JSc - EUSL (2016) Vol.7, P 19 - 34

ISSN 1391 - 586X : © 2016 by Faculty of Science, Eastern University, Sri Lanka

Teaching Formal Methods using Scaffolding: A Semi-automated Translation Process from UML Classes to Schemas

S. Mohanarajah* and T. Sritharan **

*Department of Mathematics and Computer Science, Claflin University, Orangeburg, USA.

**University of Colombo School of Computing, University of Colombo, Colombo, Sri Lanka.

Abstract

Formal Methods in Software Engineering is an inherently complex and challenging discipline to learn. This research attempts to design an instructional strategy to simplify the process of learning an Object Oriented Formal Language known as Object-Z using a Computer Based Learning system. A constructionist instructional technique is used to promote active learning by encouraging students to construct more complex artefacts such as formal models based on less complex ones such as semi-formal models. A step-by-step methodology is implemented to transform simple UML class diagrams to Object-Z schemas. This methodology could be extended to implement a semi-automated translation system for UML to Object Models. Scaffolding is used at the initial learning stages to alleviate the difficulty associated with complex transformation processes. The proposed instructional strategy brings various techniques together to enhance the learning experience. A functional prototype is implemented to teach the fundamental concepts of Object-Z. Both objective and subjective evaluations using the prototype indicate that the proposed CBL system has a statistically significant impact on learning Object-Z.

Keywords: Formal Methods, Object-Z, Computer Based Learning Systems

I. Formal methods

Formal specification is an important vehicle to attain reliability in the system development process. This is often taught in the third year at Universities [1]. However, based on the undecidability result, it is impossible to build a general formal tool that could verify whether an algorithm is a solution to a problem. The recent report 'Formal Methods for Safe and Secure Computer Systems' [16] suggests three ways to work around the 'undecidability' obstacle: restrict to systems for which verification is decidable, limit to some weaker but sufficient specifications and finally use semi-automated verification systems that require human intervention.

Despite its importance in industry and commerce, formal specification has not been well received by software engineering students. Martin Fowler, a leading object technology consultant, states, "Formal methods are hard to understand and manipulate, often harder to deal with than programming languages" ([15], p. 12). The following are some of the reasons stated in the literature relating to students' difficulties in learning formal methods:

- Insufficient mathematical ability
- Complex notation and structure
- Lack of motivation (at the beginning) [31]
- Inability to abstract details [10].

In this research, we investigates whether a Computer Based Learning (CBL) with scaffolding using a semi-automated step-by-step transformation process from UML diagrams to Schemas would help students to learn formal methods. Initially, we selected Object-Z for our study because it is an object oriented formal language and it is more related to UML than any other formal languages. We implemented a prototype and conducted an evaluation.

2. OBJECT-Z: AN OVERVIEW

Object-Z is an OO formal notation based on Z ([9], [10]). Z is a state-based formal notation [4]. The class schema in Object-Z is a named entity, which encapsulates the state schema and a collection of operation schemas. Figure 1 outlines the structure of the class schema

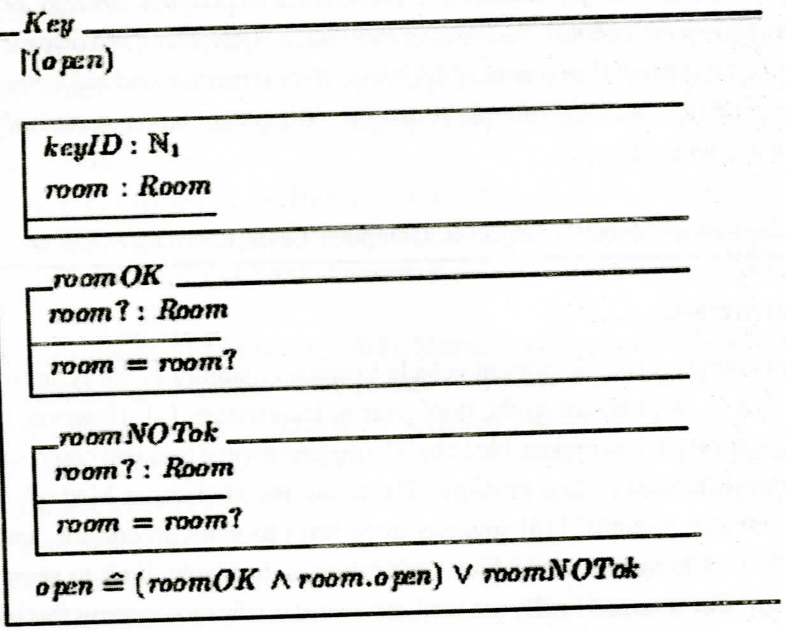


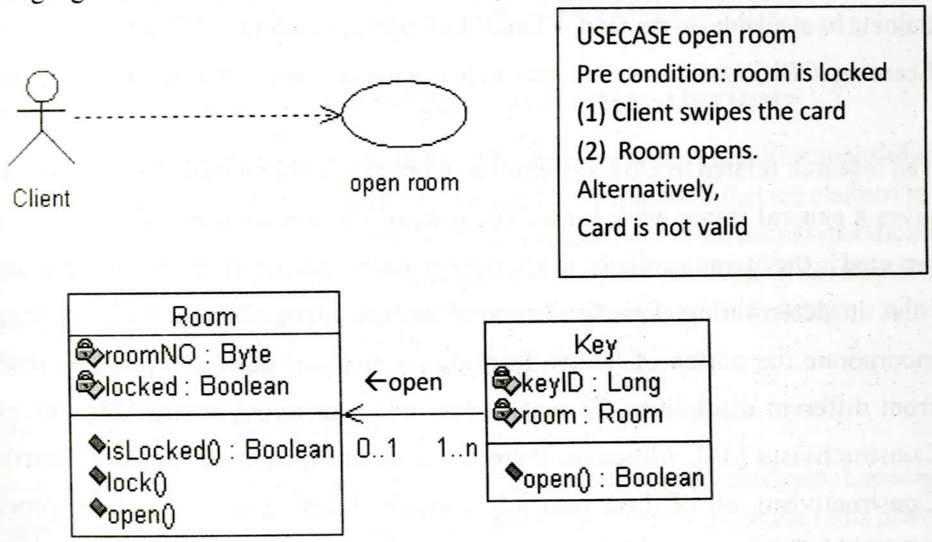
Figure 1: Class Schema in Object-Z

Schemas in Object-Z, similar to Z, have two parts: the declaration part and the predicate part. The declaration part primarily lists the relevant identifiers with types, and the predicate part contains constraints pertinent to the schema, in first-order predicate logic. If there are no such constraints, the predicate part may be omitted.

Object orientation views a software system as a collection of interacting objects; therefore, the specification of a system will contain a collection of class schemas. Each class can be easily understood in isolation. There are some composition operations (e.g. the parallel operator) available in Object-Z to cater for message passing between objects. Special constructs are available to specify inheritance and polymorphism. The notion of Object containment is used to specify object compositions. The class union feature in Object-Z, which is not available in UML, facilitates rich class associations. Some of the important constructs are explained in the following paragraphs using a case study (see Figure 2). For more information, refer to [9], [10]

Case Study

A building has many rooms. Each room has at least one key (a magnetized card). A key can open only one room. At any time, a room may be in a locked or unlocked state. Opening a given room is considered the main requirement of this system.



USECASE open room
Pre condition: room is locked
(1) Client swipes the card
(2) Room opens.
Alternatively,
Card is not valid

Figure 2: Case Study

3. UML: AN OVERVIEW

UML is just a collection of complex graphical notations, albeit a powerful one, used in the OO software development process [6]. It includes a number of different diagrams for use in different phases of the software development process. It can also be used to model hardware functions, business processes, etc. UML is an outcome of combining a collection of best practices within the software development process [15] and is a de-facto standard for OO software development notation. The main artefacts are defined by UML notations themselves. It can be extended through the use of stereotypes. The use case diagram, class diagram and interaction-sequence diagram are all important diagrams in the UML notation [6], [5], and [15].

4. COMPUTER BASED LEARNING SYSTEMS (CBL SYSTEMS)

With recent advances in information and communication technologies, the educational needs of society demand radical changes in learning practices. The traditional notion of primary-secondary-tertiary education followed by professional work has become inadequate. The current knowledge-driven, competitive, marketing economy requires that knowledge and training be available on-demand, offered in a flexible mode and be life-long. Computer Based Learning (CBL) systems are expected to play a major role in catering for these changes [7].

The research related to CBL systems has more than a four decade long history. Figure 3 gives a general frame work for a CBL system. There have been thousands of systems reported in the literature related to a variety of disciplines. Learning theories play important roles in determining their fundamental architectures. Current CBL systems often incorporate the notion of 'active learning', which has been promoted by researchers from different disciplines for many years under different names [28], for example, Constructivism [34]. Although, there are different theoretical positions attributed to Constructivism, all of them basically consider learning to be an active process and knowledge is constructed by the learner, based on past experience [22].

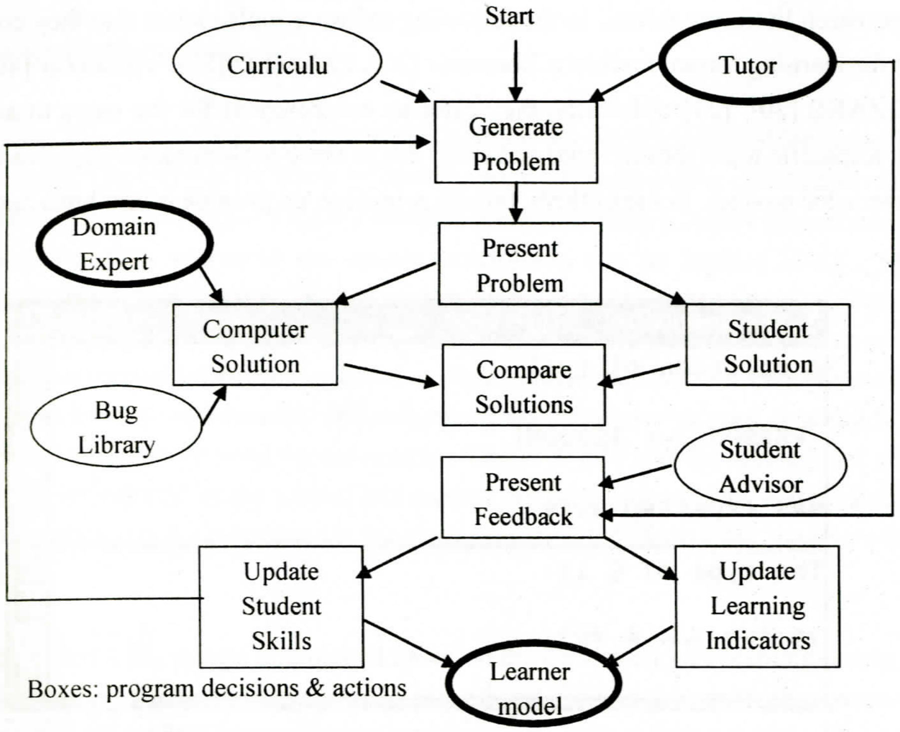


Figure 3: Computer Based Learming Systeams (after [36])

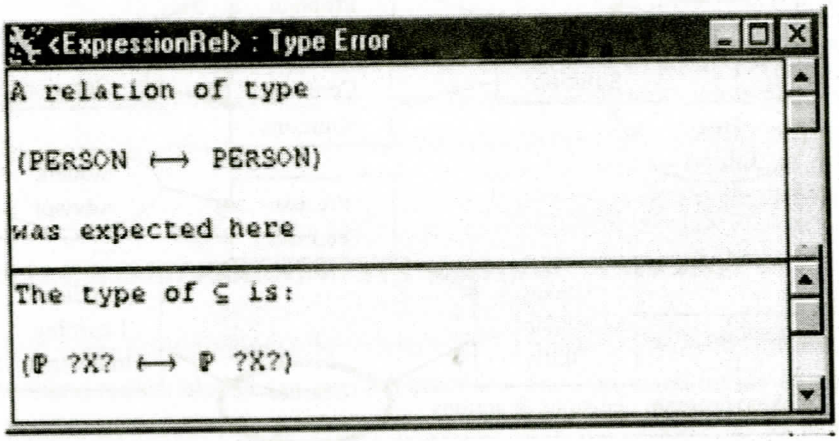
5. LITERATURE REVIEW – CBL SYSTEMS FOR FORMAL METHODS

The software systems that are primarily designed for learning formal specifications fall into two categories. The first category includes all the formal method tools that are claimed to be useful for learning. They are, in general, developed by researchers in the formal specification discipline. The second category includes systems which are principally developed for pedagogical purpose by researchers in the CBL systems discipline. Each category will be considered in turn.

a. Formal Method Tools for Teaching

There are more than hundred tools available for various formal methods. Especially about thirty of them were created for the Z notation alone. Almost all of the tools provide syntax directed editing facility so that a specification document may be created or edited easily in the environment. Formalizer (Figure 4), for example, is a syntax-directed editor, browser and type checker for Z notation. It does not have proof or refinement facility. In pedagogical point of view, at maximum it gives short error reports. Almost all the other tools meant for Z notation are similar to Formalizer.

The research literature related to the following software tools claims that they could be used for learning formal methods: Zbrowser [31], ZAL/ZED [32], VisualiZer [40], and ZTC/ZANS [20], [21]. Basically, they offer an environment for the users to actively learn a specific topic through trial and error. All of these tools demand significant tutor guidance for novices. Some of them, though primitive, do provide useful hints and feed-back



```
A relation of type

(PERSON ↔ PERSON)

was expected here
```

```
The type of ⊆ is:

(ℙ ?X? ↔ ℙ ?X?)
```

Figure 4: Feedback in Formalizer

b. CBL Systems for Formal Methods

Despite the number of tools for aiding the development of specifications, surprisingly, there appears to be no reference in the literature for CBL systems for formal methods. MEMO-II [14] and FLUTE [8] are the only existing CBL systems found to be related to this research.

As noted by its authors, MEMO-II is intended for learning programming not for formal specification. It is an education oriented programming environment, which allows users to build programs from formal specifications via interaction with the system. P. Forcheri, and M. T. Molfino [14] claim that learning to program requires modelling capabilities. A programming problem may be modelled using two approaches. One is a computational model depending on a programing paradigm, and the other is an abstract model independent of any paradigms. Learning to construct abstract models helps software practitioners to switch effortlessly between different paradigms. MEMO-II follows the second approach; and additionally, [14] claims that it also offers facilities to map this abstract model into effective implementations.

6. OUR APPROACH

Building CBL systems that facilitate 'active learning' for complex disciplines is considered challenging. The complexity, however, could be reduced by employing educational aids such as scaffolding [19]. In order to teach a complex discipline, when a similar but relatively easy domain is already known to the learner, scaffolding may be applied using a gradual transformation of learning materials from less complex to more complex disciplines. This research investigates the ways and means of designing CBL systems for certain complex domains that support active learning based on model transformation and employing scaffolding techniques in order to reduce the difficulty associated with learning such disciplines. As an exemplar, an object oriented formal notation Object-Z [37] and the semi-formal notation UML [6] are selected as the subject and support domains, respectively. A semi-automated transformation process is designed in this study in order to transform UML models to Object-Z models.

The proposed CBL system is designed based on the Constructionist learning theory [33] - which is an extension of constructivist learning theory [34]. In this approach, we encourage and support the students to create new models based on the existing models. The students will be active in making tangible models that will be meaningful to them in the current context. Scaffolding will be used to help the students in this transformation process.

7. TRANSFORMING UML TO OBJECT-Z

Both UML models and Object-Z schemas have important similarities in their semantics. They consider Class as a template that describes a set of similar objects, and also allow the use of a Class as a type. In addition, both notations use reference semantics (unlike Z notation – which use object/value semantics). S. K. Kim and D. Carrington [26] have carried out extensive research on formalizing and transforming UML models using Object-Z notation. Other significant contributions to the research in this specific area are integrating UML with Object-Z [3]; formalizing UML with Object-Z [30]; integrating Object Modelling Technique to Object-Z [12]; and defining UML constructs using Object-Z [39]. As stated previously, Dong's group [39] uses temporal logic for sequencing, but H. Miao, and L. Lui [30] use a variable 'order' to hold time related information (for sequence diagrams).

S. K. Kim and D. Carrington ([23], [24], [25] , [26], and [27]) have separately published many papers on formalizing key UML constructs, including use case diagrams, state transition diagrams, and class diagrams, and for inconsistency checking. S. K. Kim and D. Carrington [27] explain their approach for transforming class diagrams in three steps. Firstly, they formalize the syntax (and semantics) of UML constructs using Object-Z notation. Secondly, they formalize Object-Z syntax (and semantics) using both class diagrams and Object-Z itself. Thirdly, they define a mapping between these two constructs. Transforming static structure of UML constructs (particularly class diagrams) will be discussed in detail in the next section.

In addition to the static structure of UML, some research attempts to formalize the dynamic semantics of different UML constructs in Object-Z; for example, [30] for Sequence Diagrams, [11] for Object Management Technology and [26] for State Diagrams. S. K. Kim and D. Carrington [26] treat the dynamic semantics of UML (particularly for state diagrams) in two parts; denotational semantics and operational semantics. For the denotational semantics, they include relevant invariants to Object-Z state schema. For the operational semantics, they introduce 'semantic variables' as additional attributes and define the operational semantics using Object-Z operations. S. K. Kim and D. Carrington [24] claim that encapsulating the entire definition of an UML construct into a single construct (Object-Z class schema) makes their approach unique amongst others within that type. Nevertheless, the main stream researchers ([29], [28]) are now moving towards discussing the formal aspects of OMG's Model Driven Architecture (OMG 2006).

8. TRANSFORMING UML CLASS DIAGRAMS

Being the major construct in UML, the research on transforming the class diagram has been given much attention in research circles. There are various approaches to transforming key constituents of the UML class diagram to Object-Z. N. Am´alio, and F. Polack [2], in their survey, discuss a number of different approaches reported in main stream formal methods literature for formalizing the UML class diagram to Object-Z (and Z). N. Am´alio, and F. Polack [2] generally compare S. K. Kim and D. Carrington [23], approach with J. Araujo's [3], and to some extent, S. Dupuy's [13]. In addition to these approaches, two others ([10], [30]) are also included in the following discussion. In many ways, the following discussion supplements N. Am´alio, and F. Polack survey [2]. The Room-Key case study will again be used as an illustration.

a. Identity

UML uses reference semantics. The early versions of Object-Z also used value semantics. In value semantics, based on Hall's approach [17], [18] the notion of 'self' has to be defined explicitly. For example, the Key object in value semantics is given in Figure 5.

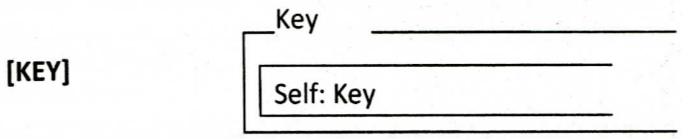


Figure 5: Value Semantics and Identity of an Object

Object-Z and UML both use reference semantics, and therefore the semantics for object identity is the same for both. An Object could refer to itself using a built-in construct called 'self' in Object-Z notation. Therefore, the object identities need not be defined as separate types. Moreover, an Object schema is considered as a template and also as a type. In this context, J.A. Hall's [18] extension/intension differentiation is not applicable.

b. Visibility List

The UML class diagram allows 3 types of accessibility: public, private or protected. There is no such protected access in Object-Z. Whilst transforming the UML classes to Object-Z schemas, none of the above researchers explained how they would treat 'protected' variables. J. Araujo's [3] notes that the visibility list in Object-Z does not differentiate attributes and methods. In addition to this, operation overloading is not possible in Object-Z, due to the format of the visibility list.

c. Class Variables

In UML, class features may be included in a Class, and they are differentiated by just under-lining. The values of class attributes are common to all the objects of that Class. There is no equivalent construct in Object-Z. None of the research mentioned above touches on this issue

d. Construction/Destruction

UML classes may explicitly include constructors and destructors. Existential constraints in an object's link may affect its creation or destruction. Object-Z assumes that the objects are always available in the environment. There is nothing to suggest in the literature as to how this conflict will be resolved, during the transformation of Classes to Object-Z.

e. Association

Class diagrams (and E-R Diagrams) include class relationships, such as association. There is no way to show association relationships in Object-Z and therefore, it should be resolved in some way, in order to represent them in Object-Z schemas. Even in program code, there is no way to explicitly include association. Basically, there are three approaches discussed in the literature: local, recursive and central.

In the first approach, an object identity (or a collection of object identities – depending on the multiplicity constraints) of the participating Classes are included as attributes into the Classes of the opposite sides of the association link. Usually, role names will be used to name those attributes. Depending on the navigational direction, if known, one side of the Class may not need to keep the link, and therefore it may not include the relevant identities as attributes. This approach is related to designing a database from E-R diagrams, where the corresponding primary keys (instead of identities) are included in the relevant tables. Duke's ([10], p. 42.) 'local view' is a good example of this approach. R. Duke, and G. Rose [10] include a class schema called 'KeySystem' which consists of a mixture of properties of the relevant control environment (in operation schemas) and the corresponding association (in state schema). It can be noted that the 'KeySystem' is a class schema but it will only have one instance.

Figure 6 illustrates the 'local' approach. For every room in the system key there is at least a key. The features related to control environment (here the operation 'insertKey') may be separated. The operation 'supplyID' will just return the identity of an object 'self' - refer to [10]. The parallel operators are used to pass parameters. If the navigation is on both sides, an attribute called keys may be included in the Room class. The 'type' of 'keys' will be a collection of identities of Key objects, which represent the collection of all the keys that open a particular room.

```
Key
  keyID: KEYID
  room: Room

Room
  roomID : ROOMID
  key: Key

KeyRoomASSandCTRL
  allRooms: ℙ Room
  allKeys: ℙ Key
  ----------------------------------------
  ∀ r : allRooms.∃ key∈ allKeys. key.room = r
  ∀ k : allKeys.∪ k.room⊆ allRooms

  insertKey =[r? : allRooms, k? : allkeys] . r? . supplyID || k? . open
```

Figure 6 : Formalizing Association Local

9. THE PROPOSED TRANSFORMATION PROCESS

In order to illustrate the process, here we considered a simple class diagram with no inheritance hierarchy. There are only association relationships. Assume use cases are ranked based on their importance.

Step-1: REPEAT steps 2 through 14, UNTIL there are no more important use cases
Step-2: Select the next important use case (initially select the most important use case)
Step-3: Consider the corresponding sequence diagram
Step-4: REPEAT steps 5 through 10, UNTIL there are no more clusters in the sequence diagram
Step-5 Select the next important cluster (initially select the most important cluster)
Step-6: REPEAT steps 7 through 9, UNTIL there are no more independent operations in the cluster
Step-7: Select the next independent operation in the cluster (initially select the most important operation)
Step-8: Generate the relevant class schema, if not generated before
Step-8.1: Complete the state schema (interact with user, if necessary)
Step-8.1.1: Specify attributes and types
Step-8.1.2: Specify dependent attributes
Step-8.1.2: Specify class invariant

Step-8.2: Complete the INIT schema (interact with user, if necessary)

Step- 8.2.1: Specify initial values for attributes

Step-9: Specify the operation schema (interact with user, if necessary)

Step-9.1: REPEAT UNTIL there are no sub-operations

Step-9.1.1: Select the next independent sub-operation

Step-9.1.2: Check the name, delta list, input and output variables

Step-9.1.3: Complete the declaration part

Step-9.1.4: Complete the prediction part (this is challenging)

Step-9.2 : Complete the operation schema

Step-10 : REPEAT Steps 11 through 13, UNTIL there are no more operations in the cluster.

Step-11: Incrementally select the next important operation that depends only on the operations that are already considered

Step-12: REPEATUNTIL there are no more associations left\

Step-12.1: Select the next important association

Step-12.2: Generate the class schema at the other end (similar to Step-8)

Step-12.3: Include relevant attributes into both state schemas

Step-12.4: Recursively include relevant invariants into both state schemas

Step-13: Specify the operation specification (similar to Step- 9)

Step 14: Complete Operation Schemas

Step-15: Consider alternative sequence diagrams (if any).

Even if there is no such diagram, alternate trivial use cases may be available. Check the use case specification and the relevant operation specification. If so, modify the schemas accordingly.

Step-16: Refine the schema

10. PROTOTYPE FOR CBL SYSTEM

Software prototypes may be developed for different purposes, such as elicitating requirements, evaluating interface design and for inspecting certain functional features [38]. The prototype in this research is developed to serve two purposes; firstly, to determine whether the proposed ideas in this research are technically feasible and secondly, to verify that the implemented artefacts are practically useful. Due to the time constraint, the interface design of the prototype is not a major focus.

A set of simple case studies are used in the CBL system. The case studies are ranked based on their complexity. Initially, the case studies are given in English and the corresponding UML models also available to the students. The students need to convert the UML models to Object-Z schemas. The appropriate level of help will be provided by the semi-automated translation system. As a student progresses, the level of direct support will be reduced, and eventually they will be able to create formal schemas with the help of semi-automated translation system only.

11. EVALUATION

The prototype was used to evaluate the key research proposals made in this research. In particular, two hypotheses are considered important and related. The first hypothesis is about the effectiveness of the overall strategy proposed for designing CBL systems for complex domains using model transformation. For practical reasons a pre-experiment design is used for objective evaluation.

A group of ten students were given a pre-test, and then asked to learn a concept using the prototype. Later, a post-test was given. The performance of the students on both tests was statistically analysed. The Student-t statistics for paired-samples was used at 0.05 significant level. The test reveals that the post-test scores are significantly higher than the pre-test scores. Another test confirmed that both the pre and post-tests were of the same level of difficulty. The meta-analysis validated the test results. The effect size of the test was greater than 2 and the power is greater than 8. Both values are reasonable compared to the similar research reported in the literature.

12. CONCLUSION

Some disciplines are inherently complex and therefore challenging to learn. A suitable instructional strategy is essential to realize this goal. This research shows that constructionism can be effectively used for designing CBL systems for teaching Object-Z. In this approach, students are encouraged to translate specifications in UML to Object-Z notation. Transforming UML models to Object-Z models is easier than creating Object-Z models from scratch. Scaffolding is used to help the students in this complex process. To the knowledge of the authors, there is no step-by-step semi-automated methodology available to facilitate this translation process. This research illustrated that the proposed step-by-step methodology to transform UML Models to Object-Z specification is suitable for pedagogical purposes. This research may be extended to cover complex UML models with inheritance hierarchies.

References

[1] ACM-SE (2004), "Curriculum guidelines for undergraduate degree programs in Software Engineering," A volume of the Computing Curricula Series: 59.

[2] N. Am´alio, and F. Polack, (2003) "Analysis and Comparison of Formalization Approaches of UML class constructs in Z and Object-Z," *ZB 20003,* Turku, Finland, LNCS, Springer.

[3] J. Araujo and P. Sawyer, (1996) "Metamorphosis: An integrated object-oriented requirement analysis and specification method," *Department of Computing,* University of Lancaster, UK.

[4] R. Barden, S. Stepney, and D. Cooper, (1994) "Z in Practice," *BCS Practitioner Series.* Prentice-Hall.

[5] F. Bennett, (1999) "Computers as Tutors: Solving the Crisis in Education." *Educational Technology & Society 2(4)* 1999, ISSN 1436-4522.

[6] G. Booch, J. Rumbaugh and I. Jacobson, (1999) "The UML User Guide," Addison-Wesley.

[7] A. Brobert, (1999) "Learners as Knowledge Workers: Some Implications," *Frontiers in Education, FIE'99,* Puerto Rico, San Juan.

[8] V. Devedzic, J. Debenham, and D. Popovic,(2000) "Teaching Formal Languages by an Intelligent Tutoring System," *Education Technology & Society* 3(2).

[9] R. Duke , P. King , G. Rose, and G. Smith, (1991) "The Object-Z specification language version 1," *Technical Report*, Software Verification Centre, Department of Computer Science, University of Queensland, 1991.

[10] R. Duke, and G. Rose, (2000) "Formal Object-Oriented Specification Using Object-Z," London, MacMillian.

[11] S. Dupuy, Y. Ledru, and M.Chabre_Peccoud, (1998) "Translating the OMT Dynamic Model into Object-Z," *ZUM' 98.*

[12] S. Dupuy, Y. Ledru, and M.Chabre_Peccoud, (2000) "Integrating OMT and object-Z." *BCS FACS/ EROS ROOM Workshop, technical report GR/K67311-2*. A. Evans and K. Lano, Imperial College, London.

[13] S. Dupuy, Y. Ledru, and M.Chabre_Peccoud, (2000) "An overview of RoZ- a tool for integrating UML and Z specifications." *12th Conference on Advanced information Systems Engineering (CAiSE'2000),* Stock-holm, Sweden.

[14] P. Forcheri, and M. T. Molfino, (1994) "Software Tools for the Learning of Programming: A proposal." *Computers Education* 23(4): 269-278.

[15] M. Fowler, (1998) "UML Distilled; Applying the standard modelling language," *Addison Wesley* Longman Inc.

[16] H. Garavel, and G. Susanne, (2013) " Formal Methods for Safe and Secure Computer Systems," *BSI Study 875,* Federal Office of Information Security, Bonn, Germany, Retrieved from https://www.bsi.bund.de/Shared Docs/ Downloads /DE/BSI/Publikationen/Studien formal_methods_study_875 formal_methods_study_875. pdf?__blob=publication File.

[17] J.A. Hall, (1990) " Using Z as a specification calculus for object-oriented analysis." *VDM'90- VDM and Z,* B. Dines and C. Hoare, A, R, Springer Verlag.

[18] J.A. Hall, (1994) "Specifying and Interpreting Class Hierarchies in Z." *ZUM'94,* Springer Verlag.

[19] K. Hogan, and M. E. Pressley,(1997) "Scaffolding Student Learning: Instructional Approaches and Issues." *The University of Albany, State University of New York,* BROOKLINE.

[20] X. Jia,(1995) "A Tutorial of ZANS — A Z Animation System," *Chicago, DePaul University.*

[21] X. Jia,(1995) "ZTC: A Type Checker for Z, User's Guide." *Chicago, DePaul University.*

[22] H. Kanuka and T. Anderson, (1999)"Using Constructivism in Technology-Mediated Learning." *Radical Pedagogy 1(2).*

[23] S. K. Kim and D. Carrington, "Formalizing the UML class diagram using Object-Z." *UML1999, LNCS1723* (Springer: Berlin): pp. 83-98.

[24] S. K. Kim and D. Carrington, (2000) "A Formal Mapping between UML Models and Object-Z Specifications." *ZB2000, LNCS1878* (Springer: Berlin): pp. 2-21.

[25] S. K. Kim and D. Carrington, (2000) "UML Metamodel Formalization with Object-Z: the State Machine Package," *Technical Report 00-29, SVRC.* The University of Queensland, Australia.

[26] S. K. Kim and D. Carrington, (2002) "A Formal Metamodeling Approach to transformation between the UML State Machine and Object-Z." *In International*

Conference on Formal Engineering Methods (ICFEM 2002), Lecture Notes in Computer Science. Springer- Verlag.

[27] S. K. Kim and D. Carrington,(2005) "An MDA Approach towards Integrating Formal and Informal Modeling Languages." *FM 2005, LNCS 2495* . Springer-Verlag, UK.

[28] D. Laurillard, (2008) "E-Learning in Higher Education. Changing higher education: The development of learning and teaching" (in press). P. Aswin. London, Routledge: 71-86.

[29] S.J. Mellor and M. J. Balcer, (2002) "Executable UML:a foundation for model-driven architecture, " Stephen J. Mellor, Marc J. Balcer. Boston, *Addison-Wesley.*

[30] H. Miao, L. Lui, and L. Li, (2002) "Formalizing UML Models with Object-Z." *International Conference on Formal Engineering Methods (ICFEM 2002),* Lecture Notes in Computer Science. Springer-Verlag.

[31] L. Mikusiak, V. Vojtek, J. Hasaralejeko, and J. Hanzelova, (1995) "Z Browser - Tool for Visualization of Z Specifications." *ZUM'95 - 9th International Conference of Z Users*, Springer- Verlag, 1995.

[32] I. Morrey, J. Siddiqi, G. Buckberry, and R. Hibberd, (1993) "Use of a specification construction and animation tool to teach formal methods." IEEE COMPSAC 93, The Seventeenth Annual International Computer Software and Applications Conference, Phoenix, Arizona, USA.

[33] S. Papert, I. Harel, (1991) "Situating Constructionism. Constructionism," Ablex Publishing Corporation: 193-206. Retrieved from http://www.papert.org/articles/SituatingConstructionism.

[34] J. Piaget, (1968) "Six Psychological Studies." New York, Vintage Books.

[35] V. J. Shute, and R. Glaser, (1990) "A large-scale evaluation of an intelligent discovery world: Smithtown." Interactive Learning Environments1: pp. 51-76.

[36] V. J. Shute, and J. Psotka, (1995) "Intelligent Tutoring Systems: Past, Present, and Future." Handbook of Research on Educational Communications and Technology. H. Jonassen, D., Scholastic Publications, 1995.

[37] G. Smith, (2000) "The Object-Z Specification Language," Kluwer Academic Publishers.

[38] I. Sommerville, (2001) "Software Engineering." Harlow, England, Addison-Wesley.

[39] J. Sun, J. S. Dong, J.liu, and H. Wank, (2001) "Z family on the web with their UML photos." TRA1-01. Singapore, School of Computing, National University of Singapore, 2001.

[40] C. N. Yap, (1999) "Visual-Z: a Methodology and Environment for Developing Visual Formal Z Specifications." Ph.D. thesis.