Journal of Software Engineering and Applications, **2012, 5, 483-491**
http://dx.doi.org/10.4236/jsea.2012.57056 Published Online July 2012 (http://www.SciRP.org/journal/jsea)

State Based Static and Dynamic Formal Analysis of UML State Diagrams

Fahad Alhumaidan

Department of Computer Science, College of Computer Sciences and Information Technology, King Faisal University, Hofuf, KSA.
Email: falhumaidan@kfu.edu.sa

Received March 28th, 2012; revised April 25th, 2012; accepted May 11th, 2012

ABSTRACT

Design and specification is a serious issue in software engineering because of the semantics involved in transforming the real world problems to computer software systems. Unified Modeling Language (UML) has been accepted as a de facto standard for design and specification of object oriented systems. Unfortunately, UML structures lack defining semantics of a system. Formal methods are proved powerful, particularly, at requirement specification and design level. For a moment, formal methods are not welcomed because of much use of mathematics in formal languages. Therefore, a linkage between UML and formal methods is required to overcome the above deficiencies. In this paper, a new approach is developed by integrating UML and Z specification focusing on state diagram considering both the syntax and semantics. It is believed that this new approach will be effective and useful both at academics and industrial level. The resultant formal models of the approach are analyzed and validated using Z/Eves tool.

Keywords: UML; State Diagram; Formal Methods; Z Notation; Validation and Verification

1. Introduction

In software engineering, requirements analysis and design specification is a challenging task because transformation of real world issues to verifiable computer models has made it a really hard problem. Formal description of system's requirements plays a vital role at initial phases of software engineering. Formal specification is the mathematical description of a system that may be used to construct the consistent system in a systematic and unambiguous way. If we are able to describe formal specification of a system then it can easily be proved and demonstrated the correctness of the required system using computer verification tools. Formal description of a system has obvious advantages over the existing traditional approaches, for example, an incorrect and inconsistent design can be changed and modified before its implementation reducing the software construction cost. Formal methods, based on discrete mathematics such as logic, set theory or graphs, are mathematical techniques used to describe formal description ensuring quality of software systems. But formal methods are not as used at industrial level as their benefits are observed. This is because, for a moment, the software industry people do not have much mathematical background as required in real software engineering.

The Unified Modeling Language (UML) is a standard set of notations and diagrams for specifying, visualizing and constructing artifacts of software systems as well as for business modeling and other non-software systems [1]. UML is a multi-lingual graph based de facto standard used for design and development of object oriented (OO) systems, despite the fact that its semantics is still semi-formal and allows ambiguities in design of a system [2]. Following are few major issues in modeling using UML diagrams being hybrid and visual language:

- UML structures are based on graphical notations and are prone to causing errors.
- The hidden semantics of UML allow ambiguities at the design level of computer software systems.
- The same system needed to be developed can be described by multiple notations or diagrams which may cause inconsistency or ambiguity in design of it.
- UML model may have multiple interpretations that means, the recipient of the design may not find what the author(s) has put in the diagrams.

To overcome the above issues, modeling power of UML can be enhanced by defining semantic rules in a formal way for the diagrams used in design of a system [3]. This is because, unfortunately, much of the UML structures are based on graphical notations having informal or semi-formal definitions which are prone to cause errors [4] as mentioned above. As a result, there is need of forma-

lizing UML diagrams to get full benefit at design level capturing complete functionality of the system to be developed. This integration of formal notations and UML diagrams will result an approach for complete, consistent and correct modeling of a system. Z notation is a formal language, having computer tool support, used to describe and analyze the systems increasing confidence at an abstract level of specification. In this paper, state diagram is transformed to Z specification following syntax and semantics rules for modeling of statics and dynamics of a system. This work is part of our ongoing project on integration UML and formal methods [5]. There exists a few work formalizing UML and formal methods presented in the next section in which mostly it is focused on syntax of the diagrams. In our work, instead of defining only syntactical mapping between UML and Z we have proposed and developed a conceptual model by capturing its semantics hidden under the diagrams. The major objectives of this research are:

- Identifying and proposing an integration of UML and formal approaches to be useful in modeling of complex software systems
- Investigating and providing syntactical and semantics-based relationships between most commonly used UML diagrams and Z notation
- Analyzing and proving correctness of the proposed integration of above approaches
- Developing an approach to provide an automated tool support to transform the UML abstract models to Z specification

For rest of the paper: In Section 2, related work is discussed. Approach used is presented in Section 3. Integration of state diagram and Z notation is given in Section 4. Finally conclusion and future work are discussed in Section 5.

2. Related Work

Although there exits a lot of work [6-10] 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. It is mentioned that only closely related work is discussed in this section. For example, [11] have developed Alloy Constraint Analyzer tool supporting the description of a system whose state space involves relational structures which are complex in nature. By the tool it is possible to analyze and develop a model by investigating the consequences of given constraints by an incremental approach. An approach is demonstrated using XML which is in fact a transformation tool to analyze visualize Timed Communicating Object Z (TCOZ) models into various UML diagrams animating specification with a multi-paradigm programming language as discussed in [12]. It is described a way of creating tables and SQL code for Z specifications according to UML diagrams in [13]. A case study is discussed by a formal verification method for Cooperative Composition Modeling Language (CCML) in [14]. In another work, a relationship is investigated between Petri-nets and Z notation in [15]. An integration of B and UML is presented in [16]. It is investigated the reliability issues using fuzzy logic and petri-nets in [17]. The mathematical induction technique is used to prove correctness of recursive programs in [18]. Formalization of the UML is proposed by focusing on basic constructs of class structures by taking simple case studies in [19]. A tool is developed in [20] which takes UML class diagram in the form of petal files, ASCII format files generated by Rational Rose, and evaluates it automatically and produces a list of comments. Activity model is proposed by ontology based formal method in [21]. A comparison of UML, state-charts, Z, petri nets and fuzzy logic is presented by taking a simple case study on commerce system in [22]. Some other work is listed in [23-25].

3. Tools and Methods

An introduction to approaches used in this research is presented. First merits and demerits of UML are listed. Then introduction to formal methods is given. The reasoning of formalizing UML with Z specification is provided.

3.1. UML

UML has various benefits for modeling of systems. For example, UML is a semi-formal language in which each element of the language is strongly defined [26]. That is you are confident when modeling a particular facet of a system, it will not be misleading. UML is a concise and easy to understand language [27]. The entire language is made up of simple and straightforward concepts and notations. It is comprehensive language and describes all important aspect of a system. Although UML is not a formal language but it has enough expressive power to handle massive and complex systems [28]. It is the result of best practices in modeling of systems using object-oriented concepts and has proved a successful modeling practice. UML has become a de facto standard for modeling of systems using object oriented technology [29].

Despite the above benefits, UML lacks with some important concepts and for a moment cannot be used for the complete design and specification of a system [4]. For example, UML lacks formal semantics. Meanings are hidden under diagrams which create ambiguities and misinterpretations at the implementations level. That is why integration of UML with formal languages is required.

3.2. Formal Approaches

Formal methods are based on mathematical techniques and notations for specification, development and verification of software systems particularly in the area of software engineering [30]. Use of formal methods for software design is motivated by the belief that appropriate mathematical analysis can contribute to the reliability and robustness of software design [31]. In addition, the use of formal methods in the development of high integrity safety or security systems is highly recommended [32]. Formal methods can be used at different levels of modeling and specification [33] as described below.

- At a basic level of applying formal methods, formal specification may be described and then program can be developed in an informal way. This is assumed as most cost-effective option in applications of formal methods for systems development.
- Formal development and verification may be used to produce a program in a formal manner. At this level of applications, proofs of properties from the specification to a program may be conducted. This is considered as most appropriate level of applications in high integrity systems including safety or security systems.
- Theorem proving techniques can be used to conduct proofs which are fully machine checked in a formal manner. Of course this is expensive way but is only applied if the cost of failure is high.

Formal methods may be classified in terms of property oriented and model oriented methods [34]. Property oriented methods are used to describe software in terms of properties, constraints and invariants whereas model oriented methods are used to construct a model of a system [35]. Although there are various tools and techniques available for formal notations but at the current stage of their development in formal methods, it needs an integration of formal techniques and traditional approaches for the complete design and description of a system.

3.3. Z Notation

Z notation is a model oriented specification language based on set theory and first order predicate logic used at an abstract level [36]. In this research, Z is selected to link with UML because of a natural relationship which exists between these approaches. The Z is based upon set theory including standard set operators, comprehensions, Cartesian products and power sets. On the other hand, the logic of Z is formulated using first order predicate calculus. Z allows organizing a system into smaller components known as schemas which are helpful at design level for managing a system. The schema also defines a way in which the state of a system can be described and hence can be used for modeling the dynamics of a system as well. A promising aspect of Z is its stepwise refinement that is verifiable and can be used from an abstraction into an executable code.

3.4. Z/Eves Tools

The Z/Eves is used in this research because it is one of the powerful tools used for the analysis of Z specification [37]. The Z/Eves mathematical toolkit includes the declaration of all the constants of the standard mathematical toolkit and provides useful theorems about these constants. The Z/Eves is used to analyze the system's schema expansion, precondition calculation domain checking, syntax and type checking, and general theorem proving. Any specification written in a formal notation does not mean that it is correct, complete and meaningful. It is user responsibility to make an appropriate use of the tools insuring correctness of the model. The remarkable feature of formal specifications which outclass all other traditional means of informal specification is that a formal specification can be checked and analyzed for the presence of typographical and syntactical errors. The Z/Eves tool provides various exploration techniques to prove the properties of the system.

4. State Based Formal Analysis

In this section, formal analysis of UML state diagram is presented. At first, approach used in this research is discussed. Then dynamic behavior is described based on the static definition of state diagram.

4.1. Proposed Approach

Although formal methods have a well-defined syntax and semantics but these are at the early stage of development and, hence, it needs an integrated tool support for the complete and consistent development of software systems. UML has become a de facto standard for design of object oriented systems. Therefore, it needs to define a relationship between UML diagrams and formal techniques which is analyzed and established in this research.

Some important fundamentals including use cases, class diagrams, sequence diagrams, state machine, will be selected for linking with formal approaches enhancing the modeling power of complex systems. In this way, UML will be extended by applying Z notation syntactically and semantically. A mapping defining relationship between these approaches will be established to be useful for correct and complete modeling of systems. The resultant formal models of the approach will be verified and validated using Z/Eves toolset. The proposed theory will be applied to some real World problem proving its effectiveness. The overall process of formalization and validation is shown in **Figure 1**.

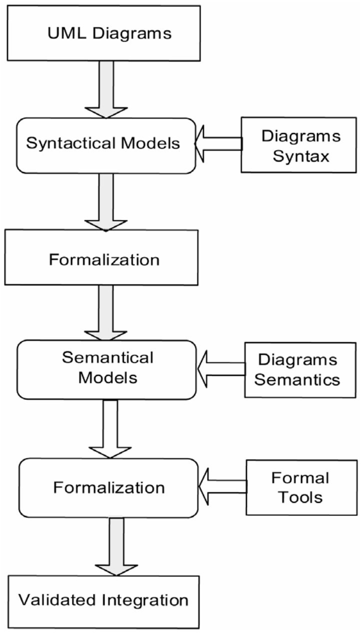

Figure 1. Proposed integration of UML and formal methods.

4.2. Formal Models

A set of definitions used in the formal model is presented in this section. The state identifier and event are represented as *S* and *Event* respectively both as set types. For simple specification, the basic set types are used. In the definition of a transition from one state to another the guard is defined as a Boolean type. A state can have three possible values that are active, passive or null represented as *Active*, *Passive* and *null* respectively. The type of state can be simple, concurrent, non-concurrent, initial or final.

[*S*, *Event*]
Boolean :: *True* | *False*
Status :: *Active* | *Passive* | *Null*
Type :: *Simple* | *Concurent* | *Nonconcurent* | *Initial* | *Final*

In modeling using sets, we do not impose any restriction upon the number of elements and a high level of abstraction is supposed. Further, we do not insist upon any effective procedure for deciding whether an arbitrary element is a member of the given collection or not. As a consequent, our sets S and Event are sets over which we cannot define any operation of set theory. For example, cardinality to know the number of elements in a set cannot be defined. Similarly, the subset, union, intersection or complement operations over the sets are not defined.

The state diagram is a collection of states related by certain types of relations. In the definition of a state, state identifier, its type, status and set of regions is required. Region is defined as a power set of sequence of states. The state is represented by a schema which consists of four components described above. All these components are encapsulated and put in the Schema *State* given below. The invariants over the schema are defined in the second part of schema.

State
name: S
type: Type
status: Status
regions: ďseq *S*Ơ

regions = Ĭ *type* = *Simple*
\# *regions* = 1 *type* = *Nonconcurent*
\# *regions* ≥ 1 *type* = *Concurent*

Invariants:

- In the state diagram, if there is no region in a state then it is a simple state.
- If there is exactly one region in a state then it is termed as non-concurrent composite state.
- If there are two or more regions in a state then it is concurrent composite state.

The collection of states is represented by the schema *States* which consists of four variables. The mapping *substates* from *State* to power set of *State* describes type of a state.

States
start: State
states: *State*
substates: State *State*
target: State

start *states*
start *target*
start dom *substates*
states Ĭ
s: State *s* dom *substates* *s* *states*
s: State *s* *states* *s* *start* *s* *target* *s* . *typ* *Simple* *s* dom *substates*
target *states*
target dom *substates*

Invariants:

- The start state is not in the collection of states.
- The start state is not the target state.
- The start state does not belong to domain of substates mapping that is it has no sub-state.
- The set of states is non-empty.
- For any state, s, if it is in the states and is not the start or target state and not the simple state then it belongs to domain of sub-states.
- The target state does not belong to states.
- The target state of the state diagram does not belong to domain of the sub-states.

To move from one state to another, a transition must be

fired. The transition consists of three components that are event, guard and action. Action is, in fact, a sequence of events in the definition of the transition. The transition is defined by a schema *Transition* in Z notation which consists of three variables which are *event*, *guard* and *action* as given blow.

Transition
event: Event
guard: Boolean
action: seq *Event*

The complete state diagram is represented by the schema *StateDiagram* as given below. The schema consists of set of states of all possible events and the transition function. The transition function takes a state and sequence of events and returns a new state.

StateDiagram
States
events: Event
transitions: State Transition State

s1: State s1 states s1 . type Final
s2: State; tran: Transition (*s2*, *tran*) dom *transitions s1 = s2*
s3: State; tran: Transition (*s3*, *tran*) dom *transitions*
s4: State s4 states s3 = s4
ev: Event ev events
s1, s2: State; tran: Transition
(*s1*, *tran*) dom *transitions*
s2 ran *transitions transitions* (*s1*, *tran*) = *s2*
ev = tran . event ev ran *tran . action*
s3, s4: State; tran: Transition
(*s3*, *tran*) dom *transitions*
s4 ran *transitions transitions* (*s3*, *tran*) = *s4*
ev: Event ev events ev = tran . event ev ran *tran . action*
s5: State; tran: Transition
(*s5*, *tran*) dom *transitions s5 . type Final*
s6: State s6 states transitions (*s5*, *tran*) = *s6*

Invariants:

- For every non final state in the state diagram, there is a transition which can be fired over it.
- For every state over which a transition is fired, it must be in the collection of states of the state diagram.
- For every event in the set of possible events, there must be two states and a transition over these states such that the event is in the transition and it is included in the sequence of events called action which must be executed after the guard condition of the transition is true.
- For any two states s1 and s2, there exists an event e such that transitions (s1, e) = s2.
- For every non-final state there is a transition which acts on it and results a new state, that is, the transition function is defined over every non-final state.

In the state diagram, it is possible that when a transition is fired, it may result the same state. The reflexive relation is satisfied over such states. That means there exists a collection of states in the state diagram over which the reflexive relation is required to be defined. Similarly, it is also possible that when a transition is fired from one state s1 to another state s2 there exists an inverse transition which can be fired from s2 to s1 that is there exists a collection of states over which the symmetric relation is defined. Finally, when a transition is fired from one state s1 to another state s2 and then a new transition is fired from s2 to s3 then a composite transition can be fired from s1 to s3 that is the transitive relation exits over the state diagram.

All of these possible relations are defined by a schema *StatesRelations* given below. The schema consists of four components that are state diagram, reflexive, symmetric and transitive relations. All of these components are put in the first part and invariants are defined over the relations in the second part of the schema for the well-defined-ness.

StatesRelations
StateDiagram
reflexive: State State
symmetric: State State
transitive: State State

s: State s states
(*s*, *s*) *reflexive*
tran: Transition (*s*, *tran*) dom *transitions*
transitions (*s*, *tran*) = *s*
s1, s2: State s1 states s2 states
(*s1*, *s2*) *symmetric*
tran1, tran2: Transition
(*s1*, *tran1*) dom *transitions* (*s2*, *tran2*) dom *transitions transitions* (*s1*, *tran1*) = *s2 tran sitions* (*s2*, *tran2*) = *s1*
s1, s3: State s1 states s3 states
(*s1*, *s3*) *transitive* *s2: State; tran1, tran2: Transition* (*s1*, *tran1*) dom *transitions s2 states*
(*s2*, *tran2*) dom *transitions transitions* (*s1*, *tran1*) = *s2 transitions* (*s2*, *tran2*) = *s3*

Invariants:

- For a state in the collection of states, the relation is

reflexive over it if there exists a transition which results the same state after firing the transition.
- Two states are in the collection of symmetric states if there exists two transitions where each one is an inverse of the other.
- A state is in the collection of transitive states if it is one of the three states over which transitivity is defined to describe the transitive relations.

To identify and define the loops, sub-states and super-states a schema *ComputingStates* is described below. The schema consists of five components namely state diagram, input state, loops, sub-states and super-states. The state diagram is given as input in the schema for all the functions computing loops, sub-states and super-states of a given state. The loops variable returns all the states over which transition function returns the same state. The *subs*! variable computes all the possible sub-states, if exist, of a given state. And similarly the *sups*! variable evaluates all the possible super-states, if exist, of a given state.

ComputingStates
StateDiagram
state?: State
loops!: State
subs!, sups!: State

state? states
loops!
= *s: State; tran: Transition*
(s, tran) dom transitions transitions (s, tran) = s s
subs! = s: State; tran: Transition
(state?, tran) dom transitions transitions (state?, tran) = s s

Invariants:
- The input state must belong to collection of all the states of the state diagram.
- The loops are computed by taking a state and the transitions which return the same state.
- The sub-states of a given state are calculated by using the recursive definition of the transition function.
- The super-states of a given state are calculated by using the recursive definition of the inverse of the sequence of transition functions by changing the source and target states with each other.

In this section, a sequence of possible transitions and events of the state diagram is described when moving from one state to another of the diagram. The specification is given using the schema *Protocol* which consists of four components named as state diagram, sequence of transitions, start state and target state. From start state the protocol is initiated, at end state it is ended. And the sequence of transitions is used to move from start to end state. The algorithm of moving from start to final state is given below in addition to few invariants required.

Protocol
StateDiagram
protocol: seq *Transition*
start: State
target: State

start states target states 1 # protocol
s: State; tran: Transition; evts: seq *Event*
(start, tran) dom transitions
transitions (start, tran) = s tran . guard = True
start . status = Active s . status = Passive
1 # evts tran . event = evts 1
i: i 1 .. # evts - 1 i + 1 # evts
(tran . action, i) applies$to
evts (i + 1) = tran . action i
protocol 1 = tran
i: i dom protocol 2 i i # protocol - 1
s1, s2: State; tran: Transition; evts: seq *Event*
(s1, tran) dom transitions transitions (s1, tran) = s2
tran . guard = True 1 # evts tran . event = evts 1
i: i 1 .. # evts - 1 i + 1 # evts
(tran . action, i) applies$to
evts (i + 1) = tran . action i
protocol i = tran
s: State; tran: Transition; evts: seq *Event*
(s, tran) dom transitions transitions (s, tran) = target
tran . guard = True s . status = Active
target . status = Passive 1 # evts
tran . event = evts 1 i: i 1 .. # evts - 1 i + 1 # evts
(tran . action, i) applies$to
evts (i+1)= tran .action i
protocol (# protocol) = tran

Invariants:
- The start state is in collection of states of the diagram.
- The target state belongs to the collection of the states.
- There exists a transition which takes start state and a sequence of events and returns the next possible state.
- There exists a sequence of transition(s) possibly single transition which takes state next to start state and traverses to reach the state previous to the target state.
- There exists a transition which takes the state previous to target and returns the target state.

As there may exist many ways of reaching from start to target state depending upon the input of events. It means, it needs to describe all the possible ways in moving from start to target state. For this purpose a new

schema *Protocols* of type power set of *Protocol* is defined. The schema describes the possible ways by using a recursive definition of the *Protocol* schema as given below.

Protocols
protocols: Protocol

pr: Protocol pr protocols pr . protocol
action: Event; event: Event;
guard: Boolean pr . start pr . states pr . target pr . states 1 # pr . protocol
pr: Protocol pr protocols pr . protocol
action: Event; event: Event;
guard: Boolean s: State; tran: Transition; evts: seq *Event*
pr . start tran dom *pr . transitions*
pr . transitions pr . start tran applies$to
pr . transitions pr . start tran = s
tran . guard = True pr . start . status = Active
s . status = Passive 1 # evts tran . event = evts 1
i: i 1 .. # evts - 1 i + 1 # evts
tran . action i applies$to
evts i + 1 = tran . action i pr . protocol 1 = *tran*
pr: Protocol pr protocols pr . protocol
action: Event; event: Event; guard: Boolean
i: i dom *pr . protocol 2 i i # pr . protocol* - 1 *s1,s2:State;tran:Transition; evts:* seq *Event*
s1 tran dom *pr . transitions*
pr . transitions s1 tran = s2 tran . guard = True
1 *# evts tran . event = evts* 1
i: i 1 .. # evts - 1 i + 1 # evts
tran . action i applies$to
evts i + 1 = tran . action i pr . protocol i = tran

Invariants:

- For a protocol, there exists a transition and sequence of events which take start to the next possible state.
- There exists a sequence of transition(s) and sequence of sequence of events which traverses all the possible states except the start and target states.
- For each protocol, there exists a transition and sequence of events which reaches to the target state based on the above recursive definition of protocol.

5. Conclusion and Future Work

Unified Modeling Language (UML) is used at initial phases because of having much support of diagrams whereas formal methods are useful at the later stages of software development because of having rigorous mathematical and computer tools support. In this way, UML and Formal Methods are both useful for design and specification of software systems therefore integration of these approaches was required for a systematic development. To facilitate the software development process an automatic generation of specification from diagrams will be much useful to capture the hidden semantics under the UML diagrams. In this research, UML state diagram is linked and formalized using Z achieving above objective. The most relevant work [38-40] was considered as starting point for this research. In addition to above following benefits are observed: 1) An approach is developed by linking UML to Z notation by defining a relationship among fundamentals of these semi-formal and formal techniques; 2) The reusability issue is addressed by defining the components and developing recursive approach to be useful easing the development process; 3) The resultant approach will be useful in the systems development and construction of automated tools for generating specification from the UML state diagram; 4) Mostly, students use UML for designing the projects, this new approach will facilitate them to improve the design at initial stages of their project development.

The research work on formalization of some other important diagrams, using Z notation, of UML including use case and sequence diagrams is in process and will appear soon.

6. Acknowledgements

I would like to acknowledge Software Engineering Research Group at the Department of Computer Science, King Faisal University, for their valuable comments.

REFERENCES

[1] B. Akbarpour, S. Tahar and A. Dekdouk, "Formalization of Cadence SPW Fixed-Point Arithmetic in HOL," *Formal Methods in System Design*, Vol. 27, 2005, pp. 173-200. doi:10.1007/s10703-005-2256-8

[2] N. H. Ali, Z. Shukur and S. Idris, "A Design of an Assessment System for UML Class Diagram," *International Conference on Computational Science and Applications*, Kuala Lumpur, 26-29 August 2007, pp. 539-546.

[3] K. Araki, A. Galloway and K. Taguchi, "Using a Process Algebra to Control B Operations," *Proceedings of 1st International Conference on Integrated Formal Methods*, Springer-Verlag, London, 1999, pp. 437-456.

[4] H. Beek, A. Fantechi, S. Gnesi and F. Mazzanti, "State/ Event-Based Software Model Checking," *Proceedings of 4th International Conference on Integrated Formal Methods*, Springer, Canterbury, Vol. 2999, 2004, pp. 128-147.

[5] M. Bo, W. Huang and J. Qin, "Automatic Verification of Security Properties of Remote Internet Voting Protocol in Symbolic Model," *Information Technology Journal*, Vol.

9, No. 8, 2010, pp. 1521-1556. doi:10.3923/itj.2010.1521.1556

[6] R. Borges and A. Mota, "Integrating UML and Formal Methods," *Electronic Notes in Theoretical Computer Science*, Vol. 84, 2003, pp. 97-112.

[7] M. Brendan and J. S. Dong, "Blending Object-Z and Timed CSP: An Introduction to TCOZ," *Proceedings of International Conference on Software Engineering*, Kyoto, 19-25 April 1998, pp. 95-104.

[8] W. S. Changchien, J. J. Shen and T. Y. Lin, "A Preliminary Correctness Evaluation Model of Object-Oriented Software Based on UML," *Journal of Applied Sciences*, Vol. 2, No. 3, 2002, pp. 356-365. doi:10.3923/jas.2002.356.365

[9] A. Dennis, B. H. Wixom and D. Tegarden, "Systems Analysis and Design with UML," 3rd Edition, Wiley, New York, 2005, 576 p.

[10] Z. Derakhshandeh, B. T. Ladani and N. Nematbakhsh, "Modeling and Combining Access Control Policies Using Constrained Policy Graph," *Journal of Applied Sciences*, Vol. 8, 2008, pp. 3561-3571. doi:10.3923/jas.2008.3561.3571

[11] J. Derrick and G. Smith, "Structural Refinement of Object-Z/CSP Specification," *Proceedings of 2nd International Conference on Integrated Formal Methods*, Springer-Verlag, London, Vol. 1945, 2000, pp. 194-213.

[12] S. A. Ehikioya and B. Ola, "A Comparison of Formalisms for Electronic Commerce Systems," *Proceedings of International Conference on Computational Cybernetics*, Vienna, 30 August-1 September 2004, pp. 253-258.

[13] F. Gervais, M. Frappier and R. Laleau, "Synthesizing B Specifications from EB3 Attribute Definitions," *Proceedings of 5th International Conference on Integrated Formal Methods*, Springer, Berlin, Vol. 3771, 2005, pp. 207-226.

[14] A. Hall, "Correctness by Construction: Integrating Formality into a Commercial Development Process," *Proceedings of International Symposium of Formal Methods Europe*, Vol. 2391, 2002, pp. 139-157.

[15] K. E. Hamdy, M. A. Elsoud and A. M. El-Halawany, "UML-Based Web Engineering Framework for Modeling Web Application," *Journal of Software Engineering*, Vol. 5, No. 2, 2011, pp. 49-63. doi:10.3923/jse.2011.49.63

[16] O. Hasan and S. Tahar, "Verification of Probabilistic Properties in the HOL Theorem Prover," *Proceedings of the Integrated Formal Methods*, Vol. 4591, 2007, pp 333-352. doi:10.1007/978-3-540-73210-5_18

[17] X. He, "Formalizing UML Class Diagrams: A Hierarchical Predicate Transition Net Approach," *Proceedings of 24th Annual International Computer Software and Applications Conference*, Taipei, 25-28 October 2000, pp. 217-222.

[18] M. Heiner and M. Heisel, "Modeling Safety Critical Systems with Z and Petri-Nets," *Proceedings of International Conference on Computer Safety, Reliability and Security*, Springer-Verlag, London, 26-28 October 1999, pp. 361-374. doi:10.1007/3-540-48249-0_31

[19] D. Jackson, I. Schechter and I. Shlyakhter, "Alcoa: The Alloy Constraint Analyzer," *Proceedings of International Conference on Software Engineering*, Limerick, 4-11 June 2000, pp. 730-733.

[20] H. Leading and J. Souquieres, "Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B," *Proceedings of 9th Asia-Pacific Software Engineering Conference*, Gold Coast, 4-6 December 2002, p. 495. doi:10.1109/APSEC.2002.1183053

[21] Liu and C. Chen, "An Improved Quasi-Static Scheduling Algorithm for Mixed Data-Control Embedded Software," *Journal of Applied Sciences*, Vol. 6, No. 7, 2006, pp. 1571-1575. doi:10.3923/jas.2006.1571.1575

[22] Z. M. Ma, "Fuzzy Conceptual Information Modeling in UML Data Model," *International Symposium on Computer Science and Computational Technology*, Shanghai, 20-22 December 2008, pp. 331-334. doi:10.1109/ISCSCT.2008.353

[23] R. Miles and K. Hamilton, "Learning UML 2.0," 1st Edition, O'Reilly Media, Sebastopol, 2006, 288 p.

[24] A. Moeini and R. O. Mesbah, "Specification and Development of Database Applications Based on Z and SQL," *Proceedings of International Conference on Information Management and Engineering*, Kuala Lumpur, 3-5 April 2009, pp. 399-405.

[25] A. M. Mostafa, A. I. Manal, E. B. Hatem and E. M. Saad, "Toward a Formalization of UML2.0 Meta-Model Using Z Specifications," *Proceedings of 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Qingdao, Vol. 3, 2007, pp. 694-701. doi:10.1109/SNPD.2007.508

[26] H. Podeswa, "UML for IT Business Analyst," 2nd Edition, *Course Technology*, 2009, 372 p.

[27] T. B. Raymond, "Integrating Formal Methods by Unifying Abstractions," Springer, Berlin, Vol. 2999, 2004, pp. 441-460.

[28] S. Sengupta and S. Bhattacharya, "Formalization of UML Diagrams and Consistency Verification: A Z Notation Based Approach," *Proceedings of India Software Engineering Conference*, Hyderabad, 19-22 February 2008, pp. 151-152. doi:10.1145/1342211.1342248

[29] Z. Shi, "Intelligent Target Fusion Recognition Based on Fuzzy Petri Nets," *Information Technology Journal*, Vol. 11, No. 4, 2012, pp. 500-503. doi:10.3923/itj.2012.500.503

[30] M. Shroff and R. B. France, "Towards Formalization of UML Class Structures in Z," 21*st International Conference on Computer Software and Applications*, Washington, 11-15 August 1997, pp. 646-651.

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

[32] J. Sun, J. S. Dong, J. Liu and H. Wang, "A XML/XSL Approach to Visualize and Animate TCOZ," *Proceedings of 8th Asia-Pacific Software Engineering Conference*, Macao, 4-7 December 2001, pp. 453-460.

[33] X. Than, H. Miao and L. Liu, "Formalizing Semantics of UML Statecharts with Z," *Proceedings of 4th Inter-*

national Conference on Computer & Information Technology, Wuhan, 14-16 September 2004, pp. 1116-1121.

[34] J. M. Wing, "A Specifier, Introduction to Formal Methods," *Computer Journal*, Vol. 23, No. 9, 1990, pp. 8-24. doi:10.1109/2.58215

[35] Z. Xiuguo and H. Liu, "Formal Verification for CCML Based Web Service Composition," *Information Technology Journal*, Vol. 10, No. 9, 2011, pp. 1692-1700.

[36] C. Yong, "Application of Wu's Method to Proving Total Correctness of Recursive Program," *Information Technology Journal*, Vol. 9, No. 7, 2010, pp. 1431-1439. doi:10.3923/itj.2010.1431.1439

[37] N. A. Zafar and F. Alhumaidan, "Transformation of Class Diagrams into Formal Specification," *International Journal Computer Science and Network Security*, Vol. 11, No. 5, 2011, pp. 289-295.

[38] S. Zarina, N. Alias, M. M. Halip and B. Idrus, "Formal Specification and Validation of Selective Acknowledgement Protocol using Z/EVES Theorem Prover," *Journal of Applied Sciences*, Vol. 6, No. 8, 2006, pp. 1712-1719. doi:10.3923/jas.2006.1712.1719

[39] Z. X. Wu, H. He, L. Chen and Y. Zhang, "Ontology Based Semantics Checking for UML Activity Model," *Information Technology Journal*, Vol. 11, No. 3, 2012, pp. 301-306. doi:10.3923/itj.2012.301.306

[40] N. A. Zafar, "Event-Action Based Model for Identification and Formalization of Relations in UML State Diagrams," *Archive De Science Journal*, Vol. 65, No. 4, 2012, pp. 17-30.

