Empirical Evaluation of Modeling Languages Using Multi-Lift System Case Study

Abbas Rasoolzadegan, Ahmad Abdollahzadeh
Intelligent Systems Laboratory (http://ce.aut.ac.ir/islab)
Information Technology and Computer Engineering Faculty
Amirkabir University of Technology (Tehran Polytechnic)

Abstract - *This paper empirically investigates the advantages and limitations of modeling languages by specifying the multi-lift system as a non-trivial case study. The multi-lift system is a suitable test bed to demonstrate the expressive power of modeling languages in specifying such a concurrent, reactive, and complex system. English, UML, and Object-Z have been chosen, respectively, as informal, semi-formal, and formal modeling languages to specify the software requirements of the multi-lift system. These modeling languages are then compared and evaluated based on their likelihood to produce a specification with some defined characteristics. The conclusion is that informal languages such as English cannot be used to produce software models, because they are prone to ambiguity. Each of the formal and semi-formal languages has some unique advantages and limitations. Semi-formal models should be supplemented with formal ones to produce high quality software.*

Keywords: Modeling languages, Object-Z, UML.

1 Introduction

A model is the mathematical meaning of a description of a domain, or a prescription of requirements, or a specification of software, i.e., is the meaning of a specification of some universe of discourse. Modeling is the process of identifying appropriate phenomena and concepts and of choosing appropriate abstractions in order to construct a set of models which reflect appropriately on the universe of discourse being modeled. In software engineering we create successions of models: $domain \rightarrow requirements \rightarrow software\ designs$. Thus the combined universes of domain engineering, requirements engineering, and software design constitute the universe of discourse of software engineering [4]. Model and modeling play a crucial role in the software development process. In software engineering, models are used to describe both the problem (requirements) and the solution (design) in order to gain a better understanding of the issues involved. Once a model has been constructed it can be analyzed to uncover flaws and expose fundamental issues [5]. This role of models cannot possibly be assumed by code. The idea is not new, but there is a recent trend towards more use of models in mainstream circles of software engineering.

Models must be written using some language or notation. A language consists of syntax and semantics. Software models should be precise, be abstract to avoid bias towards any specific implementation, and be written in terms of user-observable phenomena. Programming languages cannot be used to write them, because programs describe one way to meet the needs of a product, but not necessarily the actual requirements of stakeholders. This means that we need specialized notations to model software systems. Software modeling languages are divided into three big groups: informal, semi-formal, and formal. Informal group includes all natural languages (such as English) which are prone to ambiguity. Semi-formal languages (such as UML) have a formal syntax, which is usually diagrammatic, but no formal semantics. Formal languages (such as Object-Z) are usually textual and have a formal or mathematical syntax and semantics.

In this paper, advantages and limitations of modeling languages are investigated empirically by specifying the multi-lift system using English, UML, and Object-Z. Multi-lift system is a commonly used test bed to demonstrate the expressive power of various modeling languages in specifying concurrent reactive systems. It is not a trivial case study because of the complexity caused by inherent concurrent interaction in the system. The multi-lift system includes parallel, distributed, embedded, and real-time software. In a real-time software system the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced [1]. An embedded computer system is a system that uses a computer as a component, but whose prime function is not that of a computer. Parallel and distributed software systems have their own complex features such as the concurrent interactions between various system components, the

reactive nature of the systems, and various message passing schemes between system components.

The conclusion of the empirical evaluation of modeling languages is that each modeling language has some unique advantages and limitations. Using only one of them as the sole modeling approach will not guarantee high quality software. Literature review also confirmed this conclusion [4], [8]. As a result, we can supplement semi-formal models with formal ones to introduce precision in the development process and to sweeten formal languages usage. Such a combination leads to develop the software with the desired quality [11-14]. Proposing a new approach to integrate semi-formal and formal models is left for future work.

The rest of this paper is organized as follows: Section 2 presents informal specification of requirements and expected properties of given multi-lift system. This system is rather an "ideal" one in which some of the technical corners are cut. In section 3 formal specifications of the system's requirements is presented using Object-Z. Section 4 specifies these requirements visually in UML. Finally, Section 5 draws conclusions and discusses future work.

2 Informal Specification

A lift system for a multi-floor building consists of multiple elevators. The elevators are supposed to be used in a building having floors numbered from 1 to *MaxFloor*. There are two direction buttons on each floor (except the top floor and the lobby) for the passengers to call for going up and down. There is only one 'down' button at the top floor and one 'up' button in the lobby. Inside each elevator there is a panel of floor buttons each of which indicates a destination floor. Door is one of important parts of a lift system. When an elevator car stops in a particular floor, its door is opened for passenger to come out and come in to the car. Arrival sensor is used in every floor for detecting the elevator. Once an elevator reaches a particular floor, the floor's arrival sensor detects the elevator and stops the car. Any button can be pushed at any time. Any external floor button is turned on from the time it is pushed until an elevator with the same travel direction stops at the floor and opens the door. Any internal 'on' button of a lift is turned off when the elevator visits the corresponding floor.

The multi-lift system that is defined in this paper has the basic functions that all elevator systems have such as moving up and down, open and close doors, and of course, pick up passengers. When a car stops at a floor, the door is opened and the car lantern, which indicates the current move direction of the car, is illuminated for notifying the passengers of the current move direction of the car. In order to certify system safety, the emergency brake will be triggered and the car will be forced to stop under any unsafe conditions. The lift system is responsible for controlling the lifts. Passengers interact with the lift system by pressing buttons on the individual floors or on the control panel inside the elevators. Initially, all lifts stay on the ground floor of the building. If a passenger enters a lift and presses the button for the *k-th* floor the request information is sent to the system. Then the lift moves up to the *k-th* floor. When the lift arrives at the destination, it opens the door for a certain period *M* seconds of time, then closes it again, and becomes idle. Moreover, when a passenger on *m-th* floor calls a lift by pressing the up or down button, the most suitable lift is moved to the *m-th* floor by the lift system and the door is opened on arrival. The passenger may request to go to a particular floor by pressing the corresponding button on the control panel inside the lift. If there is no passenger interaction on the control panel within *M* seconds, then the lift will close the door and go idle at that floor.

3 Formal Specification

Formal specification of the multi-lift system is presented using Object-Z [2] which is a state-based language that favors a model-oriented style of specification. It is an extension to Z to facilitate the structuring of a model in an Object Oriented (OO) style. It provides the class schema as a structuring mechanism (a collection of an internal state schema and operation schemas) and support for OO notions such as object, inheritance, and polymorphism. Object-Z is the most successful OO extension to Z among the ones that emerged in early 1990s [2].

In this section, just the main components of the lift system are presented due to the space limitations. More information about the details of the formal specification has been elaborated in [16]

LiftButton

$\upharpoonright(num, state, \text{INIT}, Press, TurnOff, Blink)$

$Button[\textbf{redef}\ \ Press]$

$num : \mathbb{N}$

$state : ButtonStatus \mid Blink$

$1 \leq num \leq \#floors$

Press

$\Delta(state)$

$state = Off \land\ num \in liftb.floorsServiced \land\ num \neq liftb.current$

$state' = On \land\ liftb.AddIntReqToIntList \bullet [floorNum? = num]$

Blink

$\Delta(state)$

$state' = Blink$

FloorButton

$\upharpoonright(direction, state, \text{INIT}, Press, TurnOff)$

$Button[\textbf{redef}\ \ Press]$

$direction : MotionDirection$

$floor : Floor$

$selt \in floor.buttons$

Press

$\Delta(statc)$

$state = Off \land state' = On$
$floor.Press \bullet [md? = direction]$

Passenger

$\upharpoonright()$

$id : \mathbb{N}$

Press

$button? :\downarrow Button$

$button?.Press$

TurnOff

$button? :\downarrow Button$

$button?.TurnOff$

Floor

$\upharpoonright(\textsc{Init}, GetNum, Press, TurnOff)$

$num : \mathbb{N}$
$fmdi : \mathrm{seq}\ FMotionDirectionIndicator_{\textcircled{C}}$
$flpi : \mathrm{seq}\ FLiftPositionIndicator_{\textcircled{C}}$
$buttons : \mathbb{P}\ FloorButton_{\textcircled{C}}$
$centc : CentralController$
$building : Building$

$1 \le num \le \#floors$
$1 \le \#buttons \le 2$
$\#fmdi = \#lifts$
$\#flpi = \#lifts$
$\forall\ lpi : \mathrm{ran}\ flpi \bullet lpi.flr = self$
$\forall\ mdi : \mathrm{ran}\ fmdi \bullet mdi.fr = self$
$\forall\ b : buttons \bullet b.floor = self$
$self \in centc.floors$
$self \in building.floorsb$

Init

$\forall\ m : \mathrm{ran}\ fmdi \bullet m.\textsc{Init}$
$\forall\ l : \mathrm{ran}\ flpi \bullet l.\textsc{Init}$
$\forall\ b : buttons \bullet b.\textsc{Init}$

GetNum

$num! : \mathbb{N}$

Press

$md? : MotionDirection$

$centc.PutReqToQ \bullet [extReq? = (num, md?)]$

TurnOff

$md? : MotionDirection$

$\exists\ button \in buttons \bullet button.direction = md? \Rightarrow button.TurnOff$

LocalController

$\upharpoonright(ds, \textsc{Init}, OpenDoor, ClosеDoor, UpdateDispatchingStrategy, Move, GetId, GetLiftInfo, AddExtReqToExtList, DeleteExtReqFromExtList, UpdateFloorsServiced, UpdateStandbyFloor, Stop, LiftMoved)$

$id : \mathbb{N}$
$liftInfo$
$ds : DispatchingStrategy$
$centcont : CentralController$
$extRequestList : Requests$
$standbyFloor : \mathbb{N}$
$lift : Lift$

$id = lift.num$
$self \in centcont.loccents$
$self = lift.lc$

Init

$extRequestList = \varnothing$
$standbyFloor = 1$

OpenDoor

$lift.OpenDoor$

CloseDoor

$lift.CloseDoor$

Move

$ds = Closed \land \neg(lift.IsWeightOver)$
$lift.md = Up \rightarrow$
$\quad (\exists\ r : extRequestList \bullet r.1 > lift.current) \lor$
$\quad (\exists\ i : \mathrm{dom}\ lift.fbuts \bullet (i > current \land fbuts(i).state = On)) \Rightarrow$
$\quad lift.Move \bullet [direction? = Up]$
$lift.md = Down \Rightarrow$
$\quad (\exists\ r : extRequestList \bullet r.1 < lift.current) \lor$
$\quad (\exists\ i : \mathrm{dom}\ lift.fbuts \bullet (i < current \land fbuts(i).state = On)) \Rightarrow$
$\quad lift.Move \bullet [direction? = Down]$
$[extRequestList = \varnothing \land lift.intRequestList = \varnothing] \bullet$
$\quad lift.fbuts(lift.standbyFloor).state = On$

Stop

$[lift.fbuts(lift.current).state = On] \bullet$
$\quad lift.DeleteIntReqFromIntList \bullet [floorNum? = lift.current] \land$
$\quad lift.fbuts(lift.current).TurnOff$
$\lor$
$[\exists\ r \in extRequestList \bullet (r.1, r.2) = (lift.current, lift.md) \bullet$
$\quad DeleteExtReqFromExtList \bullet [extReq? = (lift.current, lift.md)] \land$
$\quad centcont.TurnOffFloorBut \bullet [(floorNum?, md?) = (lift.current, lift.md)]$
$\lor$
$[lift.md = Up \land \exists\ r \in extRequestList \bullet$
$(r.1 = lift.current \land r.2 \neq lift.md) \land \nexists\ b : lift.fbuts \bullet$
$(b.num > lift.current \land b.state = On) \land \nexists\ r \in extRequestList \bullet$
$r.1 > lift.current] \bullet$
$\quad lift.MotionDirectionChanged \land$
$\quad DeleteExtReqFromExtList \bullet [extReq? = (lift.current, lift.md)] \land$
$\quad centcont.TurnOffFloorBut \bullet [(floorNum?, md?) = (lift.current, lift.md)]$
$\lor$
$[lift.md = Down \land \exists\ r \in extRequestList \bullet$
$(r.1 = lift.current \land r.2 \neq lift.md) \land \nexists\ b : lift.fbuts \bullet$
$(b.num < lift.current \land b.state = On) \land \nexists\ r \in extRequestList \bullet$
$r.1 < lift.current] \bullet$
$\quad lift.MotionDirectionChanged \land$
$\quad DeleteExtReqFromExtList \bullet [extReq? = (lift.current, lift.md)] \land$
$\quad centcont.TurnOffFloorBut \bullet [(floorNum?, md?) = (lift.current, lift.md)]$
$lift.stop \land OpenDoor$

UpdateDispatchingStrategy

$\Delta(ds)$
$newDS? : DispatchingStrategy$

$ds' = newDS$

GetLiftInfo

$liftInfo!$

AddExtReqToExtList

$\Delta(extRequestList)$
$extReq? : Requests$

$extRequestList' = extReq \cup extRequestList \land$
$lift.fbuts(floorNum?).state = Blink$

DeleteExtReqFromExtList

$\Delta(extRequestList)$
$extReq? : Requests$

$extRequestList' = extRequestList \setminus \{extReq?\}$

UpdateFloorsServiced

$newSet? : \mathbb{P}\,\mathbb{N}$

$lift.floorsServiced = newSet?$

UpdateStandbyFloor

$floorNum? : \mathbb{N}$

$lift.standbyFloor = floorNum?$

GetId

$id! : \mathbb{N}$

LiftMoved

$centcont.UpdateFloorsIndicators \bullet$
$[(liftNum?, floorNum?, md?) = (lift.num, lift.current, lift.md)]$

CentralController

$\upharpoonright(INIT, PutReqToQ, GetReqFromQ, ResponseToExtReq, TurnOffFloorBut, UpdateFloorsIndicators)$

$locconts : \mathrm{seq}\, LocalController$
$reqQ : RequestQueue$
$strategy :\downarrow ExternalRequestAllocator_{\circledS}$
$tmcc : TrafficManager_{\circledcirc}$
$csg : ControlStrategyGenerator_{\circledcirc}$
$floors : \mathrm{seq}\, Floor$
$buildingc : Building$

$\#locconts = \#lifts$
$\forall\, l : \mathrm{ran}\, locconts \bullet l.centcont = self$
$csg.ceco = self$
$strategy.centralcontroller = self$
$tmcc.cctrl = self$
$\forall\, f : \mathrm{ran}\, floors \bullet f.centc = self$
$buildingc.cc = self$

INIT

$csg.INIT$

PutReqToQ

$extReq? : Requests$

$reqQ.Join \bullet [item? = extReq?]$

$ResponseToExtReq \mathrel{\widehat=} [reqQ.size \neq 0] \bullet$
$(reqQ.Leave \wedge\ strategy.AllocateExternalRequest) \bullet$
$([item! = extReq?] \wedge\ locconts(liftNum!).AddExtReqToExtList) \bullet$
$([extReq? = item!] \wedge\ reqQ.Remove) \bullet [item? = item!]$

TurnOffFloorBut

$floorNum? : \mathbb{N}$
$md? : MotionDirectiom$

$floors(floorNum?).TurnOff(md?)$

UpdateFloorsIndicators

$liftNum? : \mathbb{N}$
$floorNum? : \mathbb{N}$
$md? : MotionDirectiom$

$\forall\, f : \mathrm{ran}\, floors \bullet$
$(f.fmdi(liftNum?).state = Off \Rightarrow f.fmdi(liftNum?).TurnOn)$
$\wedge\, f.fmdi(liftNum?).content = md? \wedge$
$f.flpi(liftNum?).content = floorNum?$

4 Semi-formal Specification

In this section the multi-lift system is specified and designed using UML which is an industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as other non-software systems. UML simplifies the complex process of software design, making a "blueprint" for construction, and is now the standard notation for software architecture. The use of the UML notation for grasping the static structure of the system and for highlighting the system behavior greatly assists the understanding of the formal model. This effectively outlines an integrated approach to modeling the parallel, complex, and distributed software systems. UML provides both the structural views and behavioral views of the system. In this section, a brief UML documentation package is given based on informal and formal models presented before. More detailed information has been explained in [16].

According to the requirements document of our study, the use case diagram of the multi-lift system includes six use cases as shown in Figure 1.

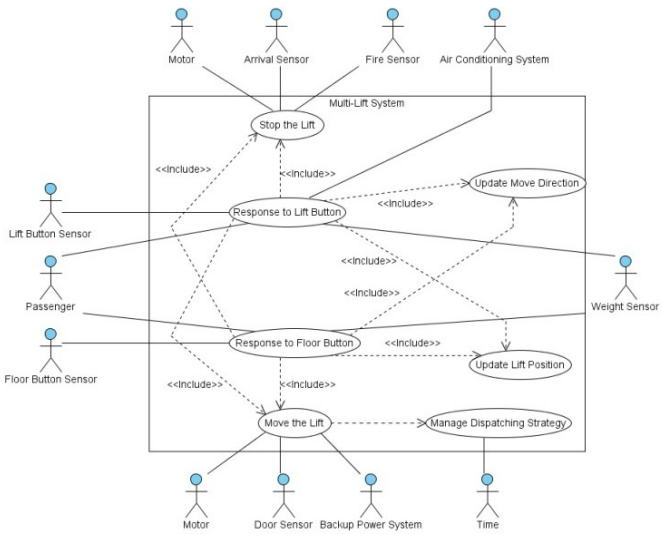


Figure1. Use case diagram of multi-lift system

As an instance, Figure 2 shows the sequence diagram illustrating the scenarios of the use case "Response to Floor Button". Figure 3 illustrates one of the activity diagrams corresponding to the narrative (various scenarios) of the use case "Response to Lift Button".

As the main characteristic states of the lift are the move direction and the door status, in Figure 4, a product of the state spaces of *MoveDirection* and *DoorStatus* forms the state diagram of the lift system. Transitions between states are labeled with Object-Z operations with instantiated preconditions. The instantiated preconditions for each operation are also partitioned and labeled so that they can be presented in a reusable fashion.

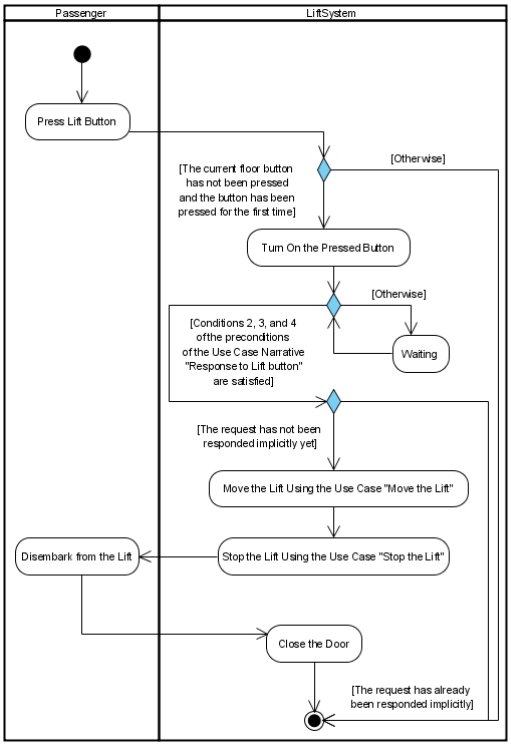


Figure3. Activity diagram corresponding to the narrative of the use case "Response to Lift Button"

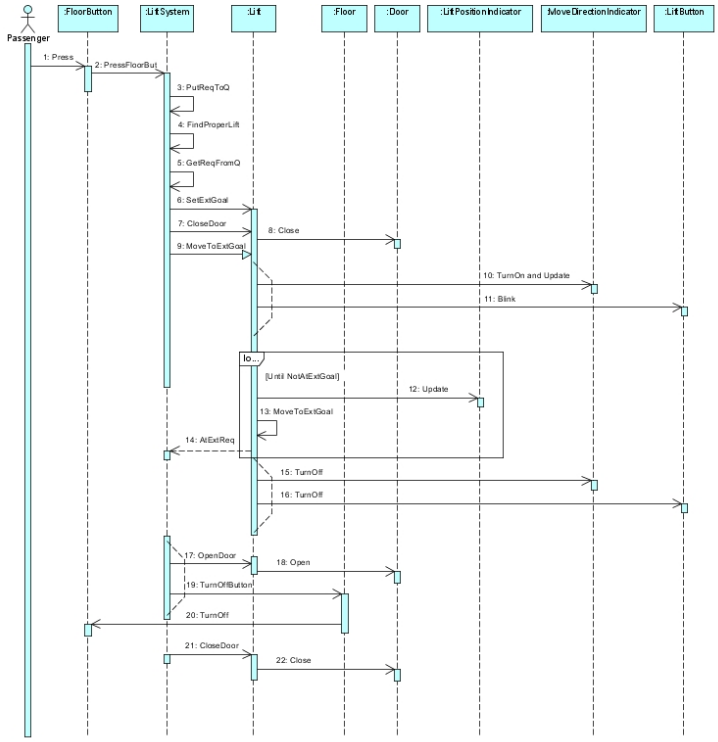


Table1. Comparison of the mentioned modeling languages

Criteria \ Modeling Language	English	UML	Object-Z
Produce readable models	2	3	1
Requires least amount of technical skills	3	2	1
Facilitates development process	1	2	3
Produce testable models	1	2	3
Produce specification decoupled from design	1	2	3
Has least potential for ambiguity	1	2	3
Helps to know when to stop each development phase	1	2	3
Has tool support	1	2	2
Produces maintainable models	1	2	2
Encourage rigorous understanding of requirements	1	2	3
Helps check internal data consistency	1	3	2
Helps validate requirements	1	3	2
Helps verify requirements	1	2	3
Subject to automatic analysis	1	2	3
Helps to know whether all requirements are captured	1	2	3
Total scores for all criteria	18	33	37

Message No. 1:

$$PressButton \mathrel{\widehat{=}} PressUp \vee PressDown$$

Message No. 10:

$$MoveDirectionIndicator.Update \mathrel{\widehat{=}} MoveDirectionIndicator.TurnUp \vee MoveDirectionIndicator.TurnDown$$

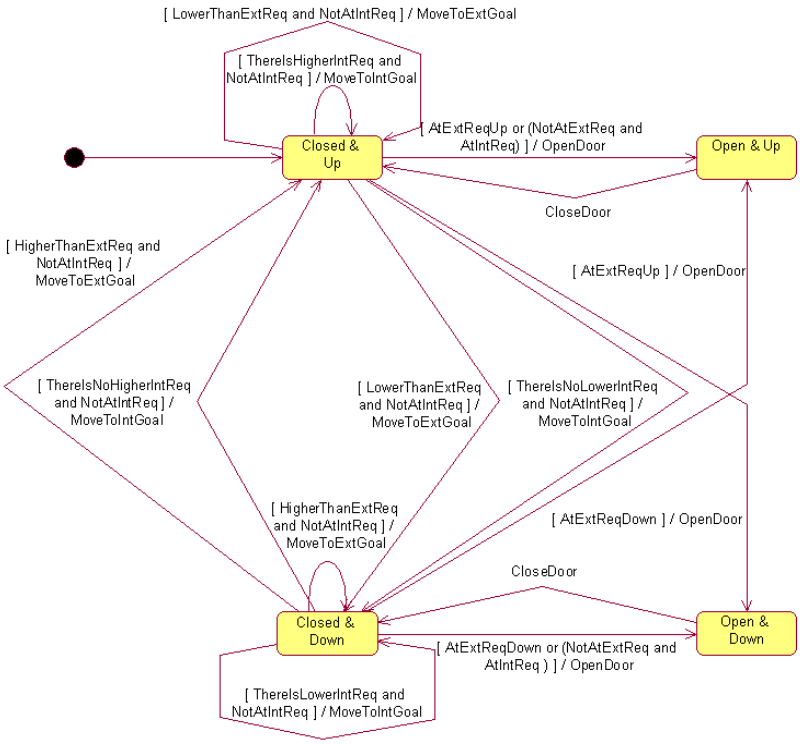
Message No. 12:

$$LiftPositionIndicator.Update \mathrel{\widehat{=}} LiftPositionIndicator.Increase \vee LiftPositionIndicator.Decrease$$

Message No. 19:

$$TurnOffButton \mathrel{\widehat{=}} DownOff \vee UpOff$$

Figure2. Sequence diagram corresponding to the narrative of the use case "Response to Floor Button"



$$AtExtReqDown == \exists g : extGoal \bullet current - first(g) \wedge second(g) - Down \qquad NotAtExtReq == \nexists g : extGoal \bullet current = first(g)$$

$$AtExtReqUp == \exists g : extGoal \bullet current = first(g) \wedge second(g) = Up \qquad LowerThanExtReq == \exists g : extGoal \bullet current < first(g)$$

$$higherThanExtReq == \exists g : extGoal \bullet current > first(g) \qquad NotAtIntReq == fbuts(current).state \neq On$$

$$ThereIsHigherIntReq == \exists i : \mathrm{dom}\, fbuts \bullet (i > current \wedge fbuts(i).state = On)$$

$$AtIntReq == fbuts(current).state = On \qquad ThereIsNoHigherIntReq = \neg ThereIsHigherIntReq$$

$$ThereIsLowerIntReq == \exists i : \mathrm{dom}\, fbuts \bullet (i < current \wedge fbuts(i).state = On)$$

$$ThereIsNoLowerIntReq = \neg ThereIsLowerIntReq$$

Figure4. State diagram of the Lift System

5 Discussion

Table 1 summarizes the relative merits of each of the three modeling languages (English, UML, and Objec-Z) in producing useful artifacts to support various phases of the development cycle according to our empirical experiment on specifying the multi-lift system. A point system ranges from 1 (worst) to 3 (best) whereby a score of 3 for one of the three languages indicates that it best addresses the criteria of merit compared to the other two languages. A language is better than another language for a particular basis of comparison if the former scores a higher point than the latter. Two languages scoring the same number of points for a given criteria of merit imply that none clearly addresses the criteria better than the other. Even though a language may score the same number of points for two different bases of comparison, it does not imply that the language performs equally well in both areas of comparison. For each basis of comparison, the languages are rated relative to one another.

The actual values of the total scores for each language give us some intuitive ideas on how the languages rated comparatively. Overall, the English as an informal language is the least effective tool to produce high quality models for specification and design. The Object-Z as a formal language seems to offer the best alternative. What has been evident throughout the discussion on the relative merits of the various modeling languages is that none of them is ideal in all respects. In other words, each language has some unique advantages and limitations. Using only one of them as the sole modeling approach leads not to a high quality software

The results obtained by this empirical experiment confirm the discussions and results raised from the literature review. According to the literature, formal modeling languages such as Object-Z take a precise approach to software development, delivering reliable software; however, in addition to high cost involvements, they require a level of expertise that is not common in commercial development communities. These limitations lead to decreasing their practicality [4]. Semi-formal modeling languages such as UML that are widely used in practical large-scale software development do not take a rigorous approach to reliability of software in development [8]. Moreover, the increasing importance of developing high quality software beside increasing necessity of *Requirements Engineering* [4] as well as need for further abstraction leads to increasing use of various types of models [4-5]. Models are used at different phases of software development, ranging from requirements to detailed design for specification, analysis, validation (rather than simulation or prototyping [6]), and verification of customer requirements, problems, and solutions. This is the idea behind *Model-Driven Software Engineering* (MDSE), an approach that advocates models, rather than code, as the primary artifacts of software developments. *Model transformation* and *model refinement* have key roles in MDSE [7].

Investigation of advantages and limitations of semi-formal and formal modeling languages, theoretically (by surveying the literature) and empirically (by defining suitable case studies), shows that combination of various types of modeling languages ensures achieving high quality (such as reliable, yet flexible and reusable) software [9-10], [15]. Regarding to the above-mentioned conclusion, we are going to propose a new approach to integrate Object-Z and UML notations using a bidirectional and precise transformation. Accordingly, software is initially modeled using Object-Z. These formal models, along with formal refinement ensure correctness. With an iterative and evolutionary approach, formal models are transformed to UML. Applying design patterns on visualized models improves some other aspects of quality such as flexibility, reusability, and scalability. The improved models are then re-formalized. By specifying the multi-lift system manually in this paper, we provide a suitable test bed to empirically assess the systematic approach that will be proposed in the future. The new approach will be applied on the existing formal specification produced by Object-Z. Then the output will be compared with the current visual specification presented by UML. The more these two models are similar, the more the new approach will be precise.

6 References

[1] Kopetz, H. (1997) *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Springer, first edition.

[2] Duke, R., Rose, G. (2000) *Formal Object-Oriented Specification Using Object-Z*. MacMillan Press.

[3] Booch, G., Rumbaugh, J., and Jacobson, I. (1999) *The Unified Modeling Language User Guide*, Addison-Wesley Professional.

[4] Bjørner, D. (2006) *Software Engineering 3: Domains, Requirements, and Software Design*, Springer.

[5] Williams, J.R. (2009) *Automatic Formalization of UML to Z*, MSc Thesis, Department of Computer Science, University of York.

[6] Pressman, R. (2009) *Software Engineering: A Practitioner's Approach*, 7th edition, McGraw Hill.

[7] Schmidt, D.C. (2006) "Model-driven engineering", IEEE Computer, vol. 39, no. 2, pp. 25-31.

[8] Porres, I. (2001) *Modeling and Analyzing Software Behavior in UML*, PhD thesis, Department of Computer Science, Abo Akademi University, Finland.

[9] Amálio, N. (2006) *Generative frameworks for rigorous model-driven development*, PhD thesis, Dept. of Computer Science, University of York.

[10] Razali, R., Snook, C., Poppleton, M., Garratt, P. (2008) "Usability Assessment of a UML-based Formal Modeling Method Using Cognitive Dimensions Framework", Human Technology: An Interdisciplinary J. on Humans in ICT Environments.

[11] Baresi, L., Pezzè, M. (2001) "On Formalizing UML with High-level Petri Nets", In Concurrent Object-Oriented Programming and Petri Nets, Lecture Notes in Computer Science (LNCS) 2001, pp. 276-304, Springer.

[12] Risco-Martin, J.L., de la Cruz, J.M., Mittal, S., and Zeigler, B.P. (2009) "eUDEVS: Executable UML with DEVS theory of modeling and simulation", Simulation, vol. 85, Issue 11/12, pp. 750-77.

[13] Kim, S.K., Carrington, D.A. (1999) "Formalizing the UML class diagram using Object-Z", LNCS 1723/1999, 753.

[14] Kim, S.K., Carrington, D.A. (2000) "A formal mapping between UML models and Object-Z specifications" In ZB'00: Proc. of 1st Int. Conf. on B and Z users on Formal Specification and Development in Z and B, pp. 2-21, Springer.

[15] Stevens, B. (2006) "Implementing Object-Z with Perfect Developer", J. of Object Technology, vol. 5, no. 2, pp. 189-202.

[16] Rasoolzadegan, A., Abdollahzadeh, A. (2011) *Specifying a Parallel, Distributed, Real-Time, and Embedded System: Multi-Lift System Case Study*, Technical Report, Information Technology and Computer Engineering Faculty, Amirkabir University of Technology, Tehran, Iran.