VERIFYING DATABASE CONCEPTS DURING FORMAL METHOD TOWARDS CORRECTION BY CONSTRUCTION

Enas E. El-Sharawy*
Computer Dept., College of Science and Humanities in Jubail,
Imam Abdulrahman Bin Faisal University, Kingdom of Saudi Arabia, Jubail Industrial City
eeelsharawy@iau.edu.sa

Thowiba H. Elawd
Computer Dept., College of Science and Humanities in Jubail,
Imam Abdulrahman Bin Faisal University, Kingdom of Saudi Arabia, Jubail Industrial City
teahmed@iau.edu.sa

Linda O. Elbadry
Computer Dept., College of Science and Humanities in Jubail,
Imam Abdulrahman Bin Faisal University, Kingdom of Saudi Arabia, Jubail Industrial City
loeali@iau.edu.sa

Abstract - Nowadays the generation of the reliable and verified database model is considered a challenge in the software engineering field, so Using the formal method "Event-B" as a validation tool for the database becomes an effective and objective concept to evaluate any database model. Event-B is a formal method which the usage of for modeling and verifying the software program construction. The functional properties of the Database require particular descriptions and also, the valid operations of many essential databases are established. Therefore, a verifiable database design system is incredibly desirable. This research pursuit to present database in the formal method and acquire demonstrated and automatic Event-B code that guided by the Student enrolment case study, using the validation in UML-B and verification approach with the RODIN platform tool. UML-B is a graphical plug-in of the RODIN framework and the Event-B formal method language helps the modeler in modeling and Validation of the generated database model.

Keywords: Formal Methods, Event-B, UML-B, DataBase, Refinement

1. Introduction

Database frameworks hold huge resources whereupon basic choices depend. These resources and choices can be a piece of wellbeing or business basic spaces like wellbeing and patient frameworks or endeavor smart frameworks. This accentuates the way that database frameworks are a significant field in software engineering [1] and along these lines require an evident and thorough structure and execution.

This research attempts to address the topic of how to validate structure databases so the database will be valid by construction. To address this, we propose an approach for a model-based database plan in the formal method utilizing UML-like documentation that supports the design and validation of framework models. We use Event-B which is a formal method for thorough particular and check of computerized frameworks [2]. It has been upheld in an open instrument stage called Rodin [3]. We model our framework utilizing UML-like documentation in the Rodin apparatus called UML-B [4] which supports demonstrating in class outlines and state machine charts. The UML-B instrument makes an interpretation of UML-B models to Event-B models and the Rodin device is utilized to confirm their consistency. As class charts are normally used to demonstrate database frameworks, utilizing UML-B class outlines will be clearer for database creators than displaying databases straightforwardly in Event-B. This paper is organized as pursues: In Section 2, we give foundation about the themes that are connected to this research "main concepts". Section 3 portrays how to model and validate the database through Event-B**.** In Section 4, we plot the most related work. Finally, the Conclusion is presented and displayed in Section 5.

2. Main Concepts

2.1. Event-B Modeling language

Event-B is one of the formal methods used for modeling and verification methods. Event-B model has two structures: First; the static part "context" that defines the "types", "constants" and "axioms"; Second; the dynamic part "machine" with "variables", "invariants" and "events" occur before the event execution. Event results "actions" change the variable state. Every event that constructs the "machine" must preserve all defined of its "invariant". Event-B refinement makes a modeler to gradually build the model with among abstract levels [2].

The refinement technique is the main concept in the Event-B which enables a modeler to gradually build the system so it becomes more precise and closer to reality [5].

To apply the Event-B refinement to a "context" in Event-B can be achieved using new sets, constants, and axioms adding. To apply the Event-B refinement to a "machine" may include adding new "variables", invariants and new "events" or refining existing events [2].

Compared to different formal methods Like VDM [6], Z [7] and B method [8], Event-B consider more flexible modeling language with validation that is executed by proof Obligation. This feature is important in our research as verified databases containing many valid operations and associations.

2.2. UML-B

UML-B is a graphical framework for the Event-B formal modeling language that is based on UML diagrams "Unified Modeling Language" [4]. The model is converted or translated into Event-B for automatic Event-B code also considers as a verification method. UML-B supports modeling with "class diagrams" which are used to describe the data structure and the behavioral part of the database [4].

UML-B enables the modeler to design one of three different types when adding an event to a class. These kinds are "normal", "constructor" or "destructor" events.

UML-B class diagram also must be refined by adding new "attributes" or new "associations". [9].

3. Modeling and Validation database through Event-B

This section presents how to establish a database model in UML-B. We will use an example for the student enrolment database. Modeling databases gradually are building by abstraction and refinement in Event-B and UML-B. This method assists the modeler to decrease the complexity of a model. Since the Event-B proofs are automatically generated when using refinement in Event-B, the modeler can make sure that the database is valid by construction.

3.1. UML-B Class Validation

Assume we have two classes in the Student database enrolment system called X and Y with no attributes definition. In UML-B we create them also with AddY,RemoveY and UpdateR events for Y class; AddX and RemoveX events for X class as shown below.

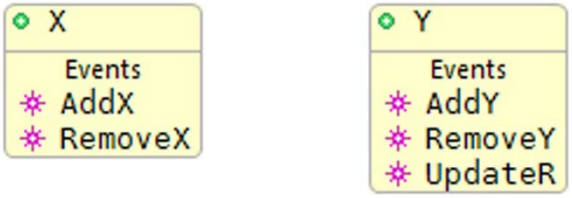

Fig. 1. Example of UML-B class diagram (Abstract model)

In this phase, we apply the automatic generation of UML-B to the Event-B model but we get the following result as shown in figure2.

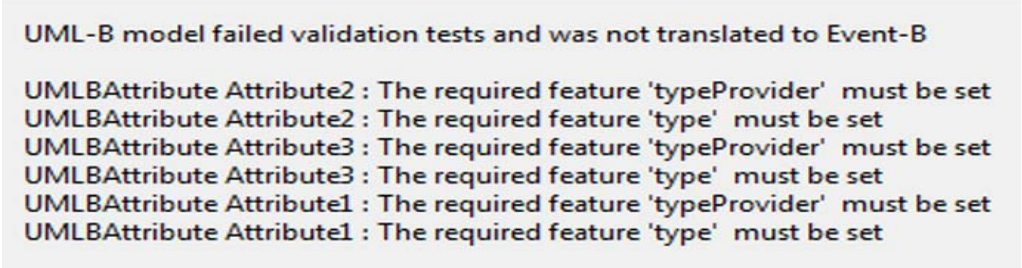

Fig. 2. UML-B Validation

Attribute1, Attribute2, and Attribute3 need a type declaration like in database application creation. So, we solved this issue then try again to translate UML-B to Event-B.

After assigning the attributes to classes the translation from UML-B to Event-B will success then we get the following Event-B model or code.

MACHINE

m1

SEES

m1_implicitContext

VARIABLES

Y // *class instances*

X // *class instances*

INVARIANTS

Y.type : Y ∈ □ (Y_SET)

X.type : X ∈ □ (X_SET)

EVENTS

INITIALISATION ≙

STATUS

ordinary

BEGIN

Y.init : Y ≔ ∅

X.init : X ≔ ∅

END

AddX ≙

STATUS

ordinary

ANY

self // *contextual instance of class Y*

WHERE

self.type : self ∈ Y

THEN

skip

END

RemoveX ≙

STATUS

ordinary

ANY

self // *contextual instance of class Y*

WHERE

self.type : self ∈ Y

THEN

skip

END

AddY ≙

STATUS

ordinary

ANY

self // *contextual instance of class X*

WHERE

```
        self.type  :  self ∈ X
THEN
        skip
END
    RemoveY  ≙
    STATUS
    ordinary
ANY
        self        //  contextual instance of class X
WHERE
        self.type  :  self ∈ X
THEN
        skip
END
    UpdateR  ≙
    STATUS
    ordinary
ANY
        self        //  contextual instance of class X
WHERE
        self.type  :  self ∈ X
THEN
        skip
END
END
```

3.2. UML-B Class Association Validation

In this step, we create a general class in our example with two subclasses staff and students. After we design the classes we add the class association between classes. When we save our design in UML-B so the UML-B validation for our design will produce if the design is correct or not. Figure 3 shows that something is not correct. All red font in the classes points to UML-B validation.

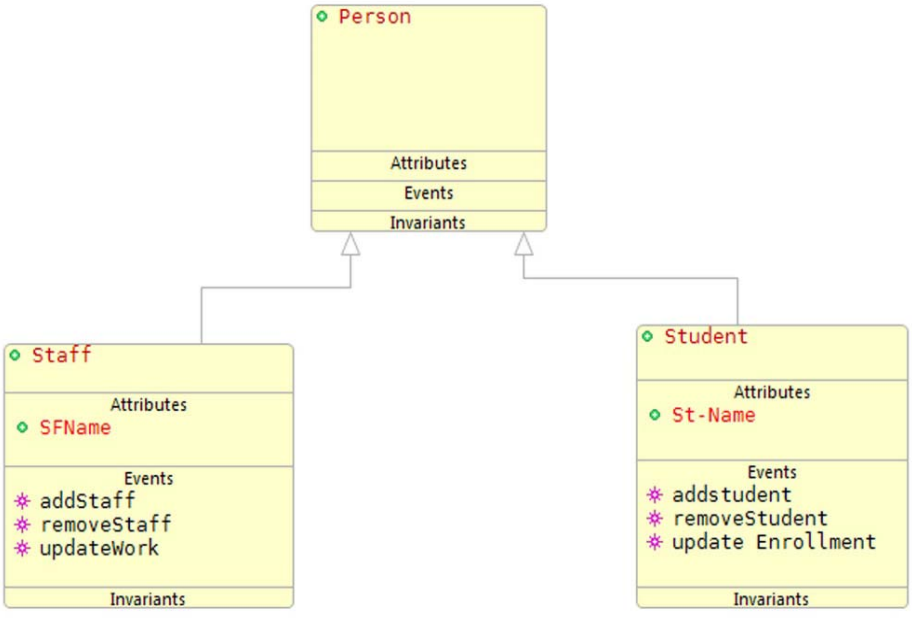

Fig. 3. UML Validation for classes association

UML-B properties window indicates the validation according to class "Staff" with attribute "SFName" as shown in figure 4.

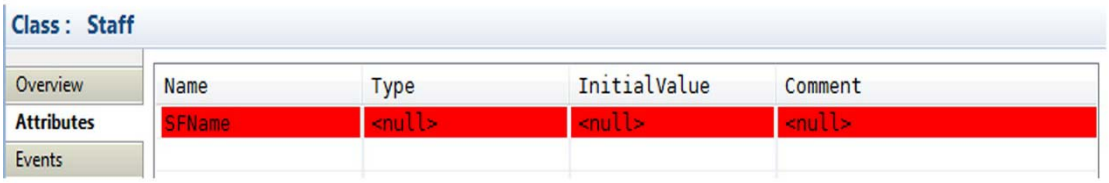

Fig. 4. Class "Staff" attributes validation

UML-B properties window indicates the validation according to class Staff with attribute "STName" as shown in figure 5.

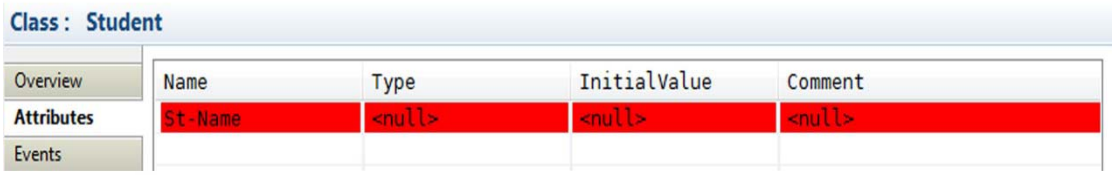

Fig. 5. Class "Student" attributes validation

3.3. Automatic generation from UML-B to Event-B

In this phase, we apply Event-B validation when UML-B translation. Figure 6 shows that attribute 1, attribute2, St-Name, and SFName have required feature "type provider" and "type" must be set. When this window appear that mean the UML-B translation failed and some manual editing is required.

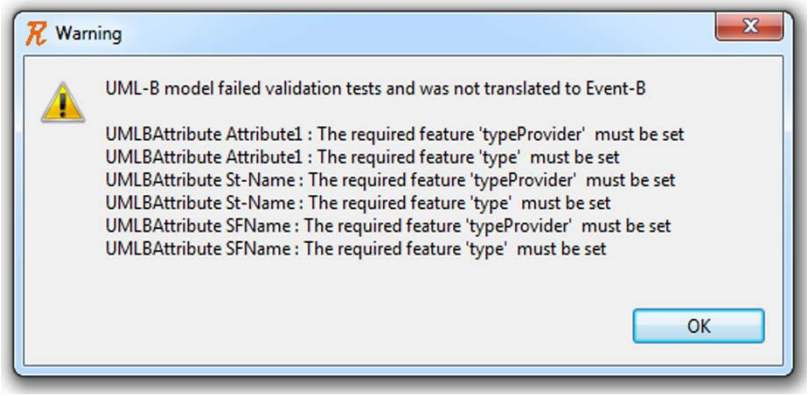

Fig. 6. Event-B validation

After the Event-B validation detection is done; we correct the required types to attributes. Now the translation of UML-B class to Event-B model success as shown in the following code:

MACHINE

Machine1

SEES

Machine1_implicitContext

VARIABLES

Person // *class instances*

Student // *class instances*

Staff // *class instances*

St-Name // *attribute of Student*

SFName // *attribute of Staff*

INVARIANTS

Person.type : Person ∈ □ (Person_SET)
Student.type : Student ∈ □ (Person)
Staff.type : Staff ∈ □ (Person)
St-Name.type : St−Name ∈ Student → Person
SFName.type : SFName ∈ Staff → Person
Invariant1 : ⊤
Invariant3 : ⊤
Invariant2 : ⊤

EVENTS

INITIALISATION ≙
STATUS
ordinary

BEGIN

Person.init : Person ≔ ∅
Student.init : Student ≔ ∅
Staff.init : Staff ≔ ∅
St-Name.init : St−Name ≔ ∅
SFName.init : SFName ≔ ∅

END

Event1 ≙
STATUS
ordinary

ANY

self // *contextual instance of class Person*

WHERE

self.type : self ∈ Person

THEN

skip

END

addstudent ≙
STATUS
ordinary

ANY

self // *contextual instance of class Student*

WHERE

self.type : self ∈ Student

THEN

skip

END

removeStudent ≙
STATUS
ordinary

ANY

self // *contextual instance of class Student*

WHERE

self.type : self ∈ Student

THEN

```
        skip
END
    update Enrollment ≙
    STATUS
    ordinary
ANY
        self        //  contextual instance of class Student
WHERE
        self.type   :   self ∈ Student
THEN
        skip
END
    addStaff ≙
    STATUS
    ordinary
ANY
        self        //  contextual instance of class Staff
WHERE
        self.type   :   self ∈ Staff
THEN
        skip
END
    removeStaff ≙
    STATUS
    ordinary
ANY
        self        //  contextual instance of class Staff
WHERE
        self.type   :   self ∈ Staff
THEN
        skip
END
    updateWork ≙
    STATUS
    ordinary
ANY
        self        //  contextual instance of class Staff
WHERE
        self.type   :   self ∈ Staff
THEN
        skip
END
END
```

3.4. Event-B Validation "Proof Obligations:

After all the previous steps of validation beginning from UML-B validation to Event-B validation through translation from UML-B to Event-B model, we get Event-B model that verified by the asset of proof obligation as shown in figure 7.

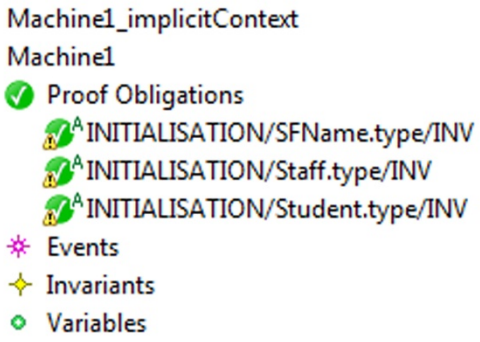

Fig. 7. Proof Obligation for generated Event-B model

3.5 Summary of approach

The following diagram that called figure 8 present the phases of our approach.

By using our approach to model a database in Event-B Also, this approach can help the modelers to model verified and a complex system using UML_B. The approach can be summarized in the following steps shown in figure8 :

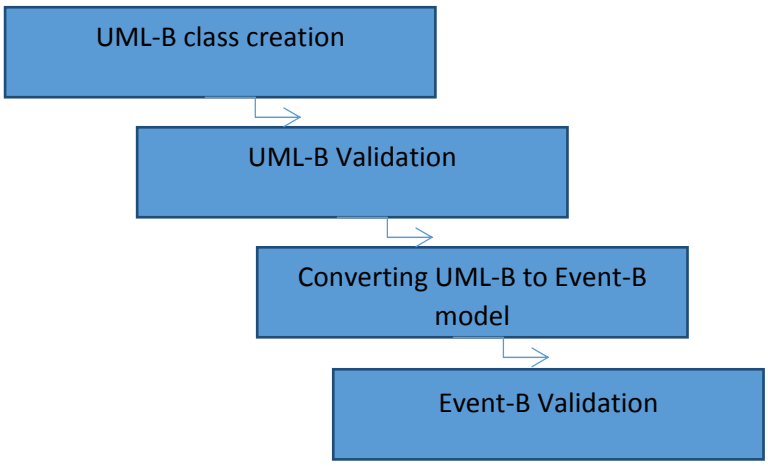

Fig. 8. Our approach gradually phases

- Modeling UML-B class.
- UML-B validation.
- Converting UML-B to Event-B model.
- Event-B Validation.

4. Related Work

Schlatter and Aicherig introduce a database development of an industrial project using VDM-SL that considers one of the formal methods [10].

Enas and et al. [11] studied nine UML class diagrams, which they uploaded as UML templates to be used as patterns then detect an anti-pattern get an important result UML-B consider as anti-pattern detection and validation tool.

The authors in [12] and [13] formalize relational databases in "Z" specifications; Z also considers as one popular language of formal methods.

In [12] Barros introduce a different student database functions as well as "transactions", "sorting", "aggregations" and other different database elements. There is no tool provided in which modelers can use to automatically generate a database for the formal methods tools.

In [14] Davies et al. present how to formalize a database using UML and OCL "Object Constraint Language" [15] using the notation called Booster [16].

Mammar and Laleau in [17] have also shown the database by the formal method using UML graphic notation. Their research helps modelers in designing databases using a UML diagram and then translates that model to a B specification and on to Java and SQL code.

Wang and Wahls in [18] developed a new plug-in in the framework called Rodin that can generate Java and JDBC code to build database and design database queries.

None of these related works can give a strong assurance for modeling and validating database in the formal methods so we can get database model correct and valid by construction. Moreover, they do not cover the validation of the UML-B or Event-B model. The approach of modeling and validation database by formal methods modeling steps is an important contribution of this work.

Modeling in Event-B is dependent on layered refinement and mathematical proof as stated in [19] and [2], our contribution in this research is modeling and validating database through multiple phases beginning by UML-B class diagram until to get Event-B model "code". We present different validation areas like UML-B validation, Event-B generation, and Event-B validation.

5. Conclusion

Formal methods languages help the modeler for modeling database is a strong issue which has been considering as validation by construction in software engineering. The related work to our research does not address how to design the model in different validation phases with validation assurance where each phase the modeler presents the specification of the database and methods for validation. Our research provides an approach for modeling and validating the databases with different overviews through steps of our approach.

References

[1] Thomas M Connolly & Carolyn E Begg (2005): Database systems: a practical approach to design, implementation, and management. Pearson Education.
[2] Jean-Raymond Abrial (2010): Modeling in Event-B: system and software engineering. Cambridge University Press, doi:10.1017/CBO9781139195881.
[3] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta & Laurent Voisin (2010): Rodin: an open toolset for modeling and reasoning in Event-B. International journal on software
[4] Colin Snook & Michael Butler (2008): UML-B and Event-B: An Integration of Languages and Tools. In: Proceedings of the IASTED International Conference on Software Engineering, SE '08, ACTA Press, Anaheim, CA, USA, pp. 336–341. tools for technology transfer 12(6), pp. 447–466, DOI:10.1007/s10009-010-0145-y.
[5] Jean-Raymond Abrial & Stefan Hallerstede (2007): Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamenta Informaticae 77(1-2), pp. 1–28.
[6] Cliff B Jones (1990): Systematic software development using VDM. 2, Citeseer
[7] J Michael Spivey & JR Abrial (1992): The Z notation. Prentice-Hall Hemel Hempstead.
[8] Jean-Raymond Abrial & Jean-Raymond Abrial (2005): The B-book: assigning programs to meanings. Cambridge University Press.
[9] Eman K. Elsayed, Enas E. El-Shara(2018): Detecting Design Level Anti-patterns; Structure and Semantics in UML Class Diagrams, Journal of computer, Volume 13, Number 6. doi: 10.17706/jcp.13.6.638-654 Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[10] Rudi Schlatte & Bernhard K Aichernig (1999): Database development of a work-flow planning and tracking system using VDM-SL. In: Workshop Materials: VDM in Practice, pp. 109–125.
[11] Enas E. Elsharawy and etl. : (2016): Semantic Anti-patterns Detection in UML Models based on Ontology Catalogue", Artificial Intelligence and Machine Learning Journal, ISSN:1687-4846, vo.16, No.1, Delaware, USA.
[12] Roberto Souto Maior Barros (1998): On the formal specification and derivation of relational database applications. Electronic Notes in Theoretical Computer Science 14, pp. 3–29, DOI:10.1016/S1571-0661(05)80226-9.
[13] Saeed Khalafinejad & Seyed-Hassan Mirian-Hosseinabadi (2013): Translation of Z specifications to executable code: Application to the database domain. Information and Software Technology 55(6), pp. 1017–1044, DOI:10.1016/j.infsof.2012.12.007.
[14] Jim Davies, James Welch, Alessandra Cavarra & Edward Crichton (2006): On the generation of object databases using Booster. In: Engineering of Complex Computer Systems, 2006. ICECCS 2006. 11th IEEE International Conference on, IEEE, pp. 10–pp, DOI:10.1109/ICECCS.2006.65.
[15] Jos Warmer & Anneke Kleppe (1999): The Object Constraint Language: Precise Modeling with UML.
[16] Jim Davies, Charles Crichton, Edward Crichton, David Neilson & Ib Holm Sørensen (2005): Formality, evolution, and model-driven software engineering. Electronic Notes in Theoretical Computer Science 130, pp. 39–55, DOI:10.1016/j.entcs.2005.03.004.
[17] Amel Mammar & R´ egine Laleau (2006): From a B formal specification to an executable code: application to the relational database domain. Information and Software Technology 48(4), pp. 253–279, DOI:10.1016/j.infsof.2005.05.002.
[18] Qi Wang & Tim Wahls (2014): Translating Event-B machines to database applications. In: Software Engineering and Formal Methods, Springer, pp. 265–270, DOI:10.1007/978-3-319-10431-7 19.
[19] Michael Butler (2013): Mastering System Analysis and Design through Abstraction and Refinement. In: Engineering Dependable Software Systems, IOS Press, pp. 49–78, DOI:10.3233/978-1-61499-207-3-49.