Formalization of UML Use Case Diagram—A Z Notation Based Approach

Sabnam Sengupta[1] ,Swapan Bhattacharya[2]
Department of Computer Science & Engineering, Jadavpur University, Kolkata – 700032, India.
[1]sabnam_sg@yahoo.com, [2]bswapan2000@yahoo.co.in

Abstract- **A Unified Modeling Language (UML) [1, 2] use case diagram is a visual tool that provides a way for developers to come to a common understanding with system's end users and domain experts. The behavior of a use case can be specified by describing a flow of events in text for an outsider to understand it easily. The flow of events of a use case is specified in an informal structured text. Therefore, it is not possible to automate the verification of tracking requirements captured in a use case diagram in the design phase. On the other hand, Z is a formal language, which is non-executable, but, a strongly typed specification language. We here propose a structured-expandable format of a use case, which is expressed in Z notation and then represented visually using an Entity-Relationship diagram. Implementation of our approach would bridge the gap between a formal language, which is mathematical and difficult to understand and UML use case diagram that is visual, easy to comprehend and that is used widely to capture requirements. Development of a tool based on this approach will produce a visual representation of a formalized UML use case diagram, from which automated traceability and verification of the design phase can be achieved.**

Index Terms: **UML, Use case diagram, Z notation, Formalization, ER Diagram, Traceability, Automated verification.**

I. Introduction

A use case is an object-oriented modeling construct that is used to define the behavior of a system. Interactions between the user and the system are described through a prototypical course of actions along with a possible set of alternative courses of action. Primarily, use cases have been associated with requirements gathering and domain analysis. However, with the release of the Unified Modeling Language (UML) specification version 1.1, the scope of use cases has broadened to include modeling constructs at all levels. Due to this expanded scope, the representation of use cases has taken an increasing importance.

UML use case diagrams are important for visualizing, specifying and documenting behavior of an element. They are applied to capture the intended behavior of the system being developed without specifying how the behavior is implemented. When the system is implemented, collaboration or sequence diagrams realize the use cases.

Formal specifications use mathematical notation to describe in a precise way the properties, which an information system must have, without unduly constraining the way in which these properties are achieved. This abstraction makes formal specifications useful in the process of developing a computer system. A formal specification can serve as a single, reliable reference point for all associated with the entire software life cycle. Because it is independent of the program code, a formal specification of a system can be completed early in its development. It can be a valuable means of promoting a common understanding among all those concerned with the system.

The Z notation [3, 4] is a strongly typed, mathematical, specification language. It is not an executable notation; it cannot be interpreted or compiled into a running program. There are some tools for checking Z texts for syntax and type errors in much the same way that a compiler checks code in an executable programming language. Z is a model-based notation.

One of the other main ingredients in Z is a way of decomposing a specification into small pieces called schemas. The schema is the feature that most distinguishes Z from other formal notations. In Z, schemas are used to describe both static and dynamic aspects of a system.

As "abstraction" is the key concept for both use case diagrams and Z notation, we here propose an approach to "formalize" UML use case diagram using Z notation and representing it visually using a Entity Relationship diagram as that is understandable for the users and developers.

II. Review of Related Works

Lots of research work is going on in the direction of formalizing UML using Z notation. There are a very few works in the direction of visualizing Z notation, [19] is an example of the works in that domain. In the domain of formalization, some of the works are getting done with an objective to map Object-Z [6], which is an object-oriented extension to Z with UML and some of the works are done with an objective to formalize UML with Z or any extension of Z. A comparison of formalizing approaches of UML class constructs in Z and Object-Z is presented in [12].

As in Z, a system is modeled by representing its state, a collection of state variables and their values,- and some operations that can change its state, most of the works focus on representing or translating UML class diagrams with Z notation or Object-Z. Some times, state diagrams have been used for this purpose.

We here summarize some of the relevant works, which is focused on the formalization based on Object Z, and then, based on Z or any extension of that. A XML/XSL approach for projection of Object-Z concepts to UML is discussed in [10], whereas in [11], a mapping for translating systems modeled in UML incorporating OCL in Object-Z is proposed. An extension of the development method Fusion/UML that translates the results of analysis and design into the formal specification language Object-Z is proposed in [13], which also establishes a consistency relationship between analysis and design. A translation between some of the UML models into Z and Lustre formal specifications in order to use a theorem prover and a test generator to validate the models is proposed in [14]. Here Z is used to validate the static part of the UML model using RoZ as a tool and Z-EVES theorem prover. In [16] a meta-model-based transformation between UML and Object-Z is presented, whereas in [17], a framework that integrates UML with Object-Z is presented to support requirements elicitation and analysis activities. a precise and descriptive semantics for core modeling concepts in Object-Z and a formal description for UML class constructs is presented in [18].

A meta-modular framework for the combined use of UML and Z is presented in [7] by using templates and generics. At the instantiation-level, UML models are translated into Z specifications by instantiating the corresponding meta-level semantic interpretations. This approach is entirely based on UML class diagrams. A method for formalizing syntax and semantics of UML state-chart diagrams using Z is discussed in [8]. This paper discusses the hierarchical and concurrent structures of States. AML, an augmented object oriented modeling language, which is based on UML and Z and a supporting tool Venus are presented by Xiaoping Jia in [9], but this paper did not present any technical or theoretical breakthroughs. Again, AML consists of UML class and to some extent state diagrams. An extension of Z notation is also performed for this purpose and Zext is introduced. [15] is focused towards the formalization of the primary UML constructs used to build class structures. Soon-Kyeong Kim, David Carrington in [18], use various diagrams to visualize a Z specification, which includes both static and dynamic aspects of formal specifications including complex constraints in the visualization scope.

As said earlier, most of the works discussed in this section, focus on formalization of either UML class diagram, or, statechart diagram. In this paper, we propose to formalize UML use case diagram based on which automated traceability and verification in the design phase can be achieved.

III. Scope of Work

In this paper we have used Z notation as a standard for expressing the UML use case diagram in a formal manner.

Interestingly, although the UML defines a use case, it does not prescribe any strict format or style for writing one. To address that, we propose a structured, expandable format of a use case, from which, we propose to derive a Z notation schema. The schema is type-checked using ZTC [5], a Z notation type checker and then represented visually using an ER diagram. We demonstrate our approach with the help of a case study of a retail system.

Our approach would bridge the gap between UML use case diagrams that has become a de-facto industry standard for capturing requirements and understandable by the users, domain experts and developers, and Z notation, that is formal, i.e., mathematical and difficult to understand by the users and the developers. A tool based on our approach would generate a visual representation of a "formalized" use case diagram from which automated verification in the design phase can be achieved. Our approach can be diagrammatically represented as in Fig 1.

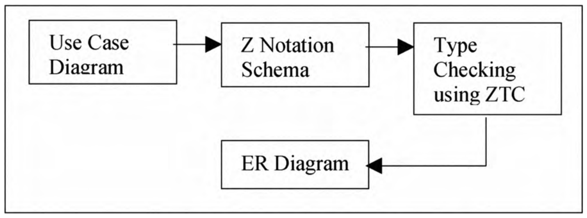

Fig 1: Diagrammatic representation of the approach discussed in this paper

IV. Representation of a Use Case Diagram Using Z Notation Schema

In this paper, we propose to map events of a use case with the use case itself. So that, when requirements are captured using use case diagrams, if some restrictions are enforced on the structure and relationship among the use cases themselves and the use cases and their events, we can formalize the use case diagram using Z notation. This will help us tracking the requirements in the design phase and also verifying that all the requirements have been addressed in the design automatically.

The following diagram is a UML use case diagram which shows an actors and use cases and relationship among the use cases.

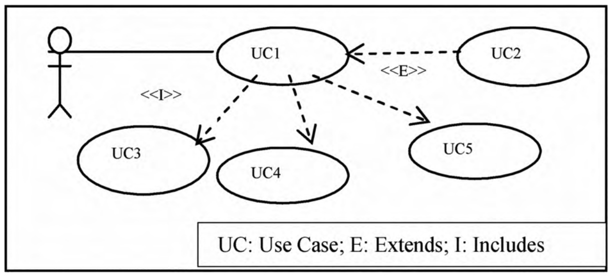

Fig 2: A UML Use Case Diagram

Interestingly, although the UML defines a use case, it does not prescribe any strict format or style for writing one.

We here propose a structured-expandable format of a use case in the following manner.

Use Case Name

1. id
2. Brief Description

3. Actors *
4. Relationship
   4.1 <<extends>>
   4.2 <<includes>>
5. Flow of Events
   5.1 Basic Flow
      5.1.1 Event 1
         5.1.1.1 ID
         5.1.1.2 Description
      5.1.2 Event 2
         5.1.2.1 ID
         5.1.2.2 Description
      5.1.3 Event 3
      …………………………
      …………………………
   5.2 Alternate Flows
      5.2.1 Alternate Flow 1
         5.2.1.1 Event 1
         5.2.1.2 Event 2
         ……………………..
         ……………………..
      5.2.2 Alternate Flow 2
      5.2.3 Alternate Flow 3

We represent each event of the Flow of events or Alternative flow of events using a Z notation schema:

```
── Event ──────────────────────────
 id: ℤ
 ucId: ℤ
 desc: seq char
 type:  seq char
───────────────────────────────────
 type ∈ {basic, alternate}
 ucId = dom UseCase
 #ucId = limit
───────────────────────────────────
```

We now represent the structure of a use case using a Z notation schema:

```
── UseCase ────────────────────────
 id: ℤ
 relationshipId: ℙ ℤ
 eventId: ℙ ℤ
───────────────────────────────────
 relationshipId= dom Relationship
 eventId = dom Event
 #eventId ≥ limit
───────────────────────────────────
```

where “limit” is a global variable can be axiomatically defined as

```
| limit : ℕ
|─────────────
| limit = 1
```

and Relationship is a schema defined as:

```
── Relationship ───────────────────
 id: ℤ
 ucId: ℙ ℤ
 relType: seq char
───────────────────────────────────
 relType ∈ {<<extend>>, <<include>>}
 ucId = dom UseCase
 #ucId = 2
───────────────────────────────────
```

V. Type Checking Z Notation Schema

Z is a non-executable but strongly-typed specification language. ZTC is a type-checker for Z, which determines if there are syntactical and typing errors in Z specifications. There is no compiler for Z.

ZTC accepts two forms of input: LATEX with oz or zed packages, and ZSL. The ZSL form of input (plain text style) for the schema Event is as follows:

```
Schema Event
    id: Z
    ucId: Z
    desc: seq char
    type: seq char
where
    type ∈ {basic, alternate}
    ucId = dom UseCase
end schema
```

The ZSL form of input for the schema Event is as follows:

```
Schema UseCase
    id: Z
    relationshipId: P Z
    eventId: P Z
where
    relationshipId = dom Relationship
    eventId = dom Event
    #eventId ≥ limit
end schema
```

and the global variable limit can be written using ZSL as:

```
global
    limit : N
axiom
    limit = 1
end axiom
```

and the ZSL form of input for the Relationship schema is:

```
Schema Relationship
    id: Z
    ucId: P Z
    relType: seq char
where
    relType ∈ {<<extend>>, <<include>>}
    ucId = dom UseCase
    #ucId = 2
end schema
```

All these ZSL forms of input are saved with .zsl extension. For example, the ZSL input form for the schema is saved as “Event.zsl” and so on.

After saving these input files with the .zsl extension, ZTC is run to check for type errors with the command:

```
 ZTC Event.zsl -T
```

requesting a type report.

VI. Derivation of E-R Diagram from Z Notation Schema

We can represent the Z notation schemas representing a Use Case, an event and a relationship between two use cases in an ER diagram as shown in Fig 3:

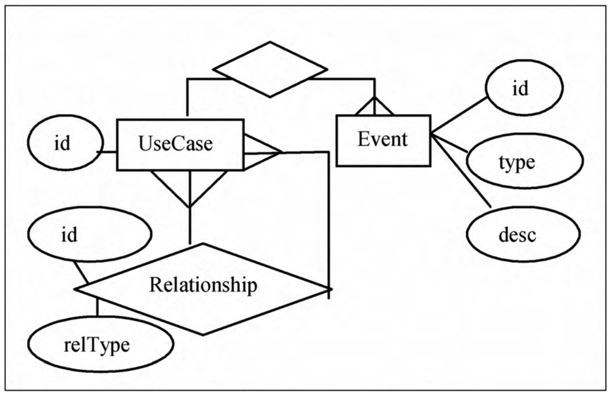

Fig 3: ER diagram derived from Z Notation Schema of Use Case Diagram

This ER diagram is implemented in the tabular format as below:

Use Case ID	Use Case Name	Event ID	Event Desc

Relationship ID	Relating Use Case ID	Related Use Case ID	Relationship Type

VII. Case Study

The use case diagram of a retail system is given in fig 5 in Appendix.

We are considering a system where both the customer and a customer service representative can place an order and track an order. The customer service representative can also give instructions for shipping an order and bill a customer. The customer service representative can also give instructions for shipping a partial order provided a "Place Rush Order" option was selected by the customer while placing the order. All these activities include an activity "Validate Customer"

The partial Table structure of this system corresponding to the ER diagram is given in Table 1 and Table 2 of Appendix. For brevity, only parts of the table structures are shown.

VIII. Future Scope of Work

In future we intend to focus on automated verification of requirements captured in a use case diagram in the design phase. In order to do so, we can derive a "Use Case Class Diagram" corresponding to the proposed ER diagram. This "Use Case Class Diagram" is different from UML class diagram and can be represented as in Fig 4.

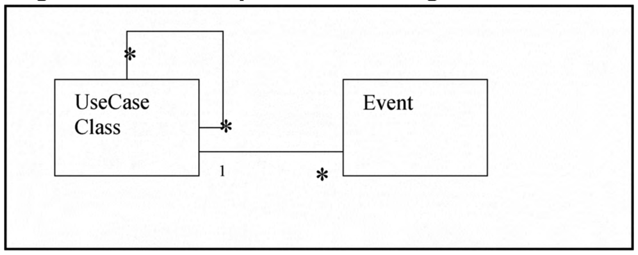

Fig 4: Use Case Class Diagram corresponding to the ER Diagram

At this stage, we can make a class of a class diagram be the child of a UseCaseClass. This will help us automated tracking requirements captured in a use case in the design phase.

With an extension of our approach we can achieve formalization of UML diagrams from which automated verification, consistency checking and execution is possible.

We now explain our approach with the help of a case study of a retail system.

IX. Conclusion

Software development is a human intensive activity and the need for better and higher quality software has led to the evolving field of software engineering. In the recent decades, we have witnessed a phenomenal increase in the functional and structural complexity of software systems being produced. Object oriented methodology has emerged as one of the most common paradigms for design and development of information systems today. Timely production of such products and verification of their functional correctness has been major challenges. In particular, it has been observed that incorrect or incomplete specification of the software is often a major hindrance in the development process. A Formal Specification, on the other hand, is made in a formal language, and hence is unambiguous, and can be tested for correctness. It is cost-effective to formalize characteristics at initial stages of software development. So, it makes much more sense from the perspective of cost effectiveness as well as software evolution if we can formalize the behavior of the system right at the functional specification level.

The adoption of UML as a standard for modeling functional specifications of object-oriented systems has made modeling simpler and easy to understand with lots of tools supporting it. However, UML being a visual language is semi-formal in nature and hence formalizing functional specification using UML has triggered challenging opportunities of research in this domain. There is mainly one direction of research - focusing on formalization of one or more UML diagrams.

In this paper, we propose to formalize Use Case diagram, which has become a de-facto industry standard for capturing requirements with Z notation and again represent it visually using ER diagram.

Formalization of UML diagrams would empower us achieving automated execution at different stages of software development that will remove ambiguities at all stages.

Our work is a step forward in that direction.

Reference:

[1] Grady Booch, James Rumbaugh, Ivar Jacobson; The Unified Modeling Language User Guide (1999)
[2] OMG: Unified Modeling Language Specification, version 2.0. Available at http://www.omg.org/uml
[3] J.M. Spivey, The Z Notation, A Reference Manual, 2nd edition. Prentice Hall International, 1992.
[4] Jim Woodcock, Jim Davis; Using Z Specification, Refinement and Proof, Prentice Hall, 1996
[5] Xiaoping Jia, ZTC: A Type Checker for Z Notation User's Guide Version 2.2, October 2002
[6] Smith G.O., The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers (2000)

[7] Nuno Am´alio, Susan Stepney, and Fiona Polack, Modular UML Semantics: Interpretations in Z Based on Templates and Generics, FACS'03 Workshop on Formal Aspects of Component Software, Pisa, Italy, September 2003
[8] Xuede Zhan, Huaikou Miao, Ling Liu, Formalizing the Semantics of UML Statecharts with Z*, The Fourth International Conference on Computer and Information Technology (CIT'04) 09 14 - 09, 2004
[9] Xiaoping Jia, A Pragmatic Approach to Formalizing Object-Oriented Modeling and Development, COMPSAC, 1997
[10] Jing Sun, Jin Song Dong, Jing Liu, Hai Wang, Object-Z Web Environment and Projections to UML, 10th International World Wide Web Conference (WWW-10), May 2001.
[11] David Roe, Krysia Broda, and Alessandra Russo, Mapping UML Models incorporating OCL Constraints into Object-Z, Technical Reports - Computing - Imperial College London, 2003
[12] Nuno Amalio and Fiona Polack, Comparison of Formalism Approaches of UML Class Constructs in Z and Object-Z, 2003
[13] Margot Bittner, Florian Kamm¨uller, Translating Fusion/UML to Object-Z, Technical University of Berlin, June 2003.
[14] Sophie Dupuy-Chessa and Lydie du Bousquet, Validation of UML models thanks to Z and Lustre, Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity, 2001
[15] M. Shroff, R.B. France, Towards a formalization of UML class structures in Z, COMPSAC '97 - 21st International Computer Software and Applications Conference 08 11 - 08, 1997 Washington, DC
[16] Soon-Kyeong Kim, David Carrington, Roger Duke, A Metamodel-based transformation between UML and Object-Z, Queensland IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01) 09 05 - 09, 2001 Stresa, Italy
[17] S.-K. Kim, D. Carrington, An integrated framework with UML and Object-Z for developing a precise and understandable specification: the light control case study, Seventh Asia-Pacific Software Engineering Conference (APSEC'00) 12 05 - 12, 2000 Singapore
[18] Soon-Kyeong Kim and David Carrington, A Formal Mapping between UML Models and Object-Z Specifications, Proceedings of the First International Conference of B and Z Users on Formal Specification and Development in Z and B, 2000
[19] Thomas Tilley, Towards an FCA based tool for visualizing formal specifications, ICCS 2003, The 11th International Conference on Conceptual Structures, July 2003, Dresden.

Appendix

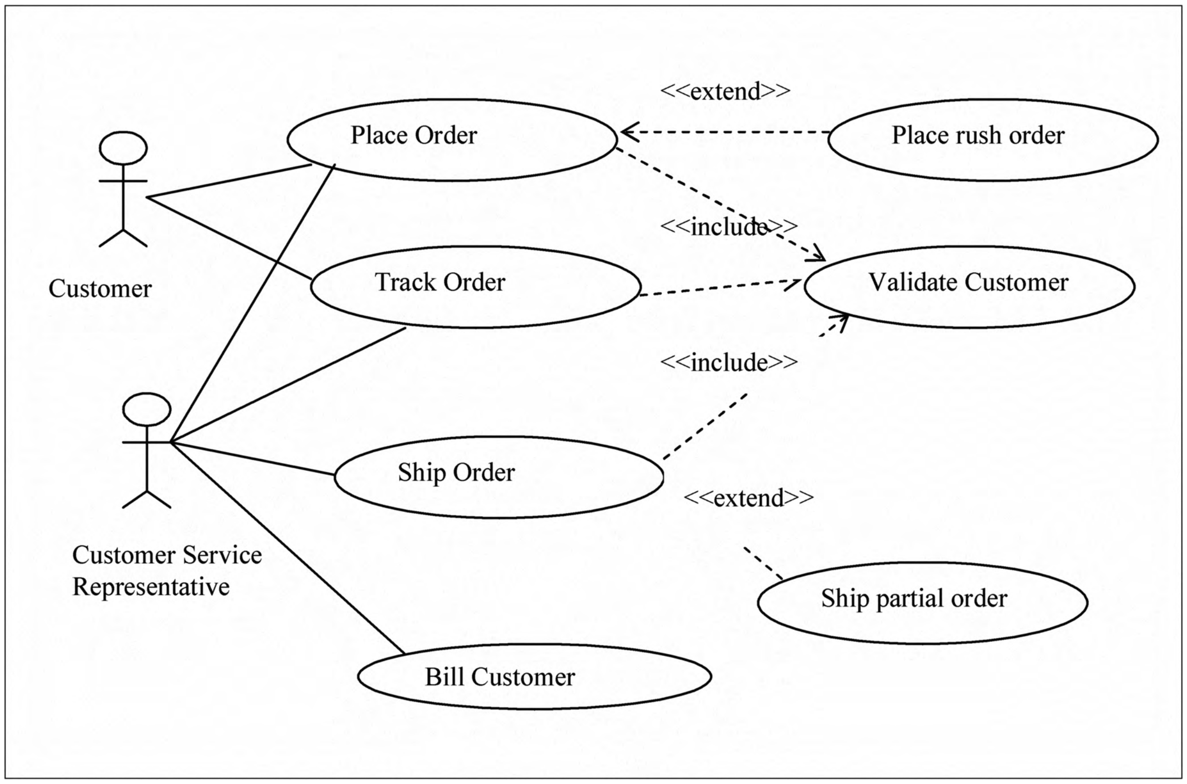

Fig 5: Use Case Diagram of a Retail System

TABLE I
PARTIAL TABLE DEMONSTRATING USE CASES AND THEIR EVENTS

Use Case ID	Use Case Name	Event ID	Event Type	Event Description
01	Place Order	01	Basic	The customer or the representative enters the product ID/s using keyboard.
01	Place Order	02	Basic	The system checks whether the product ID/s is/are valid or not.
01	Place Order	03	Basic	The product ID/s is/or valid and the system prompt the user to enter the customer ID and pin.
01	Place Order	04	Basic	The "Validate Customer" use case is *included.*
05	Ship Order	29	Basic	The customer service representative enters the customer Id and order Id using keyboard.
05	Ship Order	30	Basic	The "Validate Customer" use case is *included.*
05	Ship Order	30	Basic	The system validates the order ID.
05	Ship Order	31	Basic	The system shows the product/s associated with the order and if they are available or not.
05	Ship Order	32	Basic	If all the products are available, the representative generates a "Ship" order, the use case ends
06	Ship Partial Order	39	Basic	"Ship Order" use case is *extended.*

TABLE II
PARTIAL TABLE DEMONSTRATING USE CASES AND THEIR RELATIONSHIP

Relationship ID	Relating Use Case ID	Related Use Case ID	Relationship Type
01	01	04	<<include>>
02	02	01	<<extend>>
03	03	04	<<include>>
04	05	04	<<include>>
05	06	05	<<extend>>