Formalization of Relations in

UML State Diagrams, Archives Des Sciences Journal, 65(4), 2012.
24. N. A. Zafar and F. Alhumaidan, Transformation of Class Diagrams into Formal Specification, Int'l Journal Computer Science and Network Security, 289-95, 2011.
25. F. Alhumaidan, A Critical Analysis and Treatment of Important UML Diagrams Enhancing Modeling Power, Intelligent Information Management, 4(5), pp. 231-37, 2012.
26. J. M. Spivey, The Z Notation: A Reference Manual, Englewood Cliffs NJ, Prentice-Hall, 1989.
27. I. Meisels and M. Saaltink, The Z/Eves Reference Manual, TR-97-5493-03d, ORA Canada, 1997.

Nazir A. Zafar was born in 1969 in Pakistan. He received his M.Sc. (Math. 1991), M. Phil (Math. 1993), and M.Sc. (Nucl. Engg. 1994) from Quaid-i-Azam University, Pakistan. He was awarded PhD in computer science from Kyushu University, Japan in 2004.
He is Associate Professor at College of Computer Sciences and Information Technology, King Faisal University, Saudi Arabia. His current research interests are modelling of systems, formal approaches, integration of approaches, safety and security critical systems, etc. He has worked as Dean, Faculty of Information Technology, University of Central Punjab, Pakistan. He has leaded various scientific committees.

Fahad M. Alhumaidan was born in 1966 in Saudi Arabia. He did his PhD from University of Newcastle Upon Tyne, UK. He is working as Assistant Professor and Vice Dean at College of Computer Sciences and IT (CCSIT), King Faisal University, Saudi Arabia. He is also Chairman of Computer Science Department at CCSIT.
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 and Networks.

10/21/2013