TRACEABILITY OF UNIFIED MODELING LANGUAGE DIAGRAMS FROM USE CASE MAPS

Ahmed Lawgali

University of Benghazi, Faculty of Information Technology

ABSTRACT

The Unified Modeling Language (UML) is a general purpose modeling language for specifying, constructing and documenting the artifacts of software systems. It is used in developing systems by combining the use of different types of diagrams to express different views of the systems. These diagrams allow transition between requirements and implementation. The lack of traceability between the diagrams makes any changes difficult and expensive. In this paper, it is proposed using the Use Case Maps (UCMs) notation which allows the full description of the system in terms of high-level causal scenario and helps in visualizing and understanding the system in early stage. UCMs was used in the early stage to describe the system and generate the proper UML diagrams from UCMs. By defining a traceability relationship between UCMs and UML, we facilitate the maintains and the consistency of the UML diagrams.

KEYWORDS

UCMs, UML

1. INTRODUCTION

The Unified Modeling Language (UML) [6] is a general purpose modeling language used for specifying, constructing, and documenting the artifacts of software systems. It is a very important part of developing object oriented software, and software development process. The UML represents a collection of the best engineering practices that have proven successful in the modeling of large and complex systems [6]. It allows the transition between requirement, and implementation, by combining the use of different types of diagrams to express different views of the systems. Each of these diagrams deals with specific system aspects such as scenarios, structure, and component behavior. This permits designers to focus on different issues at different times. A diagramming technique called Use Case Maps (UCMs) allows the full description of the system in high level terms and helps in visualizing and understanding the system in it's early stages. UCMs allows the description of the system in terms of high level casual scenarios by superimposing scenario paths on a structure of abstract components and describing the casual relationship between responsibilities [8]. The UML has made a significant impact on the progress of systems development. There are many papers which discuss the problem of the gap between use case and the other behavior diagrams [2][3][4]. In [5] the authors discussed the relationship between UCM, and Message Sequence Chart (MSC), and also defined a transition that allows movement from UCM to MSC in a systematic and traceable manner. In [1] the authors introduced a paper addressed Driving Message Sequence from Use Case Maps Scenario Specification. In [11] the work presents the results of an experiment combining existing tool supported techniques, for the generation of MSCs from UCMs and for the synthesis of SDL from MSC. However, a significant problem remains which is the lack traceability between diagrams. UML combines the use of several different diagrams, and these diagrams are constantly modified as new requirements are added, however, we face difficulties in identifying which diagrams need

DOI : 10.5121/ijsea.2016.7607

to be modified. The lack of traceability between the diagrams makes the changes difficult and expensive. The lack of consistency between the diagrams makes them appear as if they are not related. To facilitate the maintenance of consistency between diagrams we must define traceability relationships. Traceability can be defined as how different diagrams relate to each other and how they affect each other. It produces the linkage of elements in different diagrams.

It is proposed using UCMs in the early stages to describe the system, and trace the UML diagrams from UCMs. We provide the relationship between the UCM and the UML diagrams as a guide to the designer in forming UML diagrams. By defining the traceability relationship between UCMs and UML diagrams, the maintenance of consistency was facilitated between the diagrams. The UCMs was used as a starting point for generating UML diagrams, because of their suitability for high level visual representation, and their ability to simply, and successfully, depict the design of complex systems [7].

The primary purpose of this paper , is to provide an approach to illustrate the traceability relationship between UCMs and UML digrams. This allows the generation of UML diagrams from UCMs and helps in solving the lack of traceability problem between the UML diagrams.

2. Background: UCM And UML

2.1 Use Case Maps

Use Case Maps (UCM) is a scenario-based approach. It provides a high level view of causal sequence in the system as a whole. It has been proposed by *Buhr* and *Casselman* [8]. UCM is used to describe scenario paths in terms of causal relationships between responsibilities which may be allocated to components. Causal relationships help us to distinguish between two types of events. Events caused by other events, and events observed one after another, without one affecting the other. UCM allows the full description of the system in high level terms and helps in visualizing and understanding the system in its early stage.

A UCM path may have any shape as long as it is continuous. It expresses the sequence of responsibilities, that need to be executed by system components in order to achieve the overall objective of the system. The path starts at a starting point (depicted by a filled circle) and ends at an end point (shown as a bar). Between the start and end points, the scenario path may perform some responsibilities along the path. Figure 1 illustrates the basic elements of UCMs.

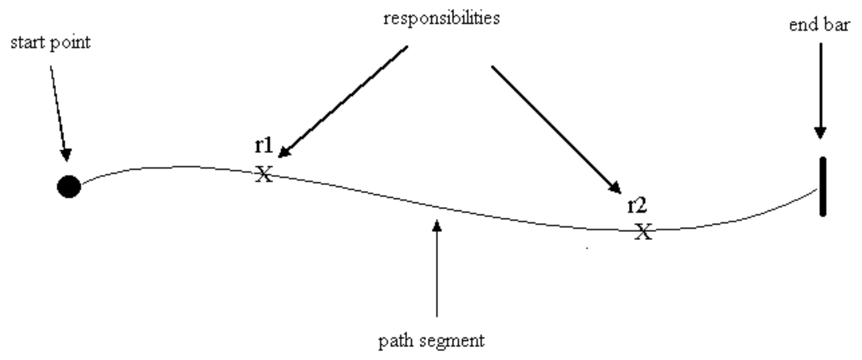

Figure 1. UCM basic path

2.2 Unified Modeling Language

The Unified Modeling Language (UML) [6] is a standard language for specifying, visualizing, constructing and documenting the artifacts of software systems. The UML represents a collection of the best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software, and the software development process.

UML is used in developing systems by using different types of diagrams as shown in figure 2, which express different views of the systems, allowing transition between requirement, and implementation. It is used to graphically express the design of object oriented systems. UML expresses the static and dynamic aspects of object oriented systems

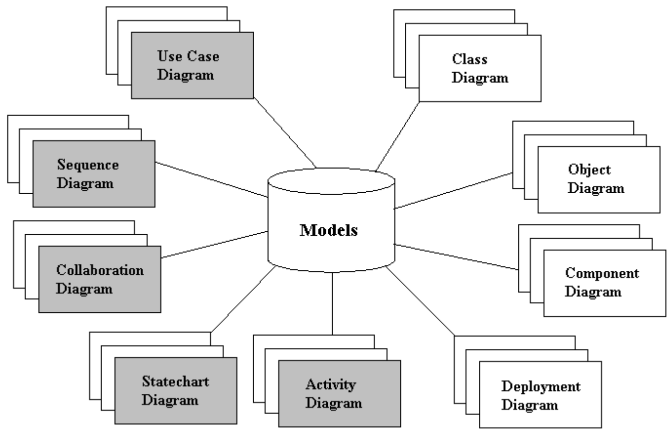

Figure 2. UML diagrams

3. DEFINING TRACEABILITY RELATIONSHIPS BETWEEN UCM AND UML DIAGRAMS

Traceability is the property that defines how different models relate to each other. It allows the linking of the elements contained in different models. The existence of traceability relationships allows for the evaluation of the impact of modifications on the different models, and making of changes to the affected models in a consistent manner [10]. Figure 3, explains the relation between UCMs and UML diagrams and the traceability between them. The derivation of UML diagrams starts by deriving use case diagrams from UCMs. This approach two ways of generating UML diagrams. One way is to generate directly from UCMs the UML diagrams. For example, use case diagram and activity diagram. The other way is to generate the UML diagrams from UCM and other UML diagrams. For example, the interaction diagrams(*Sequence diagram and Collaboration diagram*) in the figure was derived from UCMs, use case diagram and class diagram. The state diagram was derived from UCMs and the interaction diagrams. The arrows in this approach explain the traceability. The relationship between UCMs and implementation diagrams (*Component diagrams and Deployment diagrams*) was not provided. Because these diagrams are not used in early stage design and UCMs used in early stage to describe high level view of the system.

The approach provided explains the relation between the UCMs scenario and the UML diagrams as a guide to the designer to deduce UML diagrams from UCMs which make the UML diagrams consistent.

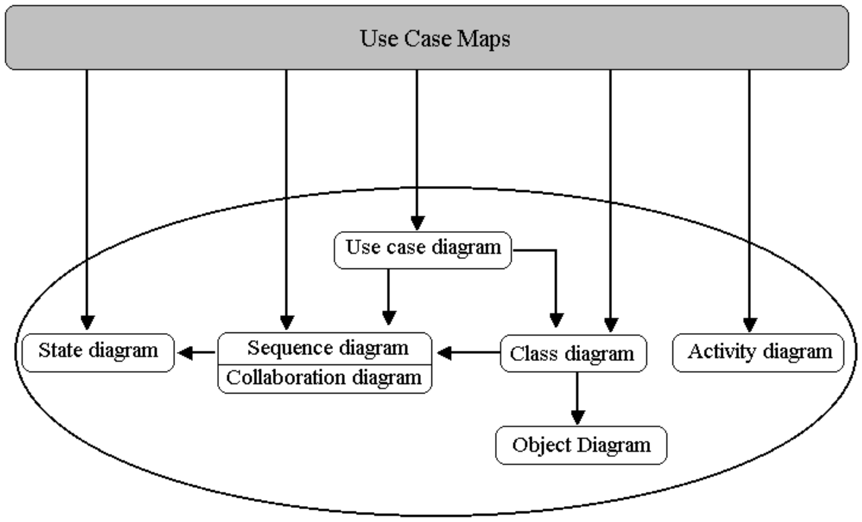

Figure 3. Relation between UCM and UML diagrams

3.1. UCMs and Use Case Diagram

A use case is a set of scenarios that describes an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. UCMs scenarios are formed of components and paths. There is no direct representation of the actor in UCM. But the components in UCM may represent an internal or external entity. External component in UCM may represent an actor which can affect the system. By analyzing the components represented in the UCMs, we can specify which of these components represent an internal entity, and which represent an external entity to the system, and also which of these components can affect the system like an actor in the use case diagram. In [4] the author explains the relation between use case and stubs. All of them hide the details, and all of them are executed when the precondition triggers to reach the postcondition. The stub hides details which may be illustrated by a *plug-in* scenario. Therefore, from plug-in scenario for stubs we can derive details for the use case such as (*precondition* , *postcondition* , … *etc*). Use case diagram have two types of relationship between use case: *Uses* and *Extends*

3.1.1. Uses Relationship

The *uses* relationship was used, when we have a frequently used sequence of behavior in many use cases, and we don't want to repeat the same description in all the use cases [9]. By analyzing the stubs candidate to be a use case represented in UCMs and a relation among others. If the relation between one stub and another stub represents a *uses* relationship, then it show us that the use cases derived from these stubs are in *uses* relationship among each other.

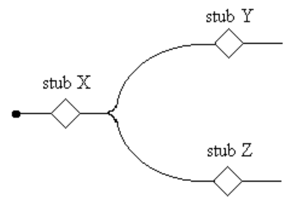

Figure 4. Uses relationship between stubs

Figure 4, shows that stub X and stub Y will be executed, or stub X and stub Z will be executed. In other words stub X may be copied with stub Y and stub Z. Then it can be derived a uses relationship between stub X and stub Y and also a uses relationship between stub X and stub Z. From the previous figure 4, it can be derived from stub X, stub Y and stub Z , use case X, use case Y and use case Z. There is *uses* a relationship between use case X, and use case Y and also a *uses* relationship between use case X and use case Z as illustrated in figure 5.

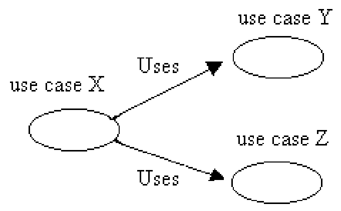

Figure 5. Uses relationship between use cases

3.1.2. Extends Relationship

An extends relationship is used when a use case needs an extra functionality from another use case [9]. By examining the stub candidate to be use case in the UCMs it was found that if two stubs are in an extends relationship, this will show us to that the use case derived by these stubs are in an extends relationship.

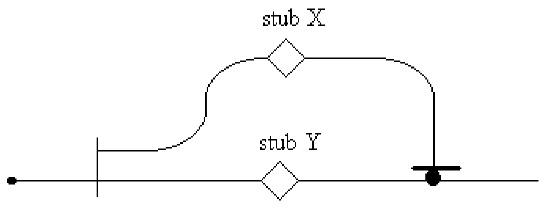

Figure 6. Extends relationship between stubs

From the scenario in figure 6 it was noticed that, stub X will be executed, then at the waiting place, the scenario stops until stub Y is executed, then resumes execution. In other words the execution of the scenario stops after stub X and it resumes the execution only if stub Y executed. This guides us to derive an extends relationship between stub X and stub Y, as it is illustrated in the figure 7.

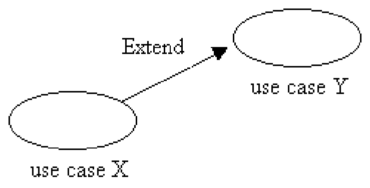

Figure 7. Extends relationship between use cases

3.2. UCMs and Class Diagram

A class diagram describes the static structure of a system. The components in UCMs describes the static structure of the system. A component is composed of a name, and responsibility, that explains the operations inside the component. There was a relationship noticed between the class and the component which represents an internal entity to the system. Each has a name, and each explains the static structure of the system. Also it was noticed that from the responsibility in the component some class's operations can be derived. Also details for use case, derived from UCMs, help in deriving class operation. From these operations it can be derived some of the class's attributes. For example figure 8, shows a component named customer, which has two responsibilities: check customer, and create customer.

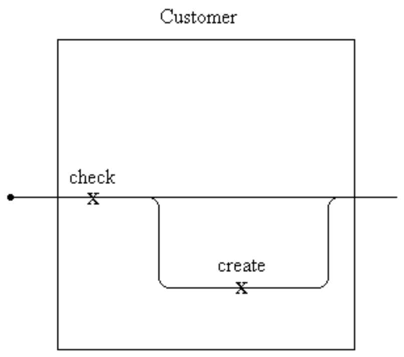

Figure 8. Component has two responsibilities

Two operations can be derived from the component responsibilities *create customer*, and *check customer*. There is no direct representation to the attribute. Howevere, the attribute can be derived from the operation. For example, it can be derived from operation create customer some attributes related to the customer like *the customer's name* … etc. So it can be represented the class diagram as in figure 9.

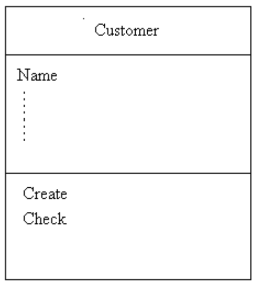

Figure 9. Class diagram derived from component

The path segments that connect two components candidates to classes may guide us to the *association* relationship between these classes. Also by understanding the UCMs, and analyzing responsibilities, and preconditions, it may be derived the *multiplicity* of the association relationship between classes.

3.3. UCMs and Object Diagram

Object diagrams represent static snapshots of instances of things found in class diagrams. They show class instance with data value. The object diagram can be derived from the *component*, if data is available (e.g *attribute in the class*) in the component. If data is not available in UCMs, it can be derived the object diagram from the *class diagram*. Here there is a clear example of traceability. Because from UCMs the class can be derived and from the class the object can be derived as it shown in figure 10.

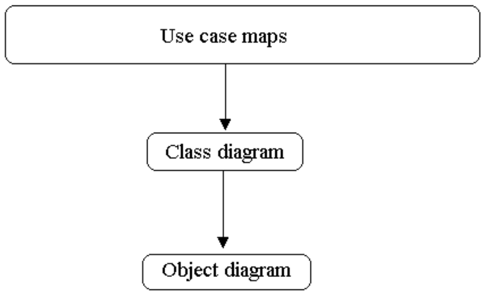

Figure 10. Relationship between UCM and object diagram

3.4. UCMs and Interaction Diagram

An interaction diagrams model the behavior of use case, by describing the set of objects which interact to complete the task. Two kinds of interaction diagrams are sequence, and collaboration diagrams. We have to focus mainly on sequence diagrams. Sequence diagrams display a set of objects, and the messages sent and received among them with respect to the order of these messages. Sequence diagrams generally show the sequence of events that occur. UCMs describe a causality relationship between responsibilities, in other words when responsibility X is executed that causes the execution of responsibility Y. However, without any messages exchanged between the components. This will prompt us to derive the sequence diagram in such a way that message X will have to be executed first, then message Y will be executed. The sequence of responsibilities in UCMs will lead us to the sequence of messages in the sequence diagram.

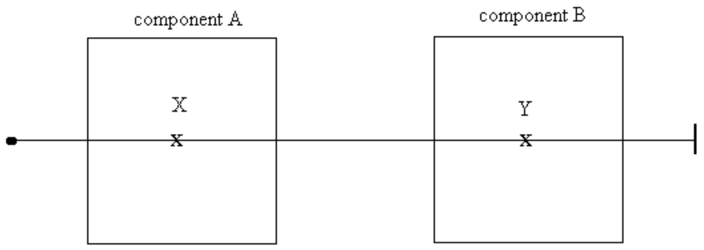

Figure 11. Simple UCM with has two components

Figure 11, shows the scenario executed responsibility X in component A first, then it executed responsibility Y in component B. This shows us that in a sequence diagram, object A will execute the message X first, and then object B will execute the message Y . This is can be done in different ways:

1) Object A executes message X and sends a message to object B to execute message Y as shown in figure 12.

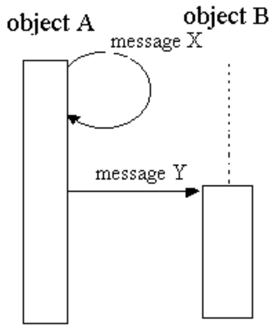

Figure 12. Sequence diagram has two objects

2) Object B sends a message to object A to execute message X, and object B executes message Y as shown in figure 13.

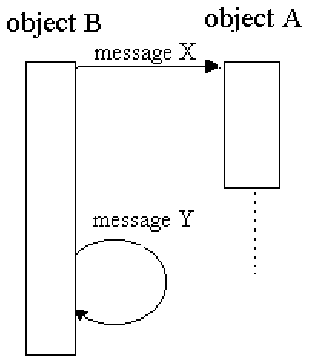

Figure 13. Sequence diagram has two objects

3) Another object C sends a message to object A, to execute message X, and object C also sends a message to object B, to execute message Y, as shown in figure 14.

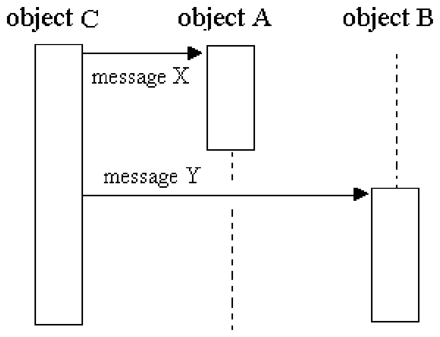

Figure 14. Sequence diagram has three objects

From these ways object A executes message X, and then object B executes message Y. The designer can recognize the messages sent between the objects by analyzing the responsibility in UCMs, and by analyzing the derived details for use case and the derived operation in the class.

3.5. UCMs and State Diagram

State diagrams are used to describe the behaviour of a system. State diagrams describe all of the possible states of an object as events occur. The current state of an object is a result of the events that have occurred to the object. All state diagrams begin with an initial state of the object. In the UCMs scenario the responsibility is considered an event which affects the component state. Since the movement from one responsibility to another affects the components state. We have to specify the state before and after each responsibility on the path segment. This helps the designer in forming the state diagram. Also in the sequence diagram, it can be specified the state before and after each operation, to help designer in forming the state diagram. This example explains how the UCMs help in forming the state diagram. For example we have a UCMs scenario for a car rental. The scenario begins by gathering information about the new customer and making the booking operation. Then the customer may pick up the car and return it back after rental time ends or cancels the booking as it is shown in figure 15.

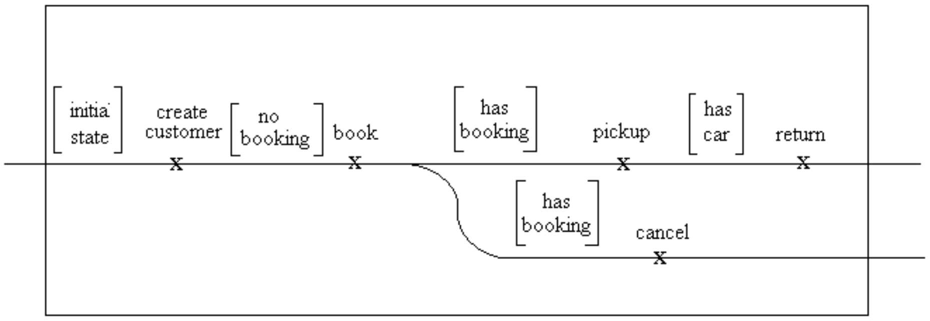

Figure 15. UCMs illustrated with states

To form the state diagram from the previous UCMs scenario, we have to specify the states on the path as states in the state diagram. And specify the responsibilities as events cause the movement from one state to another. All of these guide us to design the state diagram from the previous UCM scenario as it is shown figure 16.

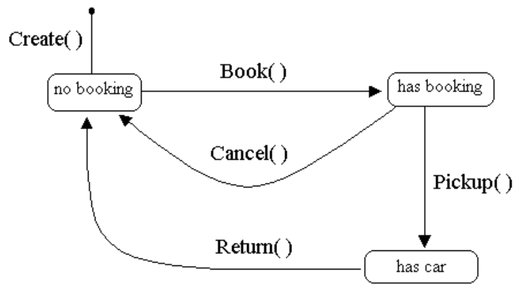

Figure 16. State diagram derived from UCMs

3.6. UCMs and Activity Diagram

An activity diagram is a variation of a state diagram that focuses on a flow of activity driven by internal processing within an object, rather than by events that are external to it. Activity diagrams show the flow of activities through the system. An activity diagram can be divided into swimlanes. Each swimlane represents one focus of responsibility in the activity, and each may be handled by a distinct operation in one or more objects. The order of the swimlanes has no significance . Each action is assigned to one swimlane. There was a relation between the responsibility in the UCMs and the activity in the activity diagram. Both of them focus on sequence of action. From this it can be derived a very strong relationship between the UCMs scenario and the activity diagram. It is possible to represent responsibility in the UCMs scenario as an activity in the activity diagram [4], And the stub in UCMs scenario can be represented as an activity which has its own activity diagram. The components in the UCMs scenario are represented as swimlanes in the activity diagram. For example we can have a UCMs scenario with a component A, component B, and responsibilities X, Y and Z as it is shown in figure 17.

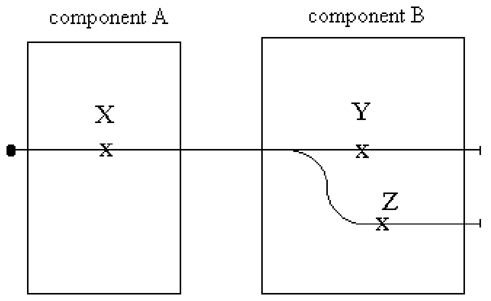

Figure 17. UCMs has two components

Component A and component B can be represented by swimlane A and swimlane B, and responsibility X in component A by activity X in swimlane A and responsibility Y, Z in component B by activity Y, Z in swimlane B. By representing the path between the activities from the previous UCMs scenario, the activity diagram was obtained as it is shown in figure 18.

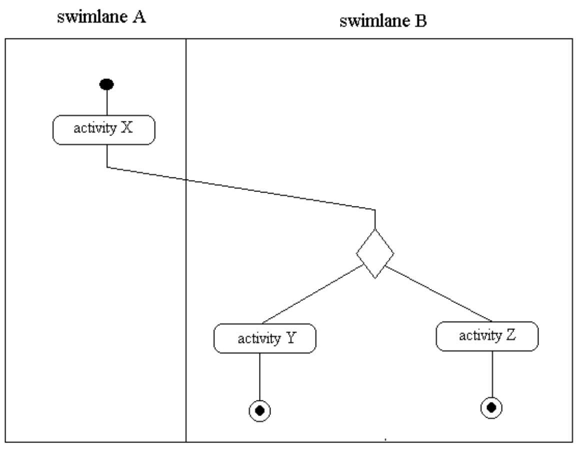

Figure 18. Activity diagram derived from UCM

5. CONCLUSION

The relation between UCMs and UML diagrams was described. This relation provides a systematic approach for generating UML diagrams from UCMs. UML uses several diagrams in developing systems. These diagrams allow transition between requirement and implementation by combining the use of different types of diagrams to express different views of the system. These diagrams are constantly modified according to the changes in the system. The lack of traceability between UML diagrams makes the changes difficult and expensive. The lack of consistency between the diagrams makes them appear as if they are not related. This paper proposed an approach to bypass these difficulties, by defining a traceability relationship between UCMs and UML diagrams, it facilitates the maintains and ensure the consistency between the diagrams. It used UCMs as the starting point for generating UML diagrams (behavior and structure). Because it is suitable for high level visual representation and their ability to simply successfully depict the design of complex system. The derivation of UML diagrams starts by deriving use case diagrams from UCMs. Our approach explains two ways of generating UML diagrams. One way is to generate directly from UCMs the UML diagrams. The other way is generate the UML diagrams from UCM and other UML diagrams. The result of this paper was the generation of UML diagrams from UCM and to solve the lack of traceability problem between UML diagrams.

We recommend that the following points are to be taken into consideration for other scholars in their future work:

- Representing data in UCMs to derive the object diagram from UCMs directly.
- Studying the possibility to derive the implementation diagrams from UCMs.
- Implementation of the tool to support this approach. This tool allows the designer to observe and control the movement between diagrams.

REFERENCES

[1] A. Miga, D. Amyot, F. Bordeleau, D. Cameron, M. Woodside, Deriving Message Sequence Charts from Use Case Maps Scenario Specifications, Tenth SDL Forum (SDL'01), Copenhagen, Denmark, 2001

[2] D. Amyot, G. Mussbacher, Bridging the Requirements/Design in Dynamic Systems with Use Case Maps, Tutorial in: 23rd International Conference on Software Engineering (ICSE'01), Toronto, Canada, 2001

[3] D. Amyot, G. Mussbacher, On the Extension of UML with Use Case Maps Concepts, In: <<UML>> 2000, 3rd International Conference on the Unified Modeling Language, Yourk, UK, 2000

[4] D. Amyot, Use Case Maps and UML for Complex Software Driven System, Technical Report, August, 1999

[5] F. Bordeleau, D. Cameron, On the Relationship between Use Case Maps and Message Sequence Charts, In: 2nd Workshop of the SDL Forum Society on SDL and MSC (SAM2000), Grenoble, France, 2000

[6] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison Wesley, 1999

[7] M. Elammari, W. Lalonde, An Agent Oriented Methodology: High Level and Intermediate Models, Proceedings of Agent Oriented Information Systems, Heidelberg, Germany, 1999

[8] R.J.A. Buhr, R.S. Casselman, Use Case Maps for Object-Oriented Systems, Prentice Hall, 1996.

[9] S. Bennett, S. McRobb, R. Farmer, Object Oriented Systems Analysis and Design using UML, McGraw-Hill, UK, 1999

[10] T. Cockram, R. Parker, D. Tiley, H. Woodward, J. Smith, A. Vickers, A System Requirements Traceability Model: An Industrial Application, Proceedings of SSS'98 - Safety Critical systems Club sixth annual symposium, Birmingham, UK, 1998

[11] Y. He, D. Amyot, A. Williams, Synthesizing SDL from Use Case Maps: An Experiment, In: 11th SDL Forum (SDL03), Stuttgart, Germany, 2003

Authors

Dr. Ahmed Lawgali, he obtained PhD in computer science from Northumbria University, Newcastle Upon Tyne, UK in 2013. His PhD thesis was Investigation of Arabic Handwriting Recognition Based on Segmentation. He joined College of Arts and Sciences (Al Abyar) – Benghazi University in 2007 as an assistant lecturer, being prompted in 2013 to a lecturer and a head of computer science department. In 2016, Lawgali joined Faculty of Information Technology - Benghazi University as a lecturer. Lawgali has published several scholarly articles in the area handwriting analysis and recognition. His research interests lie in the area pattern recognition, handwriting analysis and recognition, document recognition, biometrics, and software engineering.