Transformation of Class Diagrams into Formal Specification

Nazir Ahmad Zafar and Fahad Alhumaidan
College of Computer Sciences and Information Technology
King Faisal University, Hofuf
SAUDI ARABIA

Abstract:
Requirements analysis and design specification is a serious issue in software engineering because of semantics involved in the transformation of real world problems to computational models. Unified Modeling Language (UML) has been accepted as a standard for design and development of object oriented systems. Unfortunately, most of UML structures are based on graphical notations and are prone to causing errors. That means UML has a lack of notations for description of a complete functional system and its semantics is still semi-formal allowing ambiguities at design level. Formal methods have played a vital role, particularly, at requirement analysis and design level but, for a moment, are not fully welcomed because of involving much mathematics. Therefore, a concrete linkage of UML and formal methods is needed to overcome the above issues. In this paper, an integration of UML and Z notation is defined for class diagrams considering both the syntax and semantics at an abstract level of specification. Four major kinds of relationships, i.e., association, generalization, aggregation and composition are addressed. The formal specification is analyzed and validated using Z/Eves tool.

Keywords:
UML, Class diagrams, Formal methods, Z notation, Integration, Validation.

1. Introduction

In software engineering, requirements capturing, analysis and design specification is a serious problem which is very natural because transformation of real world problems to mathematical computational models is not an easy task. Specification of software systems has a vital role in the area of software engineering. Formal specification of a system is its mathematical description that may be used to develop a system in a consistent and systematic way. Formal specification describes about the system what it does but it does not show how it does. If we have formal specification of the system, it is easy to prove and demonstrate correctness of it using formal verification tools and techniques. The formal specification has an obvious advantage over traditional approaches that incorrect design of a system can be revised before its implementation. Formal methods are mathematical techniques based on discrete mathematics which can be used to describe formal specification of a system. On the other hand, formal methods are not much useful for describing design of a system because of having mathematical notations. Diagrams and graphical based approaches are very helpful in capturing requirements and presenting design of a system. The design of a complex system, not only requires definition of a system but it also needs to model its behavior and other dynamics.

Unified Modeling Language (UML) has been widely used in software industry and accepted as the standard notation for design and development of object oriented (OO) systems despite the fact that its semantics is still semi-formal and allows ambiguities [28], [24]. UML is a proposed common OO modeling language which is more useful if it has a maximum base to define semantic of the system in a formal way [21]. Unfortunately, much of the UML structures are based on graphical notations, having informal and semi-formal definitions, very prone to causing errors and can not be analyzed formally [2]. As a result, there is need of formalizing UML structures and diagrams to get full benefit of it at design level capturing the maximum functionality of the system to be developed. Z notation is an abstract formal specification language used to describe and analyze the systems increasing confidence. In this paper, we present the preliminary results of our research on formalization of UML diagrams. We show how the semantics of UML can be captured to define the general principles and concepts.

There exists a few work linking UML and formal methods presented in the next section. In this paper, class diagrams are selected to formalize using Z notation. Instead of defining only syntactical mapping between class diagram and Z we have proposed the conceptual model by capturing its semantics hidden under the diagrams. At first we have formalized the class by taking its name, attributes and operations at an abstract level of specification. Then the same class was reused creating subclasses, whole and part classes by a powerful concept of substation in Z notation. In defining the mapping from class diagrams to Z four major types of relationships namely, association, generalization, aggregation and composition are considered. After formalizing the classes and relations mentioned above, it were synthesized to define the class diagrams using Z schema structure. We mapped the syntactical as well as semantical mapping among both approaches. The major objectives of this research are: (1) identifying, proposing and proving an integration of UML and formal approaches to be useful in modeling of systems, (2) providing a syntactic and semantic relationship

Manuscript received May 5, 2011
Manuscript revised May 20, 2011

between UML class diagrams and Z and (3) providing an automated tool support to transform UML model to Z. Rest of the paper is organized as follows.
In section 2, related work is discussed. An introduction to formal methods is given in section 3. Formal relationship of class diagrams and Z notation is described in section 4. Finally, concluding remarks are presented in section 5.

2. Related Work

An integration of approaches is an important research area because of an introduction of new technologies and development of automated computer tools. Although there exits a lot of work [4], [8], [9], [12], [16], [23], [27] on integration of approaches but there does not exists much work on linking UML with formal techniques because of the hidden semantics under the UML diagrams. Some of the closely related work is listed in this section. For example, Jackson et al. [12], have developed Alloy Constraint Analyzer (Alcoa) tool which supports the description of systems whose state involves complex relational structure. The tool makes it possible to develop and analyze a model incrementally by investigating the consequences of given constraints. An approach is demonstrated using XML/XSL as a transformation tool to visualize TCOZ models into various UML diagrams to animate specifications with a multi-paradigm programming language in [15]. In [3], Shahreza et al. have described a way of creating tables and SQL code for Z specifications according to UML diagrams. A relationship is investigated between Petri-nets and Z in [19]. An integration of B and UML is presented in [10], [11]. In [21], formalization of the UML is proposed by focusing on basic constructs of class structures by taking simple case studies. A tool is developed in [22] which takes UML class diagram in the form of Rational Rose petal files and evaluates it automatically and produces a list of comments on the diagram. A comparison of fuzzy logic, Z, UML, statecharts, petri nets, and finite state machines is carried out by taking a simple case study on commerce system in [25]. In [29], fuzzy theory is introduced at certain levels of class diagrams of UML supporting the fuzzy modeling of computerized systems.

3. Formal Methods

Formal methods are approaches based on mathematical techniques used for describing and analyzing properties of software systems. Formal methods may be classified in terms of property oriented and model oriented methods [18]. 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 [13]. Although there are various tools and techniques available for formal notations but at the current stage of their development, it needs an integration of formal techniques and traditional approaches for the complete design and description of a system.
Z notation is a model oriented specification language based on set theory and first order predicate logic used at an abstract level [14]. In this paper, Z is selected to be linked 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. The Z is used in our research because it allows organizing a system into its smaller components known as schemas which are very helpful at design level for managing the 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.

4. Formal Specification of Class Diagrams

Although formal methods have well-defined syntax and semantics but these are at the early stage of development and hence need an integrated tool support for the complete and consistent development of software systems. We are working on a project for integration of UML and Z notation some preliminary results of it are presented in this paper. Initially, we have decided to transform semantically class diagram to Z notation. UML class diagrams are generally used to catch the static aspects of a system. The class diagrams are selected for formalization because of their common use in software development. Moreover, there is a great similarity between Z notation and UML class diagrams. For example, class diagrams are equipped with variables and operations. Further, the pre and post conditions can be defined as well in addition to class invariants but that is all about less formal and more graphical based. UML class diagrams can be used to characterize both the static and dynamic aspects of a system. On the other hand, Z has an important abstract data type called schema which is used to define the variables and constraints over it. The operations can also be defined in terms of schemas to capture the behavioral aspects of a system. Therefore a strong relationship exists between UML class diagrams and Z notation which is analyzed at an abstract level of specification.
We start with the definitions of variables used for defining the class diagram. A class consists of three parts, i.e., name, set of attributes and operations. For name variable an

identifier is used denoted by *ID*. Since attributes needs a type which is denoted by *Type* in Z notation. Finally, operation needs a set of attributes as input and produces a new set of attributes as output. For this purpose a *Composite* variable is used which is of type of finite power set of *Type* denoted by $\mathbb{P}$ *Type*.

[*ID*]; [*Type*]; *Composite* $\leftrightarrow$ $\mathbb{P}$ *Type*

Class in UML can be used to create a set of objects which share attributes, operations, relationships and other semantics of the class. The schema in Z notation can be used to create the objects which can share all the operations and may contain the attributes. Therefore class in UML is defined using the schema structure in Z and is denoted by *Class*. The schema consists of two parts, the first one is used to define variables, the second is for invariants and other constrains.

In the first part of schema, class name in UML is defined by identifier (ID) in Z. Only three type of attributes are considered, i.e., public, private and protected of type power set and denoted by $\mathbb{P}$ *Type*. The other type is a collection of operations which is in fact a relationship between a set of attributes to another set of attributes having the type *Composite* $\leftrightarrow$ *Composite*, where composite is collection of attributes of the class. The formal description is given below along with invariants over the variables defined in the second part of the schema.

Class
id: ID
attributes, public, private, protected: $\mathbb{P}$ *Type*
operations: Composite $\leftrightarrow$ *Composite*
---
attributes = public $\cup$ *private* $\cup$ *protected*
public $\cap$ *private* = $\emptyset$
public $\cap$ *protected* = $\emptyset$
private $\cap$ *protected* = $\emptyset$
c1, c2: Composite $\mid$ $c1 \mapsto c2^{+} \in$ *operations* $\bullet$ *c1* $\in$ *attributes* $\bullet$ *c2* $\in$ *attributes*

Invariants: (i) The set of all attributes is equal to public, private and protected attributes. (ii) The intersection of public and private attribute is empty, i.e., an attribute can be either private or protected but can not have the properties of both. (iii) The public and protected attributes are disjoint. (iv) The private and protected attributes are also disjoint. (v) Any of the two attributes which are given as input to any of the operation must be in the domain of operation relation.

4.1 Association

Association relationship is used by UML class diagram in order to capture the relations among the objects of the classes. It is a relationship in which it is specified about the objects how these are connected to other objects. Mathematically, a relation is a link relating a set X to set Y having information needed to be related. The elements of set X is called domain and the element of set Y is called range of the relationship. There are four kinds of association relationships, i.e., many to many, many to one, one to many and one to one. For example, think of class-teacher relationship. As one teacher can teach many classes and one class can be taught by many teachers, hence, we can describe this relation as many to many. Take another example of student-supervisor relationship. If we suppose that one student can be supervised by only one supervisor but on the other hand one supervisor can supervise many student it will be a many to one relationship. If we change the student-supervisor to supervisor-student relationship it will become one to many. If we take an example of a society where a man can marry to one women and vice-versa, it will be the case of one to one relationship. The association relationship is denoted by *Association* and is described below. The schema consists of six components, i.e., *association*, *mtom*, *onetoone*, *mtoone*, *onetom* and *classes*. The first one *association* represents to set of all possible associations, the second one *mtom* represents to many to many relationships, *onetoone* is used for one to one, *mtoone* shows to many to one, *onetom* describes one to many relationships and the last one variable *classes* represents to set of all classes over which these relations are defined. All of these components are put in the first part of the schema and invariants defining their description in addition to all possible relationship among these components are presented in the second part of it.

Association
association: Class $\leftrightarrow$ *Class*
mtom: Class $\leftrightarrow$ *Class*
onetoone: Class $\leftrightarrow$ *Class*
mtoone: Class $\leftrightarrow$ *Class*
onetom: Class $\leftrightarrow$ *Class*
classes: $\mathbb{P}$ *Class*
---
association = mtom $\cup$ *onetoone* $\cup$ *mtoone* $\cup$ *onetom*
mtom $\cap$ *onetoone* = $\emptyset$ $\bullet$ *mtom* $\cap$ *mtoone* = $\emptyset$
mtom $\cap$ *onetom* = $\emptyset$ $\bullet$ *onetoone* $\cap$ *mtoone* = $\emptyset$
onetoone $\cap$ *onetom* = $\emptyset$ $\bullet$ *mtoone* $\cap$ *onetom* = $\emptyset$
c1, c2: Class $\mid$ $c1 \mapsto c2^{+} \in$ *mtom*
$\mid$ $c1 \mapsto c2^{+} \in$ *onetom* $\mid$ *c3, c4, c: Class* $\mid$ $c3 \mapsto c^{+} \in$ *mtom* $\mid$ $c4 \mapsto c^{+} \in$ *mtom* $\bullet$ *c3 = c4*
c1, c2: Class $\mid$ $c1 \mapsto c2^{+} \in$ *mtom* $\mid$ $c1 \mapsto c2^{+} \in$ *mtoone* $\mid$ *c, c3, c4: Class* $\mid$ $c \mapsto c3^{+} \in$ *mtom* $\mid$ $c \mapsto c4^{+} \in$ *mtom* $\bullet$ *c3 = c4*

□□*c1, c2: Class* □ ⌣*c1*□ *c2*⌢ □ *mtom*
□ □ ⌣*c1*□ *c2*⌢ □ *onetoone* □ ⌣□*c1, c2, c: Class* □ ⌣*c1*□ *c*⌢ □ *mtom* □□ ⌣*c2*□ *c*⌢ □ *mtom* □ *c1 = c2*⌢
□□ ⌣□*c,c1,c2:Class* □ ⌣*c*□ *c1*⌢ □ *mtom* □ ⌣*c*□ *c2*⌢ □ *mtom* □ *c1 = c2*⌢
□dom *association* □ *classes* □ ran *association* □ *classes*
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Invariants: (i) The association relationship is union of many to many, one to one, many to one and one to many relationships. (ii) The intersection of any two types of relationships is empty. (iii) Many to many relationship will be one to many if any element in the range can not be repeated. (iv) Many to many relationship will be many to one if any element in the domain is not repeated. (v) Many to many relationship will be one to one if neither an element in the domain nor range is repeated. (vi) Domain and range of association relation must be in the set of classes.

4.2 Generalization

In generalization relationship one object (child) is based on another object (parent) in the UML class diagram. In the relationship, the child receives all the attributes, operations and relationships that are defined in the parent. The objects involved in the generalization relationship are of same type for complying with the semantics of UML. The generalization relationships are modeled to capture the attributes, operations and relationships introduced in parent classes and are reused in any number of child classes. In this way only additional attributes, operations and relationships are defined in the child class and all components of parent classes are reused. In the generalization, the parent class can have more than one children and any child class can have more than one parents. Before defining the generalization relationship, child class denoted by *SubClass* is introduced based on the class by a powerful technique that is substitution of Z notation. The relationship is described below in terms of schema consisting of seven components. In the schema the generalization, simple and multiple relationships and other components needed for its description are encapsulated.

SubClass □
Class[*ids*/*id*, *attributess*/*attributes*, *publics*/*public*, *privates*/*private*, *protecteds*/*protected*, *operationss*/*operations*]

□□*Generalization* □□□□□□□□□□□□□□□□□□□□□□□□
□*Class*
□*SubClass*
□*classes:* □ *Class*
□*subclasses:* □ *SubClass*
□*generalization: SubClass* □ *Class*
□*simple: SubClass* □ *Class*
□*multiple: SubClass* □ *Class*
□□□□□□□□□□□□□□□□□□
□*id* □ *ids*
□*attributes* □ *attributess*
□*public* □ *publics*
□*private* ⋃ *privates* = ⌣⌣
□*protected* □ *protecteds*
□dom *generalization* □ *subclasses*
□ran *generalization* □ *classes*
□*generalization = simple* ⋃ *multiple*
□□*s: SubClass; c1, c2: Class* □ ⌣*s*□ *c1*⌢ □ *simple* □ ⌣*s*□ *c2*⌢ □ □*simple* □ *c1 = c2*
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Invariants: (i) The identifiers of child and parent classes must be different. (ii) Attributes of parent class are included in the child class. (iii) The public attributes of parent class are visible to child class. (iv) Private attributes of child and parent class are disjoint. (v) Protected attributes of parent class are included in the child class. (vi) Domain of generalization relationship is in the set of child classes. (vii) Range of generalization relationship is in the set of parent classes. (viii) Generalization relationship is union of simple and multiple relationships. (ix) The relationship is simple if a child class can not have more than one parent.

4.3 Aggregation

In association relationship various types of objects are related to link, study and analyze their characteristics whereas in case of aggregation relationship objects are assembled together to create a more complex object. An aggregation describes a group of objects and their way of interaction with each other, in this way aggregation relationship is a special type of association. For example, a department can have an aggregation relationship with a college showing that the department is a part of college. To define an aggregation relationship in Z, two types of classes, i.e., whole and part are created based on class and are denoted by *WholeClass* and *PartClass* respectively. The aggregation relationship is defined by *Aggregation* schema which consists of *aggregation*, *whole* and *parts*.

WholeClass □
Class[*id*/*id*, *attributes*/*attributes*, *public*/*public*, *private*/*private*, *protected*/*protected*, *operations*/*operations*]

PartClass □
Class[*id*/*id*, *attributes*/*attributes*, *public*/*public*, *private*/*private*, *protected*/*protected*, *operations*/*operations*]

□□*Aggregation*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□*aggregation: WholeClass* □ *PartClass*
□*whole:* □ *WholeClass*
□*parts:* □ *PartClass*
□□□□□□□□□□□□□□□□□
□dom *aggregation* □ *whole*
□ran *aggregation* □ *parts*
□□*c1, c2 : WholeClass; c: PartClass* □ ◡*c1*□ *c*⌢ □ *aggregation* □ ◡*c2*□ *c*⌢ □ *aggregation* □ *c1 = c2*
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Invariants: (i) The domain of aggregation relationship is a subset of set of whole classes. (ii) The range of aggregation relationship is a subset of set of parts classes. (iii) A part class can not be a part of two different whole classes.

4.4 Composition

In UML, composition relationship is a special type of aggregation. The relationship in composition is stronger than aggregation. In aggregation if a whole is destroyed the part may exist whereas in case of composition this is not the case, i.e., part class cannot exist without the whole class. For example, if we were going to model a car, we know wheels are part of the car. The lifetime of the wheels is managed by the car. That is when car is destroyed the wheels will be destroyed. Hence this relationship must be modeled by the composition. The composition relationship is defined by *Composition* schema consisting of three components, i.e., *composition*, *whole* and *parts* defined in the first part of the schema. The invariants are defined in the second part of the schema: The domain of composition relationship is equal to the whole set. (ii) The range of composition relationship is equal to set of parts set. (iii) A part class can not be a part of two different whole classes as was the case of aggregation.

□□*Composition*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□*composition: WholeClass* □ *PartClass*
□*whole:* □ *WholeClass*
□*parts:* □ *PartClass*
□□□□□□□□□□□□□□□□□
□dom *composition = whole*
□ran *composition = parts*
□□*c1, c2 : WholeClass; c: PartClass* □ ◡*c1*□ *c*⌢ □ *composition* □ ◡*c2*□ *c*⌢ □ *composition* □ *c1 = c2*
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.5 Class Diagram

As we know class diagrams show all the possible classes of the system, their interaction in terms of relationships, operations and the attributes of the classes. Class diagrams are used from domain modeling to detailed design of the system. In the schema *ClassDiagram* given below, an abstract view of the complete class diagram is presented by defining all of its components which are needed for its description. The schema includes all general types of classes that are subclasses, parts and whole classes. Only four major kinds of relationships are considered here, i.e., association, generalization, aggregation and composition. All of these relationships are described above exclusively and put in the schema defining class diagram. The well defined-ness of the relationships among the classes is checked in terms of properties in the second part of the schema. The way of description of class diagram will facilitate the transformation of syntax and semantics of UML to Z specification.

□□*ClassDiagram* □□□□□□□□□□□□□□□□□□□□□□□□
□*ass: Association*
□*gen: Generalization*
□*agg: Aggregation*
□*comp: Composition*
□*classes:* □ *Class*
□*subclasses:* □ *SubClass*
□*parts:* □ *PartClass*
□*whole:* □ *WholeClass*
□*association: Class* □ *Class*
□*generalization: SubClass* □ *Class*
□*aggregation: WholeClass* □ *PartClass*
□*composition: WholeClass* □ *PartClass*
□□□□□□□□□□□□□□□□□
□*classes = ass . classes* ⋃ *gen . classes*
□*subclasses = gen . subclasses*
□*whole = agg . whole* ⋃ *comp . whole*
□□*c1, c2: Class* □ ◡*c1*□ *c2*⌢ □ *ass . association* □ ◡*c1*□ *c2*⌢ □ □*association*
□□*c1, c2: Class* □ ◡*c1*□ *c2*⌢ □ *association* □ ◡*c1*□ *c2*⌢ □ *ass . association*
□□*c3: SubClass; c4: Class* □ ◡*c3*□ *c4*⌢ □ *gen . generalization*
□ □ ◡*c3*□ *c4*⌢ □ *generalization*
□□*c3: SubClass; c4: Class* □ ◡*c3*□ *c4*⌢ □ *generalization*
□ □ ◡*c3*□ *c4*⌢ □ *gen . generalization*
□□*c5: WholeClass; c6: PartClass* □ ◡*c5*□ *c6*⌢ □ *agg . aggregation*
□ □ ◡*c5*□ *c6*⌢ □ *aggregation*
□□*c5: WholeClass; c6: PartClass* □ ◡*c5*□ *c6*⌢ □ *aggregation*
□ □ ◡*c5*□ *c6*⌢ □ *agg . aggregation*
□□*c7: WholeClass; c8: PartClass* □ ◡*c7*□ *c8*⌢ □ *comp . composition* □ ◡*c7*□ *c8*⌢ □ *composition*
□□*c7: WholeClass; c8: PartClass* □ ◡*c7*□ *c8*⌢ □ *composition*
□ □ ◡*c7*□ *c8*⌢ □ *comp . composition*
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

In the predicate part, it is described that: (i) The set of classes in the class diagram is equal to union of the classes defined in the association and generalization relationships. (ii) The set of subclasses in class diagram is same as the

subclasses in the generalization relationship. (iii) The set of whole classes in the class diagram is equal to union of the whole classes in the aggregation and composition relationships. (iv) The association relation which exists in the class diagram exists in the schema described for association relationship. (v) The association relation which is in the association schema is in the class diagram. (vi) The generalization relation which exists in the class diagram exists in the schema described for generalization relationship. (vii) The generalization relation that is in the generalization schema is in the class diagram. (viii) The aggregation relation which exists in the class diagram exists in the generalization relationship schema. (ix) The generalization relation that is in the generalization schema is in the class diagram as well. (x) The composition relation which exists in the class diagram exists in the composition relationship schema. (xi) The composition relation that is in the generalization schema is in the class diagram as well.

5. Conclusion

Unified Modeling Language (UML) and Formal Methods are both useful for requirement analysis and design specification. UML is usually used at initial phase because of having much support of graphs and diagrams while formal methods are used at the later stage to describe logical model because of having rigorous mathematical tool support. Therefore, an integration of UML and formal methods is needed for systematic development of computer systems. For this purpose, an automatic generation of specification from diagrams will be much useful to capture the hidden semantics under the UML notations. In this paper, UML class diagram are linked with Z notation to achieve the above objective.

In this research, an approach is developed by linking UML to Z notation which defines a relationship among fundamentals of these techniques. Some important relationships, i.e., associations, generalization, aggregation and composition of class diagram are chosen at this level of integration. This linkage will be useful in the systems development and construction of automated tools for generating specification from the UML diagrams. For linking UML with Z notation most abstract view of the diagrams was perceived to define the generic formal models independent of a system which will be equally useful for any kind of domain problem. At first, we have described the class diagram and its variants then formal description of relationships among classes is presented.

An exhaustive survey of existing work was done before initiating this research. Some interesting work [1], [6], [7], [17], [20], [26], was found but our work and approach are different because of conceptual and abstract level integration of UML and Z notation. In the most relevant existing work either examples are taken to make integration or only syntactical mappings are defined. But in our proposed integration, we have defined both the syntax as well as semantic analysis of both approaches. The classes are transformed to schemas in Z notation where more syntax is involved while relationships are formalized focusing much on the semantics of the UML diagram.

Z is used in this research because every object is assigned a unique type providing useful programming practice. Several type checking tools exist to support the specification. The Z/Eves is a powerful tool to prove and analyze the specification which was used in this research. The rich mathematical notations made it possible to reason about behavior of a specified system more rigorously and effectively.

References

[1] A. Hall, Correctness by Construction: Integrating Formality into a Commercial Development Process, Praxis Critical Systems Limited, Springer, vol. 2391, pp. 139-157, 2002.

[2] A. M. Mostafa, M. A. Ismail, H. El-Bolok and E. M. Saad, Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 694-701, 2007.

[3] A. Moeini and R. O. Mesbah, Specification and Development of Database Applications based on Z and SQL, Proceedings of 2009 International Conference on Information Management and Engineering, 2009.

[4] B. Akbarpour and S. Tahar and A. Dekdouk, Formalization of Cadence SPW Fixed-Point Arithmetic in HOL, Integrated Formal Methods, Springer, vol. 2335, pp. 185-204, 2002.

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

[6] D. K. Kaynar and N. Lynchn, The Theory of Timed I/O Automata, Morgan & Claypool Publishers, 2006.

[7] D. P. Tuan, Computing with Words in Formal Methods, University of Canberra, Australia, 2000.

[8] F. Gervais, M. Frappier and R. Laleau, Synthesizing B Specifications from EB3 Attribute Definitions, Integrated Formal Methods, Springer, vol. 3771, pp. 207-226, 2005.

[9] H. Beek, A. Fantechi, S. Gnesi and F. Mazzanti, State/Event-Based Software Model Checking, Integrated Formal Methods, Springer, vol. 2999, pp. 128-147, 2004.

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

[11] H. Leading and J. Souquieres, Integration of UML Views using B Notation, Proceedings of Workshop on Integration and Transformation of UML Models, 2002.

[12] J. Derrick and G. Smith, Structural Refinement of Object-Z/CSP Specifications, Integrated Formal Methods, Springer, vol. 1945, 2000.

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

[14] J. M. Wing, A Specifier, Introduction to Formal Methods, IEEE Computer, vol.23(9), pp.8-24, 1990.
[15] J. Sun, J. S. Dong, J. Liu and H. Wang, A XML/XSL Approach to Visualize and Animate TCOZ, Eighth Asia-Pacific Software Engineering Conference, pp. 453-60, 2001.
[16] K. Araki, A. Galloway and K. Taguchi, Integrated Formal Methods, Proceedings of the 1st International Conference on Integrated Formal Methods, Springer 1999.
[17] M. Barjaktarovic, The State-of-the-Art in Formal Methods, AFOSR Summer Research Technical Report for Rome Research Site, Formal Methods Framework, Report F30602-99-C-0166, WetStone Technologies, 1998.
[18] M. Brendan and J. S. Dong, Blending Object-Z and Timed CSP: An Introduction to TCOZ, Proceedings of International Conference on Software Engineering, 1998.
[19] M. Heiner and M. Heisel, Modeling Safety Critical Systems with Z and Petri-nets, International Conference on Computer Safety, Reliability and Security, Springer, pp. 361–374, 1999.
[20] M. L. Shahreza, Gwandu B. A. L. and D. J. Creasey, Importance of Formal Specification in the Design of Hardware Systems, School of Electron & Electrical Engineering, Birmingham University, 1994.
[21] M. Shroff, and R. B. France, Towards a Formalization of UML Class Structures in Z, 21st International Computer Software and Applications Conference, pp. 646-51, 1997.
[22] N. H. Ali, Z. Shukur and S. Idris, A Design of an Assessment System for UML Class Diagram, International Conference on Computational Science and its Applications, pp. 539–546, 2007.
[23] O. Hasan and S. Tahar, Verification of Probabilistic Properties in the HOL Theorem Prover, Integrated Formal Methods, Springer, vol. 4591, pp. 333-352, 2007.
[24] R. Borges and A. Mota, Integrating UML and Formal Methods, Electronic Notes in Theoretical Computer Science, 184, pp. 97-112, 2003.
[25] S. A. Ehikioya and B. Ola, A Comparison of Formalisms For Electronic Commerce Systems, IEEE International Conference on Computational Cybernetics, 2004.
[26] S. A. Vilkomir and J.P. Bowen, Formalization of Software Testing Criterion, South Bank University, London, 2001.
[27] T. B. Raymond, Integrating Formal Methods by Unifying Abstractions, Springer, vol. 2999, 2004.
[28] X. He, Formalizing UML Class Diagrams: A Hierarchical Predicate Transition Net Approach, Twenty-Fourth Annual International Computer Software and Applications Conference, 2000.
[29] Z. M. Ma, Fuzzy Conceptual Information Modeling in UML Data Model, International Symposium on Computer Science and Computational Technology, pp. 331-334, 2008.