Validation of Component based Software Development Model using Formal B-Method

Asif Irshad Khan*
Ph. D Scholar,
Dept. of Computer Science,
Singhania University,
Jhunjhunu, Rajasthan, India

Md. Mottahir Alam
Dept. of Computer Engineering,
Faculty of Engineering,
King Abdulaziz University, Jeddah,
Saudi Arabia

Noor-ul-Qayyum
Faculty of Computing
&
Information Technology,
King Abdulaziz University, Jeddah,
Saudi Arabia

Usman Ali Khan
Faculty of Computing
&
Information Technology,
King Abdulaziz University, Jeddah,
Saudi Arabia

ABSTRACT

IT industry in the present market situation faces high demand for performance and burgeoning user expectations; with the pressure manifesting itself in three forms – Development Cost, Time-to-market and Product Quality. One of the relevant techniques in this context is Component Based Software Development (CBSD) with a targeted and discriminative approach influencing all phases of development.

Although this approach offers great benefits of reusing ready-made components such as reducing time-to-market products and development costs, it also poses serious questions on the correctness of created systems. In this paper a methodology is proposed using a formal B-method approach, which supports formal stepwise development with refinement to model component-based systems in view of interaction among components. The proposed methodology is illustrated by a case study; Atelier B. is used to generate proof obligations and executable code automatically.

Keywords

Component Based Software Development, CBD, Formal B Method, Formal analysis, Software reuse and Model checking, interoperability, compatibility, adaptation, assembly.

1. INTRODUCTION

Now day's software is being used in everywhere from airways to railways, healthcare, telecommunications, security systems, weather forecast, earth quake alarm etc, software's are controlling every aspects of our life. Reliability and accuracy of software's plays a very crucial role as failure of software results in human losses and properties damage. For instance, now a day, with inbuilt advanced software systems, airplanes can fly in auto flight and auto control mode with less human intervention during the flight. Therefore, it is very important to rigorously check the functionalities of safety critical software. Researchers have proposed several techniques to effectively deal with these conflicting scenarios and draw optimized output. For critical systems formal methods are used to reason rigorously about the correctness of the program. Formal methods are techniques that are based on mathematical techniques and tools for the development and analysis of software systems. And can be applied at various points through the software development cycle. Correctness of the resulting product can be checked by applying appropriate mathematical modeling and analysis tool. However, there is no 100 percentage assurance of the correctness of the resulting product, but formal method techniques increase the level of correctness of the resulting product. Mostly formal methods are used in the development of safety critical systems where the cost of faults is unacceptable and it is very high.

The component-based development (CBD) uses independently developed components to build software. The motivation behind the re-use of software components is to reduce development costs and time-to-market and increase product quality. CBD if applied as per recommendation and consideration can reduce development costs remarkably and speed up the development process to meet the deadline, while enhancing the quality, flexibility and maintainability of the produced systems [1]. It is widely accepted that it is more reliable to reuse software than to create.

A component is an encapsulated unit characterized by its interfaces (provide and require services) that is meant for straightforward composition. Moreover, it is usually believed that a component interacts with its environment solely through its interfaces, and is designed in a way that it knows nothing about the environment it becomes a part of, as it should be reusable in different contexts [1].

Components provide and require services through public interfaces. The provide services are the operations performed by the component. Require services are the operations that the component needs in order to complete it's provide services and produce results. The interfaces of a component provide specification of the public services that are provide and require by the component [2]. Since the components are developed with no knowledge of their deployment context, the main issue is the correctness of the coordination and collaboration logic among components, focusing on interaction among components using their interfaces.

In this paper software Atelier B [3] is used for writing formal approach to modeling component-based systems in view of interaction among components. Atelier B is one of the tools supporting the B Method and has facilities for automatic verification and code generation. Atelier B tool is helpful in system prototyping, project management and documentation.

The paper is organized as follows: Section 2 covers brief overview of Formal Methods and Formal Specification Methods. Section 3 discussed related work. Section 4 provides an overview of Improved Component Based Software Development (ICBD) Model. Section 5 discussed about ICBD

Model evaluation using formal B method and lastly, Section 6 draws conclusions.

2. Formal Methods

The idea of program verification is not a new concept, it is used in software engineering and other fields since mid of nineteenth century. The field has evolved dynamically and today, there are many intellectual and automatic tools and techniques available for program analysis and verification.

Modern software development inevitably requires the design and analysis of a number of different artifacts. Formal methods allow the mathematically precise formulation of some of these artifacts. For instance, formulas in predicate logic capture operational requirements, state machines describe the behavior of code fragments and protocols, and object models capture static designs.

The advantage of using these formal notations is that they typically improve the overall quality of the artifacts by reveal and remove ambiguity, inconsistency and incompleteness and enabling automatic analyses that establish desirable properties. Consequently, the use of formal methods is indicated in domains in which the software has to meet very high quality standards and failure cannot be tolerated such as air-traffic control.

Moreover, the abstraction and automation capabilities of some formal techniques present a powerful weapon against the ever-increasing complexity of software. It is not necessary to apply formal methods to every aspects of a major system. Components that are safety critical should only be built using formal methods.

Formal methods are not a guarantee of correctness. It is possible that the final system, even when developed using formal methods, may have small omissions, minor bugs, and other attributes that do not meet expectations.

Mathematical models are used to describe the Systems and safety predictions can be analyzed and thoroughly verified. Formalizations can be categorized based on the Levels of Rigor, when chosen manually it is considered as low formalization, logical and discrete mathematics are used to reviews and inspections of the proposed mathematical model.

Formal specification language with type checking is considered as Medium formalization and fully formal specification language with rigorous semantics and proof checking is considered Fully Formalizations. Formal Specification Methods are as follows:

2.1 Formal Specification

The Purpose of Formal Specification is to state what system should do without describing how to do it. It is used mostly to translate the non-mathematical descriptions of a system like specifications written in natural language or using some diagrams, tables or pseudo code into formal specifications using mathematical notation with precisely defined vocabulary, syntax and semantics. Formal Specification also explains techniques that help discover problems in system requirements, the specification of a system uses a precise and unambiguous language like Z, B-method, VDM or CSP. Formal specifications may be used as a guide to the tester of a component in identifying appropriate test cases. In the early phases of software development Formal specification can be helpful in reducing requirements errors as it forces a detailed analysis of the requirements. Incompleteness and inconsistencies can be discovered and resolved.

2.2 Formal Proofs

In formal proof, formal notation and proof are written to reason about program correctness, instead of using a natural language to do the same. Most formal proofs are based in a "formal language" composed either of a subset of normal language or in symbols. A mathematical formal proof, for instance, is expressed using the symbols used in mathematics and does not rely at all on verbal language. In many cases, words are substituted for symbols so that even a non-mathematical formal proof can be understood in the form of simple symbolic logic without the use of potentially-ambiguous words.

2.3 Model Checking

Using Computer based tool to automatically perform model checking is done using Model Checker. Now days there are several Model checkers available, these model checkers have their own specific language for expressing the models, their properties and used special algorithm to implement the model checking. Some of the popular model checkers are SPIN, NuSMV and SAL.

2.4 Formal B – Method

B methods are specification method based on a mathematical formalism to build models. For refining, specifying and implementing software B method is considered as a right tool in the field of Formal methods. B methods are used to prove that a model is consistent in every possible case and mainly used for systems specification and software development. [4]

B methods follows an incremental approach in which first an abstract model of the system is defined which can be refined gradually step by step by adding further details until reached to implementation stage. [5]. The high degree of automation in verifying correctness improves scalability of B, speeds up development and, also, requires less mathematical training from the users.

Abstract machines are used to describe the Software systems in B method. Each abstract machine is expressed by set of its variables and operations, operations are used for modifying the variables values as per the specification. Abstract machine encapsulates a state and operations on the state [4].

3. RELATED WORK

Dorian et.al. [4] conducted an experimental study to merge software development approach using component-based methodology in an effort to manage complexity and to maximize reuse employing the formal B-method. Software properties like behavior and attributes are expressed in the specifications. The authors developed a tool to generate code in the spirit of the component approach from B specifications.

Xiaoli et.al. [5] proposed a framework for code generation using B formal specification based on component based method. Formal B abstract machines are used to derive software components according to their relativity. The derived software component can be translated into code directly by using proved translation rules. The whole system can be developed by applying assembly rules to software components [5]. Assembly strategies play an important role to produce the trustworthiness of generated code by the correctness of B abstract machines.

Interoperability between the different components in component-based software and system development is an issue and is addressed by Hatebur et.al. [6]. The authors proposed a methodology to address this issue. Interoperability is guaranteed by the use of the B method with its underlying concept of refinement [6].

To test and/or validate B-related experiments, open source and standardized format tools are easily available online [7] such as BRILLANT. This tool is helpful in validation formalisms (e.g., model-checking). All the necessary resources for building BRILLANT are available on the website dedicated to collaborative and free software development [7].

Zaremski et.al. [8] compare two software components to find if one component can be substituted by another. Components behavior is model using formal specification technique and Larch prover is used to prove matching components specification.

Ledang H. [9] addressed the problem of modeling in B object-oriented specifications in his research paper. He proposed a framework for software development based on objects and B from the requirements elicitation to the executable code. Moreover, B tool is used for the formal verification and analysis of object-oriented requirements models.

Authors Liu and Xiaoli [10] proposed a lightweight framework which transforms requirement documents written in natural language to executable codes. For the correctness of software development, B formal method is used.

In the framework [10], firstly elements and relations of the target system are acquired from the requirement documents through natural language analysis. Then modules are drawn using entities and relationship as per the composition clauses in B method. These modules are transformed in B specification which further refined into B implementation, and finally the B implementation is translated into executable codes.

4. OVERVIEW OF THE IMPROVED COMPONENT BASED SOFTWARE DEVELOPMENT MODEL

Reuse of existing artifacts is the essential aspect of the Component Based Software Development. These reusable artifacts have already been done with system requirement, architecture, components and case study. In this section we mentioned an overview of ICBD Model which has been empirically studied by Asif et.al. [11].

4.1 Component Based Software Development Lifecycle

The core phases of our improved CBSD model are:

i. System Requirement and Analysis

ii. System Design

iii. Component Identification and Adaption

iv. Component Integration Engineering

v. System Testing and Acceptance

vi. System Release and Deployment

These phases are shown in Fig 1. Fig 2 shows the details view of our ICBD model. A brief discussion about the model is as follows:

4.2 System Requirement, Specification and Decomposition

The first crucial step in developing an application for stakeholders is to study the system requirements by a team of software analyst for requirement elicitation. In CBSD this step is conducted by reviewing existing system requirement documents if any, and having meeting with the stakeholders.

After collecting the requirements comprehensively, requirement analysis process starts to identify common requirements of the system and subsystems, and to find potential reusable software components [12].

The major outcomes of this phase include system requirements outline and identification of the components that can be reuse on the common requirements as system analysts has knowledge of available components in the in-house repository.

4.3 Component Requirement and Selection

Based on the collected system requirements, a system architecture model is designed by matching requirement approach. From the system requirements the software team determines that which of the software requirements can be considered to composition rather than building them from scratch.

At this stage a complete cost benefit analysis is required to identify various cost factors involved in adopting the component. These cost benefits analysis assists in making a decision to reuse a component or to acquire COTS [13].

4.4 Component Adaptation and Verification

As candidate component is selected, issues related to its capability and fitness in the architecture need to be addressed too. The selected component should be fit in the system architecture design, a study usually being done to know how far the selected component is compatible with the system architecture.

Similarly verification of the component is being made through the software metrics and cost benefit analysis techniques [12]. Moreover the software architecture must be scalable so that the components can be easily integrated into the system.

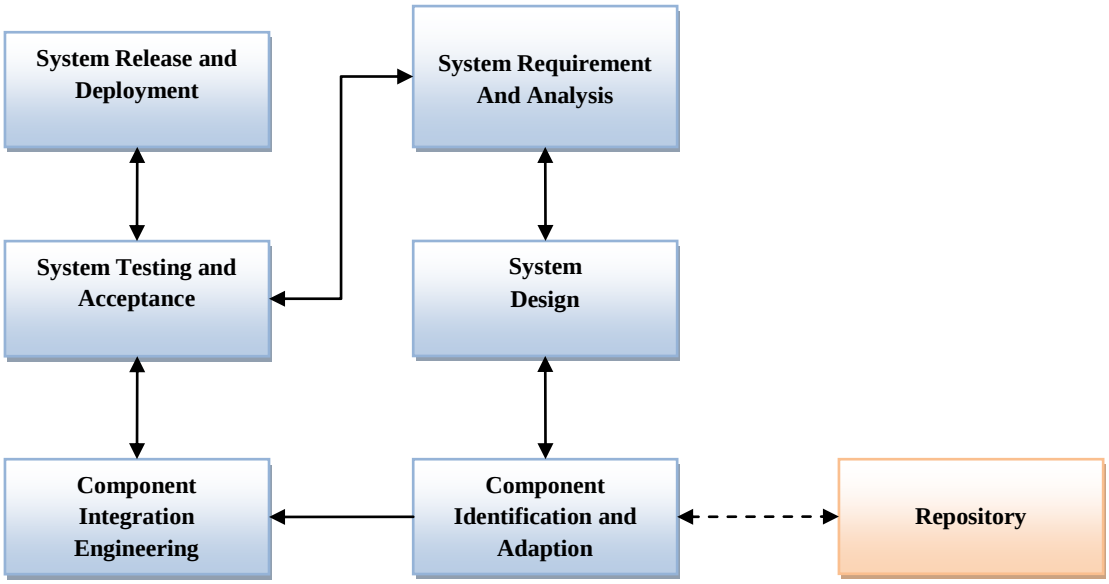

Fig 1: An Improved Model for Component Based Software Development [11]

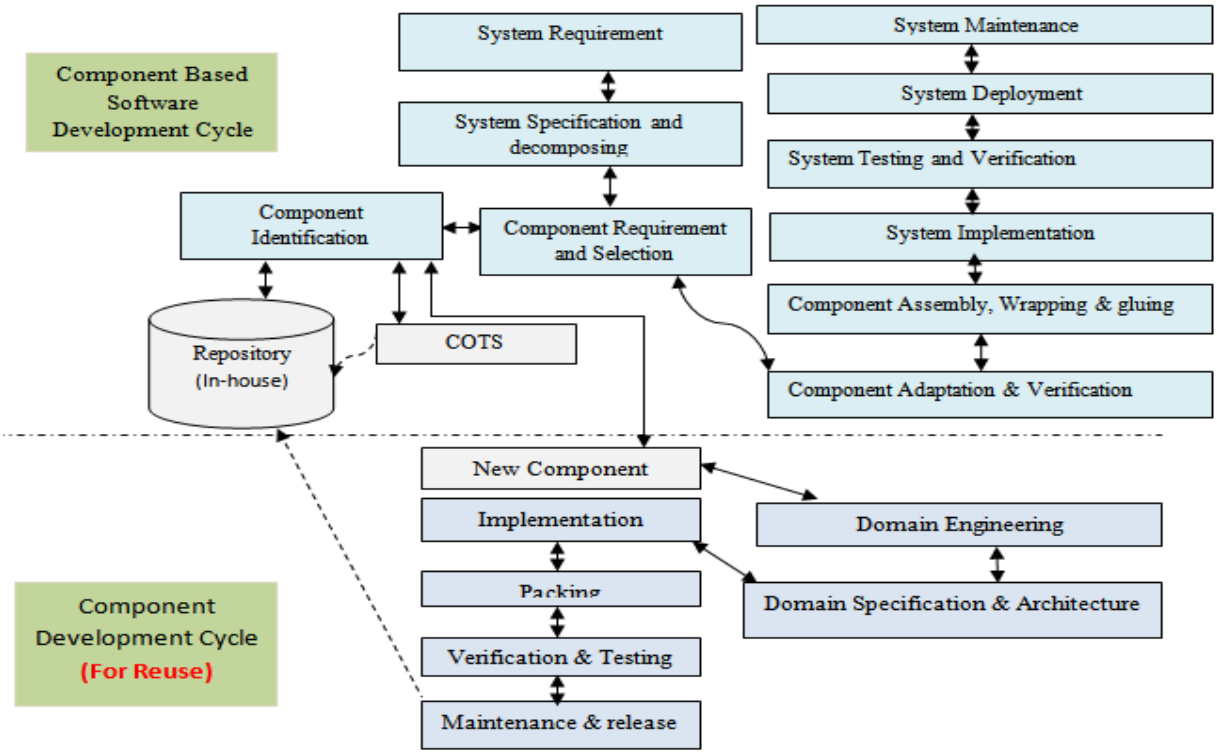

Fig 2: Detail view of Improved Model for Component Based Software Development [11]

4.5 Component Assembly, Wrapping and Gluing

Development of an application that involves integration of different components needs communication among the components through an interface or "glue". Component wrappers help in isolating a component from rest of the system components. Additionally, component wrappers are also useful in mapping data representations.

4.6 System Implementation

During implementation phase, no coding is required theoretically, however in practical, coding is usually required because all functionalities are rarely found in a component.

4.7 System Testing and Verification

The purpose of testing an application is to investigate whether the software going to be delivered to the customer is a quality

product and it really works according to the given requirements and specifications.

In case of CBSD, the lack of details about component source code and design make it very difficult to track the faults while using COTS components. This may cause difficulty in testing of individual components and integrated systems.

4.8 System Deployment

The system deployment phase involves successful release of the complete product to the customer in the specified environment so the software is made available to the customer for use. System deployment is a complex endeavour therefore it must be delivered using some specialized tool to make the deployment easy from the customer perspective.

4.9 System Maintenance

Up-gradation and substitution of components are the main tasks of the system maintenance in a CBD model. System up-gradation usually occurs when a COTS supplier releases a new version of a component or when a new COTS component is outdated. Modifications to the code wrappers and glue code are required in this phase.

5. CASE STUDY: A SIMPLE ELEVATOR CONTROL SYSTEM

We illustrate our purpose of study with a case study of a simple Elevator (Car) Control System.

5.1 System Requirement and Analysis

A simple car has the basic functionality of all modern elevator systems have, such as car position, accept car calls, show car direction up or down, car door status open, closing, closed, opening, Door obstructed. Further, an elevator controller has provisions in the cases of emergency, and alarm notification.

The car itself consists of several parts: A door, which can be opened and closed by a motor. Sensors inform the control system about the door position. A light sensor can detect objects while the door is closing. The elevator car engine moves the car up or down [14].

Passenger use car to move from one floor to another floor which is generally numbered from 0 to N, where N is the top floor and 0 is ground floor. Floor Number Displays indicate which floor an elevator is on. Each elevator cabin has one floor number display. The terminal floors and other selected floors shall be used by the controller to check the correct floor value.

Passengers arrive at the car randomly and are served in queued request, they exits the car at Elevator arrived at floor, when the car is parked with the doors closed, the door opening cycle shall be initiated.

There are car call buttons in the car corresponding to each floor. At the destination floor car must stop and door should open for a timed time, also, car door must be closed before the car starts her service. An open/close button is inside an elevator cabin and is used for opening and closing elevators doors.

Passengers know the current moving direction of the car on a lantern. For any hazardous situation emergency brake should be triggered and the car should be forced to stop.

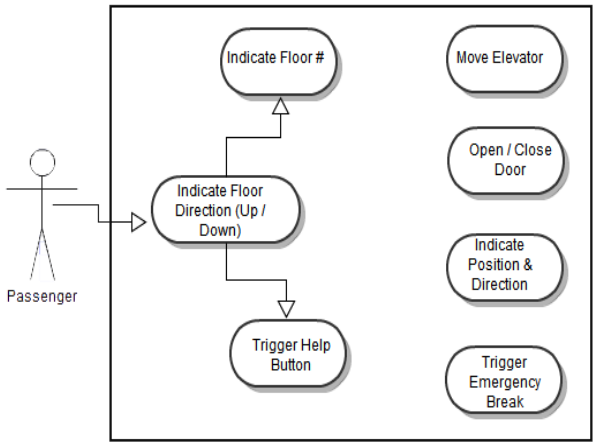

Fig 3: Use case diagram of the system

Fig 3 shows use case diagram model of the design view of the system. Use case diagrams are generally used to modeling the behavior of a system or a class or a subsystem. Mainly use case diagrams show actors their association relationships and dependency.

Move Elevator: This is the main functionality of the car, the job of the car is to read the hall calls and move the car to that floor. The car serves the calls in sequence, stopping always at the nearest call floor.

Indicate Floor number: This use case includes the passenger wish to go to a particular floor.

Trigger Help Button: This is located inside the car and is used to make an alarm sound. This use case includes the passenger safety mechanisms in case of any emergency.

Open / Close Door: The car should be able to open and close the door for the passenger to get in and out of the car. An open/close button is inside an elevator cabin and is used for opening and closing elevators doors.

The door stays open for some defined time to allow passengers to enter or exit the car, If, while the door closes, the optical sensor is interrupted or the floor's request button is pressed by a user, the door must open again. After a shorter specified waiting time the door closes again. If no hall calls is in memory, the car moves in waits state at the same floor.

Indicate Position and Direction: This use case is helpful for the passenger to know the current movement direction of the car (up/down) as well as the floor position indicated via lantern.

Trigger Emergency Break: This use case is used in case of any unusual things happened while the car is in operation.

Passenger: Is someone who uses elevator to transport from one floor to another.

5.2 System Design and overall Architecture of Elevator access control system

We use the general model of an elevator system and map it to our elevator model. Fig 4 shows the resulting component structure. These components mapped to the elevator control system specification.

Further, a partial view of the Architecture of Elevator access control system is shown in Fig 5. In the

architecture we have illustrated main component *ElevatorControl* related to the system requirements. As mentioned earlier in the paper components interact with the environment using require and provide interface or in other words a component has its behavior defined in terms of require and provide interface.

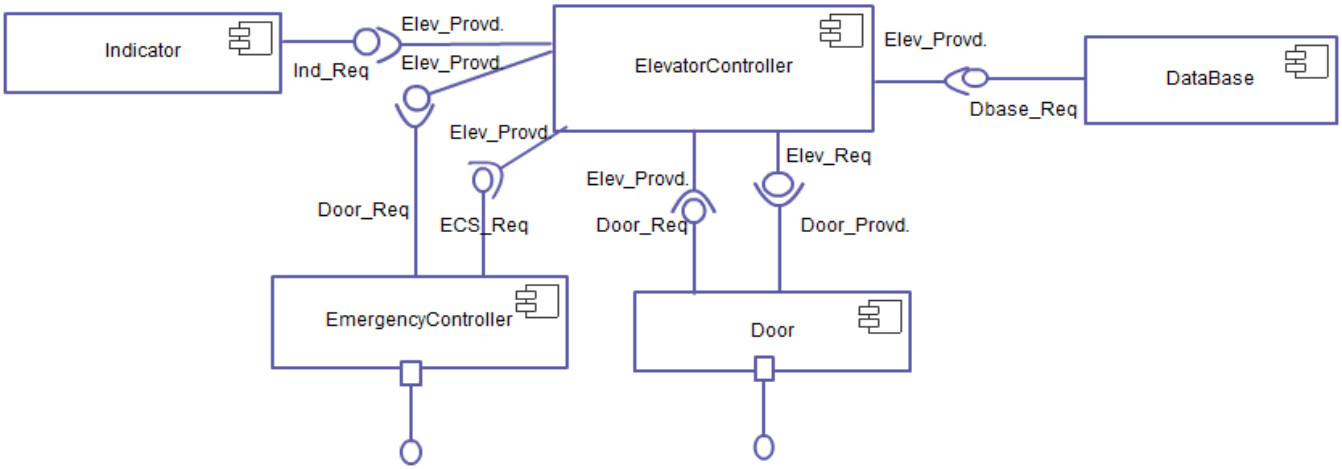

Fig 4: Component architecture of Elevator system

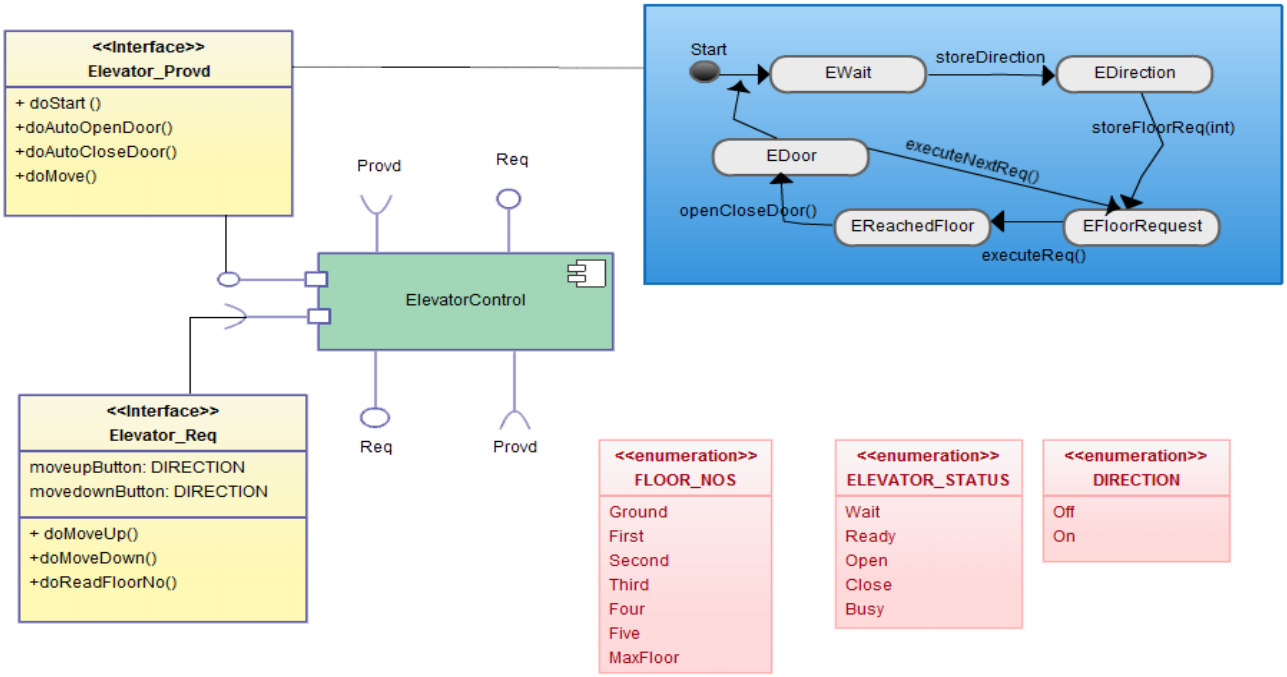

Fig 5: Partial view of the architecture of Elevator control system

Provide interface is the services which the components offers. It is what the component does, they are denoted by "socket" symbol, while require interface is what the component requires or needs to do its job. They are denoted by "lollipop" symbol.

Two interfaces as shown in Fig 7 and Fig 8, modeled by class diagram with their behavior is mentioned in the class diagram. Elevtor_Provd corresponds to the provide interface of the ElevatorControl corresponding to the component identification and Elevtor_Req is the require interface.

Enumeration data types are defined are as they are perfect for representing the state/status of the problem, further enumeration is a collection of variable constants and the business concept of the problem is reinforced because of the strong typing.

```
/* Elevator Control
 * Author: Asif Khan
 * Creation date: 2/10/2013
 */
MACHINE
    Elevator_Provd
VARIABLES
     elevator_Status , elevator_Floor,
Num
INVARIANT
    elevator_Status ∈ ELEVATOR_STATUS ∧
    elevator_Floor ∈ ELEVATOR_NOS
INITIALISATION
    elevator_Status := Wait ,
    elevator_Floor := Ground
OPERATIONS
    doStart =
      PRE   elevator_Status := Wait
      THEN  elevator_Status := Ready
      END

    doAutoOpenDoor =
      PRE   elevator_Status := Ready
      THEN  elevator_Status := Open
      END

    doAutoCloseDoor =
      PRE   elevator_Status := Open
      THEN  elevator_Status := Close
      END

    doMove (Num)=
      PRE   elevator_Status = Close ∧
            Num = ELEVATOR_NOS
      THEN  elevator_Status := Busy ∧
            elevator_Floor:= Num
END
END
```

Fig 6: Provide interface Elevator_Provd

```
/* Elevator Control
 * Author: Asif Khan
 * Creation date: 2/10/2013
 */
MACHINE
     Elevator_Req
VARIABLES
     up_button, down_button, elevator
_status
INVARIANT
    up_button ∈ DIRECTION ∧
    down_button ∈ DIRECTION ∧
    elevator_Status∈ ELEVATOR_STATUS
INITIALISATION
    up_button, down_button = Off, Off
OPERATIONS
    doMoveUp =
      PRE up_button = Off ∧
          down_button = Off
      THEN
          up_button = On
      END

     doMoveUp =
      PRE up_button = Off ∧
                  down_button = Off
      THEN
          down _button = On
      END
    doReadFloor(Num) =
      PRE   elevator_Status = Close
      THEN
           elevator_Status = Wait
END
END
```

Fig 7: Require interface Elevator_Provd

5.3 *Component Identification and adaptation (available components)*

Reusability of the component can be determined in terms of formal specifications that precise described component behavior. In addition, they are described in terms of a domain model that can be reused across applications [15].

Component Identification result in a component that satisfies a problem specification. Selected component need to go through adaptation process in some cases multiple components need to be combined to solve the problem [15].

The interface between the control system and the environment is defined in this diagram. We identified available three components, namely EmergencyControlServices, Indicator, and Door. Functionalities of these three selected are known by their interface description as shown in Fig 8, 9 and 10. Formal B methods are used to validate their behavior and control specifications.

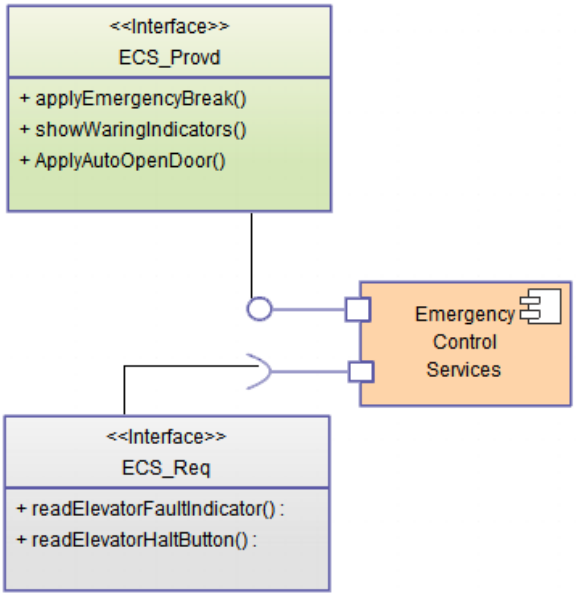

Fig 8: Component Emergency Control Services and it's require and provide interface.

Component EmergencyControlServices:

EmergencyControlServices component equipped with provide and require interface. In case of the emergency situation this component will be used to control the car and the job of the component is to make sure emergency break should be applied and Car moves to the closest floor and cancels all requests, Car open doors and keeps them open, and the system does not accept any more requests.

EmergencyControlServices component require interface ECS_Req give information to its elevator controller by two defined methods as shown Fig 8. The provide interface of the component are applied using three methods applyEmergencyBreak, showWarningindicators and ApplyAutoOpenDoor.

The component Door: As shown in Fig 9. This component is used to control the doors of the car. Door functionality of the car is handled by this component's provide interface namely, Door_Provd which contains 3 behavior to make sure that door open or close safely and handle any interruption. The Door component require interface namely, Door_Req contains 5 methods and autocloseRequest behavior sends signals to close the door after allotted timeout is reached to expiry.

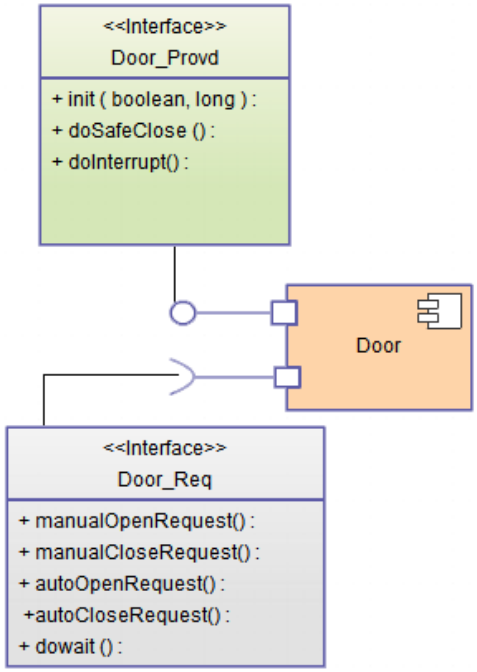

Fig 9: Component Door and it's require and provide interface.

Component Indicators: As shown in Fig 10 this component is used to show direction of the car (up and down), also, this component provide services name Indicator_provd mainly the current car floor. Warning message is flashed on the lantern in case any problems detected by the car. Indicator_provd contains 3 Indicator_Req component contains 2 methods.

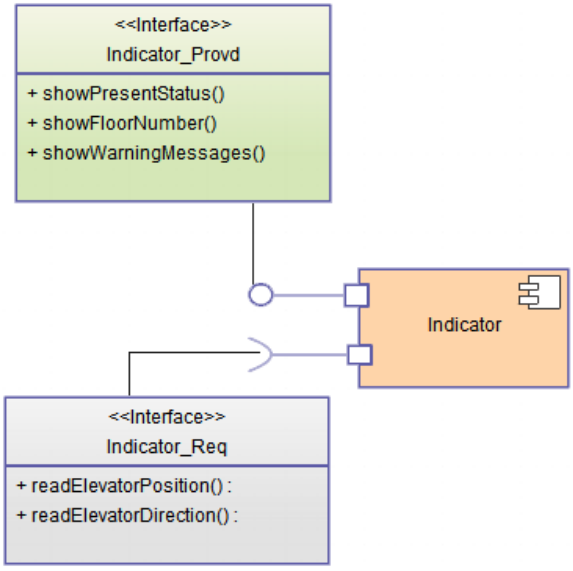

Fig 10: Component Indicator and it's require and provide interface

5.4 Component Integration and assembly

Different components are assembled using a define architecture. Require and provide interfaces of the different component play a crucial role in component assembly.

In general, deals with the different types of interfaces i.e. provide and require for all the candidate components those are connected to it. This component assembly realizes all the require interfaces of the considered components using their provide interfaces. A gluing algorithm is written to obtain the require attribute from the attribute of the provide interfaces.

UML 2.0 architecture of the component assembly is show in Fig 11, and its refinement is shown in Fig 12. Some of the tasks performed by this component assembly architecture can be listed underneath:

- It realizes the require interfaces Elevator_Req of ElevatorControl,
- It realizes the require interfaces ECS_Provd of EmergencyControlServices,
- It uses the provide interfaces of the existing components such as Door, Indicator, EmergencyControlServices and Database

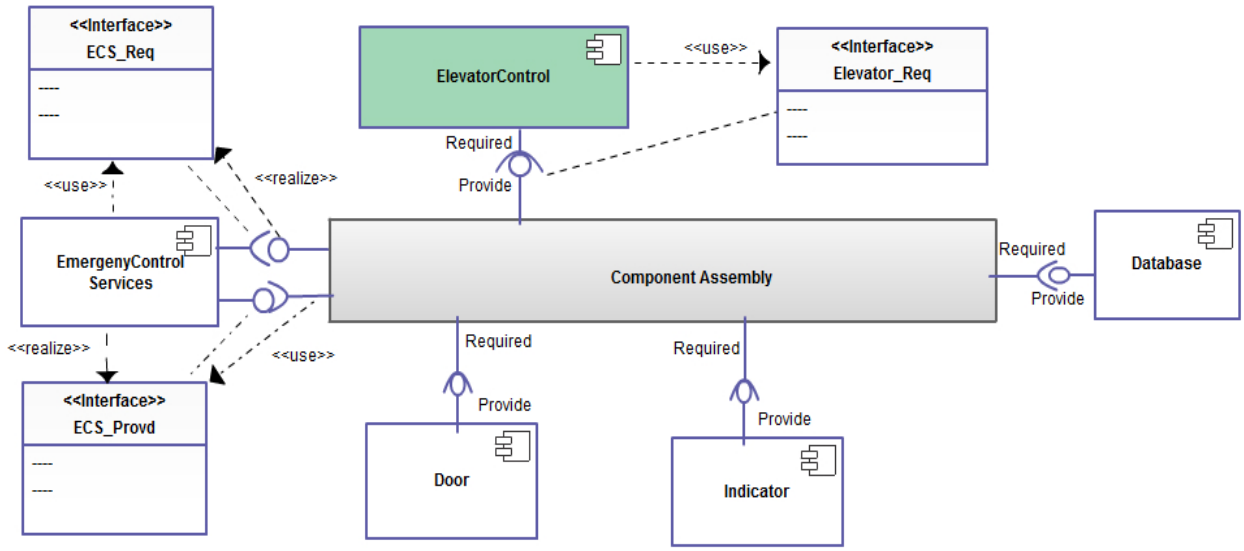

Fig 11: UML 2.0 Component Assembly diagram

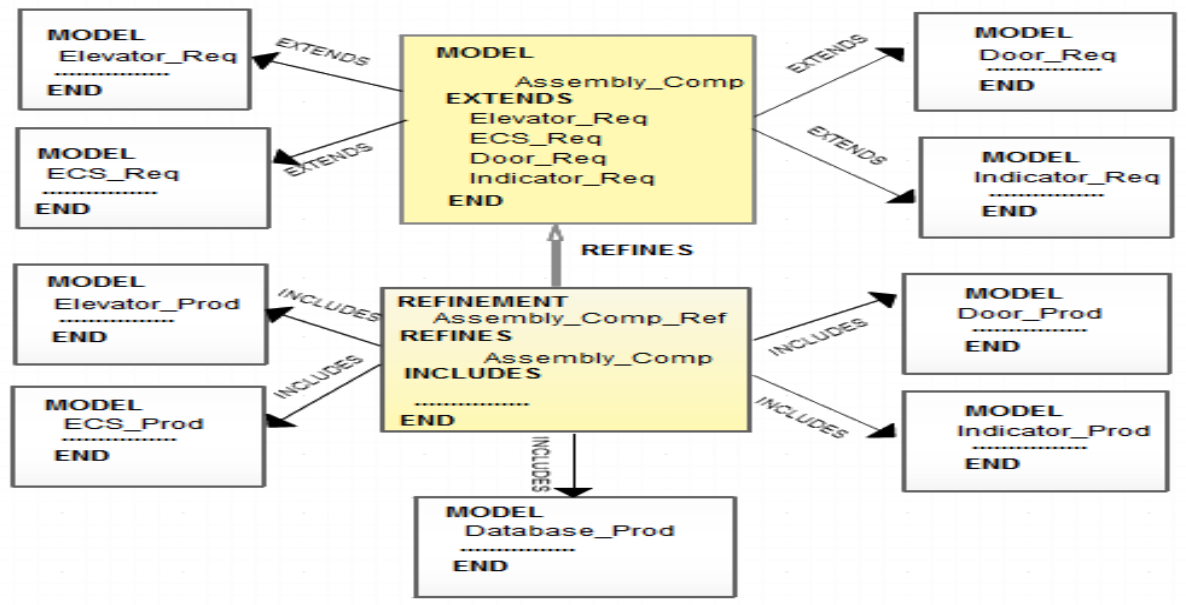

Fig 12: B Architecture of Component Assembly Refinement

5.4.1 *A gluing priori correctness Algorithm*

The glue algorithm as shown in Fig 13 plays a vital role in the integration of different component in CBSD methodology. The glue algorithm helps in the integration of mismatching components and in the implementation of missing functionalities of the require specifications of the software system, to satisfy stakeholder requirements. This gluing algorithm addresses timely identification of possible mismatches between component interfaces and missing functionalities. The glue algorithm performs two checks:

i. Check on Component interfaces using all the require and provide interfaces.

ii. Component code specification and verification to identify mismatches between require and provide interfaces.

With the help of component certification [16], we can determine the a priori correctness of the integrated solution. A certified component ensures that its interface is properly specified and its code should be verified against its specifications [17].

Algorithm: A priori gluing algorithm

1. **Input** Interfaces, code and specification of each component
2. **Output** A priori correctness result (in terms of 0 and 1)
3. **for** i= 1 → n **do** ▷ for each component
4. C_i = check (Interfaces) ∧ verify (code, specification)
5. **end for**
6. result = $\sum_{i=1}^{n} Ci$
7. **if** result == 1 **then**
8. **return** 1; ▷ A priori correctness check : PASS
9. **else**
10. return 0; ▷ for each component check : FAIL
11. **end if**

Fig 13: A priori gluing algorithm

5.5 Refinement in Component Assembly using Formal B-Method

Refinement ensures that the resulting implementation conforms to the initial abstract specification. The basic principle of formal B method is refinement or incremental approach. The refinement process starts from creating an abstract albeit implementable specification and finishes with generating an executable code [18].

The intermediate stages yield the specifications containing a mixture of abstract mathematical constructs and executable programming artefacts [18]. Fig 14 shows Component Assembly Refinements.

```
/* Elevator Control (Assembly_Comp_Ref)
* Author: Asif Khan
* Creation date: 2/16/2013
*/
REFINEMENT
    Assembly_Comp_Ref
REFINES
    Assembly_Comp
SEES
    Type
INCLUDES
    elevator.ElevatorControl_Prd
    database.Database_Prod
    light.Indicator_Prod

    door.Door_Prod
    ecs.EmergencyControlServices_Prod
```

```
INVARIANT
    (light.UpIndicator=ON ∧
    elevator.ElevatorStatus=ready ∧
    light.Indicator=ON ∧
    door.status ∈{open,close})
    ⇒ UP_Button=ON ∧ Down_Button=OFF ∧
light_indicator=FloorNO
    ∧ (light.DownIndicator=ON ∧
    elevator.ElevatorStatus=ready ∧
    light.Indicator=ON ∧
    door.status ∈{open,close})
    ⇒ Down_Button=ON ∧ UP_Button=OFF ∧
light_indicator=FloorNO
OPERATIONS
    doorAutoClose=
    BEGIN
            timer.start(10);
        IF  timer.expire() ∧
door.interference()=null
        THEN
            light.indicator=ON;
            Elevator.Status=Busy;
            door.autoclose();
        END
    END
    floorRead=
    BEGIN
        IF  doorAutoClose()=true ∧
Elevator.Status=Busy
        THEN
            Elevator.Read=FLOOR_NO;
        END
    END
    elevatorMove=
    BEGIN
        IF   doorAutoClose()=true ∧
Elevator.Status=Busy ∧ Elevator.Read=ℕ
        THEN
            ASSERT
                ¬(FLOOR_NO ≠ℕ) ;
            Elevator.doMove(FLOOR_NO);
            light.indicator=FLOOR_NO;
        END
    END
    endFloorReached=
    BEGIN
        IF
            Elevator.Floor=FLOOR_NO
        THEN
            light.indicator=FLOOR_NO;
            Elevator.Status=wait;
        END
    END
    doorAutoOpen
    BEGIN
        IF Elevator.Status=wait ∧
Elevator.Floor=FLOOR_NO
        THEN
            door.autoopen();
            timer.start(10);
            Elevator.Status=Busy;
        END
    END
END
```

Fig 14: Component Assembly Refinements

6. CONCLUSION

In this paper we proposed a formal methodology based on the refinement paradigm to specify and validate our CBD model with a case study of a simple elevator control system. Under this approach, the components are selected based on the requirements, and then, a composition is defined using B method to validate the correctness of the created system. We also proposed a priori gluing algorithm which helps in the integration of mismatching components and in the implementation of missing functionalities of the require specifications of the software system.

Further we proposed a refinement-based composition technique keeping a separation between use and realize components in order to build correct automated systems satisfying user requirements. The advantage of the refinement approach for verification is that properties which hold in abstraction are preserved by refinement.

7. REFERENCES

[1] B. Zimmerov´a 2008. Modelling and Formal Analysis of Component-Based Systems in View of Component Interaction. PhD thesis, Masaryk University, Brno, Czech Republic.

[2] Mubarak Mohammad & Vangalur Alagar (2011): A formal approach for the specification and verification of trustworthy component-based systems. J. Syst. Softw. 84, pp. 77–104, doi:10.1016/j.jss.2010.08.048.

[3] Atelier B, ClearSy System Engineering, Aix-en-Provence, France, available for download [online] http://www.atelierb.eu/en/download-atelier-b/

[4] Petit, Dorian, Vincent Poirriez, and Georges Mariano 2004. "The B method and the component-based approach." Journal of Design & Process Science: Transactions of the SDPS 8.1 (2004): 65-76.

[5] Xiaoli, Liu, et al. 2007. Code Generation from B Specification based on Component Oriented Approach in Information Technologies and Applications in Education, 2007. ISITAE'07. First IEEE International Symposium on. IEEE, 2007.

[6] Hatebur, D., Heisel, M., Souquières, J. 2006. A method for component-based software and system development. In: Proceedings of the 32tnd Euromicro Conference on Software Engineering And Advanced Applications, pp. 72–80. IEEE Computer Society Press, Los Alamitos (2006).

[7] Colin, Samuel, et al. 2005. "BRILLANT: An open source and XML-based platform for rigourous software development." Software Engineering and Formal Methods, 2005. SEFM 2005. Third IEEE International Conference on. IEEE, 2005.

[8] Zaremski, A.M., Wing, J.M. 1997. Specification Matching of Software Components. ACM TOSEM 6(4):333-369, (1997)

[9] Ledang H. 2001. Formal techniques in the object-oriented software development : an approach based on the B method[J].LORIA, 2001, 13（12） : 1-5.

[10] Liu, Xiaoli 2009. B Method Based Framework for Correct Software Development, International Conference on Computational Intelligence and Software Engineering,CiSE, 1 - 4 , 2009

[11] Asif Irshad Khan , Noor -ul-Qayyum , Usman Ali Khan 2012. An Improved Model for Component Based Software Development in Software Engineering, Vol. 2 No. 4, 2012, pp. 138-146. doi: 10.5923/j.se.20120204.07.

[12] Sajid Riaz 2012. Moving Towards Component Based Software Engineering in Train Control Applications, Final thesis, Linköpings universitet, Sweden, 2012.

[13] Critical software practices, Pro-Concepts LLC,Online Available: http://www.spmn.com/www2/16CSP.html#system

[14] Frank Strobl and Alexander Wisspeintner, Speciation of an Elevator Control System, TECHNISCHE UNIVERSITAT ,MUNCHEN , 1999 , available [Online] http://www4.informatik.tu-muenchen.de/publ/papers/elevator.pdf

[15] Penix, J. 1998. Automated Component Retrieval and Adaptation Using Formal Specifications. PhD thesis, University of Cincinnati.

[16] Morris, John, et al. 2001 Software component certification. Computer 34.9 (2001): 30-36.

[17] K.-K. Lau and M. Ornaghi 2001. A Formal Approach to Software Component Specification, Proceedings of Specification and Verification of Component- based Systems Workshop at OOPSLA2001, pages 88-96, Tampa, USA, Oc- tober 2001.

[18] L. Laibinis and E. Troubitsyna 2004. Refinement of Fault Tolerant Control Systems in B. In M.Heisel, P.Liggesmeyer, and S.Wittmann, editors, Computer Safety, Reliability, and Security - Proceedings of SAFECOMP , number 3219 in Lecture Notes in Computer Science, pages 254-268. Springer-Verlag, Sep 2004.

Author's Profile

Asif Irshad Khan received his Bachelor and Master degree in Computer Science from the Aligarh Muslim University (A.M.U), Aligarh, India in 1998 and 2001 respectively. He has more than eight years experience of teaching as lecturer to graduate and undergraduate students in different universities and worked for four years in industry before joining academia full time. He has published more than 16 research papers in reputed International journals. Currently, he is a research scholar at Department of Computer Science, Singhania University, Jhunjhunu, Rajasthan, India. His current research interests include software engineering with a focus on Component Based and Agent Oriented Software Engineering.

Md Mottahir Alam has around six years of experience working as Software Engineer (Quality) for some leading software multinationals where he worked on projects for companies like Pearson and Reader's Digest. He is ISTQB certified software tester. He has received his Bachelors degree in Electronics & Communication and Masters in Nanotechnology from Faculty of Engineering and Technology, Jamia Millia Islamia University, New Delhi. He has several papers in international journals. He is presently working as a Lecturer in the Faculty of Electrical and Computer Engineering, King Abdul Aziz University, Jeddah, Saudi Arabia. His research interest includes Software Engineering, Component Based Software Engineering and Agent Based Software Engineering.

Mr. Noor-ul-Qayyum received his Master degree in Information Technology from National University of Sciences and Technology, in 2009. He is currently working as a lecturer in King Abdul Aziz University. He has industry experience in SCORM based e-learning courseware development using ADDIE model. His research interest includes e-learning, software watermarking, and mobile agent security issues.

Dr. Usman Ali Khan: is an Associate Professor in the Information Systems Department, Faculty of Computing and Information Technology, King AbdulAziz University, Jeddah, Saudi Arabia. He received his Ph.D. in Software Engineering from the Integral, University, India. He has been working is the field of research and teaching at graduate and undergraduate level since 1995. He has published many research papers at national and international forums.