The Nucleus ***59****, No. 2 (2022) 40-47*

https://doi.org/10.71330/thenucleus.2022.1157

www.thenucleuspak.org.pk

The Nucleus

ISSN 0029-5698 (Print)

ISSN 2306-6539 (Online)

Transformation of UML Diagrams based on their Overlapping: An Algorithmic Approach

M. Rehman[1*], S. Ullah[1], A. Siddique[2]

[1]*Department of Computer Science, Khwaja Fareed University of Engineering and Information Technology, Rahim Yar Khan, Pakistan*

[2]*Department of Computer Science and Information Technology, Ghazi University, Dera Ghazi Khan, Punjab, Pakistan*

ABSTRACT

Modeling methods in general and Unified Modeling Language (UML) in particular, have gained trust in software developer communities. UML diagrams are profoundly utilized by software developers to build, model and visualize the working of the software. While the comprehension of the UML models can introduce ambiguities in the software. As there is no specification on how these models should be analyzed. Therefore, these models sometimes convey inconsistent semantics in the software systems. So, there must be some automated procedure in which these models be transformed into one another, to remove analyst-introduced ambiguities. For this purpose, the conversion between UML diagrams is a necessity. This paper aims at solving this inconsistency gap between UML diagrams by providing a novel approach to conversion between UML models. In this paper, we have proposed novel techniques to convert 1) UML Use-Case diagram into UML Communication diagram. 2) UML Communication diagram into UML Component diagram. The said approach is validated using two case studies.

Keywords: *UML, Use-Case, Communication Component, View Models*

1. Introduction

UML as its name suggests is used to model, document & visualize software systems and Object-Oriented systems. It provides a standard for making software blueprints, from conceptual modeling to behavioral modeling. The conceptual modeling paves way for business processes handling like programming language constructs, schemas and over-the-shelf code. In the same way, behavioral modeling resolves the behaviors of data flow, activity flow, methods handling and complexity handling of business processes. UML tools have always proved their best in handling complexity in terms of software design and development [1].

UML has emerged as a course of study for university students and a lot of research is done in this field. There are a variety of diagrams in UML to model different aspects of the system. Use-case diagram for the handling of functional requirements, a communication diagram for captivating message passing between classes and a Component diagram for dealing with various structural components [2, 3].

During the system development phase, there are many aspects/views of the system made under different models. If these views vary or show different systems at any abstraction level, the system formed would be inconsistent. Therefore, the temporal view of the complete system should not be varying at any point in time. Such inconsistencies/contradictions introduced during the design phase have a paramount effect on software projects [4]. For example, a particular Use-Case diagram of an ATM system shows that the system includes pin verification to the limit of three from the user. If the user fails to insert the correct pin, his/her card will be blocked and an automated alert be sent to his/her number. The same systems Class diagram shows all methods, classes and objects of the ATM system but failed to include the three-attempt verification. The mistake goes along in other diagrams and finally, a system is built without the three-attempt verification. The developer holds the designer for the mistake and the designer holds the committee who approved the design. This type of security inconsistency proves to be fatal/devastating for the companies. Among various real-time examples of software design issues, one happened in 2018, when a fighter plane system was developed to detect targets and respond accordingly. But the jet can only detect a target correctly, if more than 2 targets befall at the same time, they are detected as no target, making it a multifaceted software design failure [5].

Currently, no technique exists to transform UML diagrams into one another or to transfer any of the diagrams into code that has formal semantics [6]. Many authors have however, proposed and given algorithms to transform between the diagrams, as is specified in SLR table 1. State charts and sequence diagrams are made from Use-Case diagrams [7]. UML diagrams are made by a combination of Simple English and OCL (Object Constraint Language) [8]. These languages although rigorously trying to bring meanings into the diagrams, at the same time introduce ambiguity in the design of the system as:

a. There is no formal way to analyze and verify the diagrams.

b. The graphical models lack transformation details, i.e., they cannot be transferred into one another by any proven method.

c. As natural language is used in modeling, these diagrams sometimes give a two-way meaning, which introduces ambiguity in the software implementation.

These inconsistencies between UML diagrams like Use-Case, State and Sequence diagrams have always remained a major focus of researchers. UML models are translated into formal languages to avoid inconsistency problems described

[*]Corresponding author: madiha.rehman@kfueit.edu.pk

earlier [9]. Some authors have claimed that these inconsistencies between the UML diagrams are due to the overlapping between various diagrams of UML. As is said by [10, 11] the model would remain consistent if no overlapping of elements exists.

UML diagrams are also used to depict multifaceted systems and describe their dynamic and static behaviors. This type of analysis requires that the modeling used must be unambiguous and cope with the complexity of the system [12].

In this study, we have proposed two algorithms that can be used to transform the use-case diagram into the component diagram, using the communication diagram as a mid-path. Separate algorithms are proposed to handle each diagram's complexity. Two different case studies are incorporated, to validate the algorithms.

2. Literature Review

In today's world, the software is becoming more and more complex. It is being accepted by the software industry that it is not possible to test all aspects of the software [13]. It is thus required to maintain a quick balance between the software industry and customers by keeping the complexity level very low [14]. The OMG (Object Management Group) Modeling and MDD (Model Driven Development) are becoming more of a drift. Model Driven architecture is taking us towards transformations. These transformations between a set of models will ultimately lead to better software development. MDD focuses on software development as a queue of model transformations from requirements engineering to analysis, design, implementation and deployment [8].

The software industry has always been fonder of software assessments and assurance. Very less work has been done in the field of model quality and their concept is very poorly understood. Although Modeling is used from the very early stage of the system and software and moves until the last stage of the system is achieved [15].

From the emergence of UML in the late 1990s and towards 2007, modeling language issues dominated the software industry. UML has gone through significant changes in its semantics and metamodeling. In recent times, Researchers and the software industry found more interest in MDD (Model Driven Development) and MDA (Model Driven Architecture).

Similarly, developers seemed to be fonder of OCL (Object Constraint Language) and QVT (Query View Transformation). In January 2020, a group of researchers listed the challenges faced by the software industry in terms of Modeling. These include model analysis and verification, models at runtime, modeling databases and scalability issues [16].

In 2005, a study was conducted on the quality and implementation of conceptual models. The study was based on finding ways for model structure, validation structure and quality insurance. This study reported issues like the indetermination of scope, origin and ingenuity of structural models [17]. A group of researchers worked on finding the metrics for the UML Class diagram. They have reported a range of metrics that can be used to measure various UML Models, like size, complexity, coupling, cohesion, etc. These metrics can be ranged from UML Models to OO design models [18].

In 2009, a group of authors combined to examine the consistency level between UML Models. The study was completed with the following facts:

a. There exists a serious gap in the level of consistency between the various diagrams. And this gap covers all the problems ranging from requirement gathering to development errors [15].
b. The study also concluded that these gaps need serious formal language for consistency management [19].

A paper was presented to describe a rule production system [20]. Another paper proposed algebraic expressions to conquer inconsistencies [21]. A Study on the language view of the UML Models proposes to use only the grammar productions specified with syntax to be used with UML Models. The study further describes the language to eliminate inconsistency using XMI language [22].

Our literature review is based on the study of journal papers and conference proceedings to find out the potential problems and inconsistencies in the UML models. The problems were identified to streamline the process of Software design and development. The systematic literature review is provided in table 1.

Table 1: Systematic Literature Review

Sr.No	Paper Name	Journal / Conference, Year	Methodology
1	Structural and Semantic Similarity Measurement of UML Use-Case diagram [25]	Journal, 2020	Nazir et al have addressed the software artifacts (diagrams) reuse. They measured the structural and semantic similarity between Use-Case diagrams by using Graph Edit distance. In this technique, they converted the actors and Use-Cases into graphs and then measured the distance between edges. For semantics similarity, the authors have proposed the use of Word Net and WuPalmer techniques. The evaluation of the proposed solution is made by comparing similarity values between Pearson experts and the authors' results.
2	Automatic Transformation of User Stories into UML Use-Case diagrams using NLP Techniques [26]	Conference 2018	Meryem et al have addressed the advantage of the Use-Case diagram in terms of Requirements gathering. As the use of Agile technologies is a trend in the current scenario, the authors have proposed the conversion of User stories in to Use-Case diagram. In such an automatic way, no user story would be left out.

3	Verifying the Consistency of UML Models [13]	Conference 2016	The consistency of UML diagrams is verified by converting the consistency rules into constraints. These OCL constraints are then converted to a plug-in to check the UML models against these constraints.
4	Identification and check of inconsistencies between UML diagrams [27]	Conference, 2013	Xianhong has discussed the inconsistencies between various UML diagrams and has devised 13 rules to check the consistency. The author has also proposed to check for consistency problems manually or dynamically.
5	A framework for reuse of multi-view UML artifacts [28]	Journal 2013	The authors have proposed a mechanism to reuse UML artifacts. The mechanism would be carried out in 4 steps, the pre-filtering stage: the UML artifacts would be gathered with similar requirements. the multi-view retrieval - the requirement specifications would be matched and ranked with the previous artifact's requirements. After the two stages, the requirements would be adapted and integrated with our existing system.
6	Comparative Study on DFD to UML diagrams Transformations [6]	Journal 2011	The authors have proposed the transformation of DFD (Data Flow diagram) into UML diagrams. The Level 1 DFD can be converted to Use-Case, Level 2 DFD can be converted to interaction, while level 3 DFD with the integration of the E-R diagram can be converted to the Class diagram. The author has also explicitly stated that this conversion is tool free.
7	A systematic review of UML Model consistency management [19]	Journal, 2009	A study was conducted to examine the consistency level between UML Models. The study was completed with the following facts: There exists a serious gap in the level of consistency between the various diagrams. And this gap covers all the problems ranging from requirement gathering to development errors. The study also concluded that these gaps need serious formal language for consistency management.
8	Model Transformation in Software Performance Engineering [29]	Conference, 2006	The authors have discussed various Model Driven Engineering approaches to transform software models. They have discussed the Petriu approach to transform UML collaboration into sequence and Activity diagrams. Other frameworks discussed include Software Performance MDA Framework, PIPM, PSPM and SPMDA horizontal & vertical transformations.
9	Theoretical and Practical issues in evaluating the quality of conceptual models: current state and future directions [17]	Journal, 2005	Daniel et al have addressed the quality issues of conceptual models. The authors have suggested 12 issues in the quality of conceptual modeling like lack of consistency, lack of knowledge, no focus on product, no empirical testing, etc. In a nutshell, the problems with UML diagrams consistency were reviewed.
10	A Survey of Metrics for UML Class diagrams [18]	Journal, 2005	A group of researchers worked on finding the metrics for the UML Class diagram. They have reported a range of metrics that can be used to measure various UML Models, like size, complexity, coupling, cohesion, etc. These metrics can be ranged from UML Models to OO design Models.
11	Transformations Between UML diagrams [30]	Journal, 2003	Petri et al have used the overlapping in UML artifacts to transform the diagrams. They have suggested the common UML diagrams that have the most overlapping features be transformed into one another. As Sequence diagram can be transformed into a Class and State chart diagram. Sequence and Collaboration diagrams can be fully transformed. While no tools are used for the transformations. Their overlapping content is used for transformations.
12	An Integrated Semantic for UML Class, Object and State diagrams Based on Graph Transformations [31]	Conference, 2002	Sabine et al have addressed the software diagrams inconsistencies by proposing graph transformations. The authors have used UML Class, Object and State diagrams and converted them into graph transformations. The graph transformations can then be used for the semantic similarity between diagrams.
13	Integration and Transformation of UML Models [32]	Conference, 2002	The authors have discussed the relation between different UML models. The authors have proposed that the relation can be more technically understood if two or more models be converted to one another like a sequence diagram can be converted into state charts.
14	Verifying the Consistency of UML Models [13]	Conference 2016	The consistency of UML diagrams is verified by converting the consistency rules into constraints. These OCL constraints are then converted to a plug-in to check the UML models against these constraints.

3. UML Diagrams

3.1. *Use-Case diagram*

The Use-Case diagram is used to capture the functional requirements of a software system. It describes how a system would work. For example, how a user interacts with the system and what are the system's functionalities [23]. The main artifacts of a Use-Case diagram are:

a. *Use-Case:* Use-Cases describe a piece of behavior /action executed by the system.
b. *Actor:* Any system or person that directly or indirectly interacts with the system.
c. *Interactions:* Interaction between the Use-Cases and the actor shown with straight lines

For our study we have chosen the Use-Case diagram for validating our point due to the following reasons:

i. The Use-Case diagram is the first diagram made to capture the system's requirements. All other diagrams are made after the Use-Case.
ii. As Use-Case diagram is a behavioral diagram that shows the interactions of all systems/users happing within the system.

The only UML diagram that includes possibly all the requirements initially gathered for the system.

3.2. *Communication diagram*

The Communication diagram also previously known as the Collaboration diagram handles interactions between objects/parts of a system. Communication diagrams are used when a mix of information from Class, Use-Case and Sequence diagram is required [24]. This diagram is used to handle the behavioral aspect of the Use-Case diagram [24]. It consists of three major components:

a. *Objects:* The Classes in the scenario that depict major objects between which communication is held.
b. *Messages/Actions:* The communication between objects, diagrammed as Messages with sequential numberings. The Messages can be synchronous or asynchronous.
c. *Actor:* Any system or person that directly or indirectly interacts with the system. An Actor can be another system or a person.

For this study communication diagram is chosen due to the following reasons

i. A communication diagram is an interaction diagram that is very close to the class diagram in general.
ii. The Use-Case/actions in the Use-Case diagram can easily be mapped with objects in the Communication diagram.
iii. Both these diagrams help show the behavioral image of a system.
iv. The communication diagram being very close to the class and Use-Case is also proven to share similarities with component diagram.

3.3. Component diagram

The Component diagram being the Structural diagram of UML is used to model the structural elements of the system. This diagram is suitable to visualize the components of a system and its dependencies. It consists of three main artifacts:

a. Component: The objects of the systems are represented as Components.
b. Interface: Operations performed by the components are shown as Interfaces.
c. Dependencies: This shows how one component requires another component. The possible interaction between the two.

The component diagram is chosen due to the following reasons:

i. The component diagram is close to the communication diagram, as both diagrams focus on identifying components/classes and then identifying the message passing between the two.
ii. The component diagram is a structural diagram, therefore, to show the transformation between behavioral and structural UML this is chosen.
iii. The components from component diagram can easily be merged with objects in the communication diagram.
iv. The component diagram is very close to the class diagram. Classes represent logic, while component diagrams implement Logic.

4. Problem Statement

In 2018, a software glitch in fighter air crafts was detected, as the fighter jet was unable to detect multiple targets, it was a huge software fail and fighter planes were called off. The case was opened and the developer and analyst found that detecting an approaching target was modeled, which caused a semantic misunderstanding [5].

In a nutshell, all the issues are listed for briefing:

4.1. Gap between diagrams

As UML diagrams are made to depict the structural and behavioral components of a system. The relation between the diagrams lacks. Due to this lacking the software systems are severely affected. Software developers most often use class, activity, sequence and Use-Case to code their systems. A method from the class diagram lacking in activity cannot be coded and hence produces a serious error in the software. These types of errors are more logical and thus cannot be found during the testing process.

4.2. Transformation

Each diagram has its behavior. For a single system, it cannot be ensured that one model is completely transformed into other respective models.

4.3. Inconsistency

When diagrams convey wrong semantics, they are said to be inconsistent. We cannot rely on the models for system verification. The Inconsistency between diagrams can be elaborated as: one diagram features a static method, while the other diagram for the same system ignores the static feature and introduces another behavioral component.

4.4. Verification

The semantic inconsistency between diagrams produces ambiguities in the overall system. Thus, the system fails in the verification phase when the customer tries to verify the system on his/her requirements.

5. Proposed Methodology

As discussed earlier the gap between UML diagrams can introduce serious problems in software validation. In proposed methodology following steps have been devised:

i. Conversion of Use-Case diagram to communication diagram with the help of the proposed Algorithm 01.
ii. Conversion of communication diagram to component diagram with the help of the proposed Algorithm 02.

The Block diagram of the proposed methodology is shown in Fig. 1. While the complete methodology diagram in Fig. 2, describes the level-to-level conversion between the elements of all diagrams.

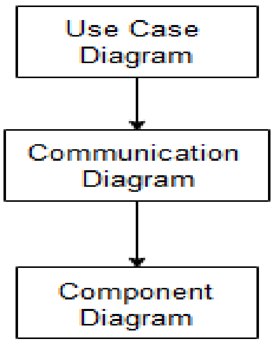

Fig. 1: Block diagram of Proposed Methodology.

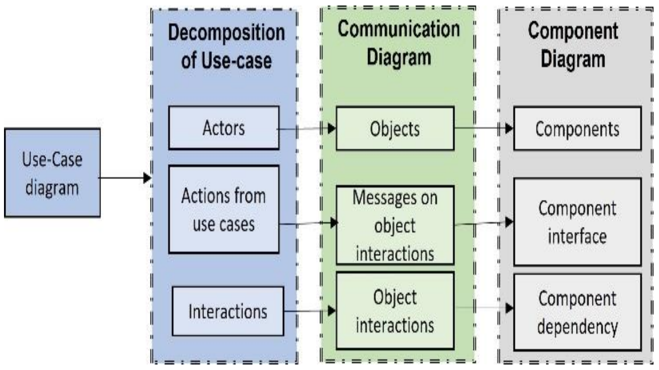

Fig. 2: Detailed Methodology diagram.

To convert the Use-Case diagram to a communication diagram, we have used the steps given in the pseudocode below as Algorithm 01. The algorithm takes a Use-Case diagram as input, decompose the diagram into pieces (Actors, interactions and Use-Cases) and one by one converts the Actors to objects in line 7 to 11. If any Actor has sub actors, for example, a customer can be a manager or an employee, lines 12 to 17 convert this type of interaction into object interactions. Line 18 to 21 transfer the actions into object messages.

Algorithm 01- Pseudocode that builds Communication diagram from Use-case diagram

```
1:  Start
2:  Input: {Use-Case diagram}
3:  Actor ← actors from input
4:  Message ← Usecases/Actions from input
5:  Object ← null store for communication diagram
6:  Boolean: isinherited
7:  If (∃ Actor)
8:  For each (Actor a)
9:  Object a ← Actor a
10: End for
11: End if
12: If (Actor a connected to Actor b)
13: Isinherited ← true
14: For each (Isinherited)
15: Object a ← get connected to object b
16: End for
17: End if
18: For each (Message)
19: Object a.Message ← Actor a. usecase
20: End for
21: end
```

The pseudo-code for proposed Algorithm 02, given below is used to convert any communication diagram to a component diagram. The Algorithm takes as input a communication diagram and decomposes the diagram into objects, Messages and Message directions. Line 7 to 11 is used to convert objects to components. Line 12 to 17 makes connections between the components. Line 18 to 25 checks the Message directions of the messages. If Messages are sent from one object to another, then an interface is required by the sending object and vice versa.

Algorithm 02: Pseudocode that builds Component diagram from Communication diagram

```
1:  Start
2:  Input: {communication diagram}
3:  Object ← Objects from communication diagram
4:  Messages ← Message interactions between objects
5:  Message direction ← Send or Receive
6:  Component ← null store for component diagram
7:  If (∃ Object)
8:  For each (Object a)
9:  Component a ← object a
10: End for
11: End if
12: If (object a connected to object b)
13: Isconnected ← true
14: For each (isconnected == true)
15: Component a ← get connected to component b
16: End for
17: End if
18: For each (Message direction)
19: If (Message direction == Send)
20: Component require interface
21: Else
22: Component provide interface
23: End if
24: End for
25: end
```

The proposed algorithms are universal techniques of conversion that can be implemented in any programming language. This provides the ease of platform independency, as any platform, language construct can be utilized to implement this study.

6. Validation with ATM System Case Study

We have used ATM (Automated Teller Machine) case study because it's easy to understand. The Use-Case diagram shown in Fig. 3 shows the main functional requirements of an ATM system.

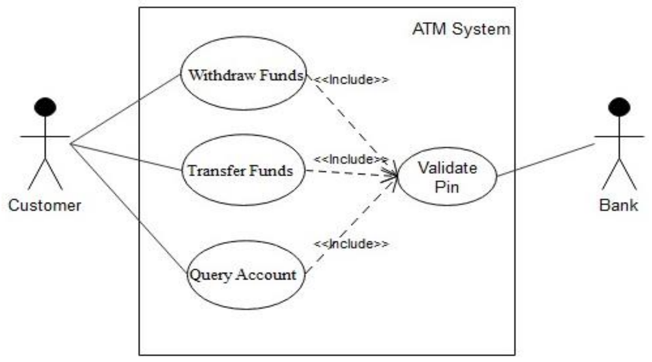

Fig. 3: Use-Case diagram of an ATM Machine.

The Use-Case comprises of

1. User / Customer of the system,
2. The System,
3. A set of actions/Use-Cases the user can perform.

The Communication diagram of the system shown in Fig. 4 uses

1. Users / Customers of the System
2. The system
3. Set of Message passing between the components.

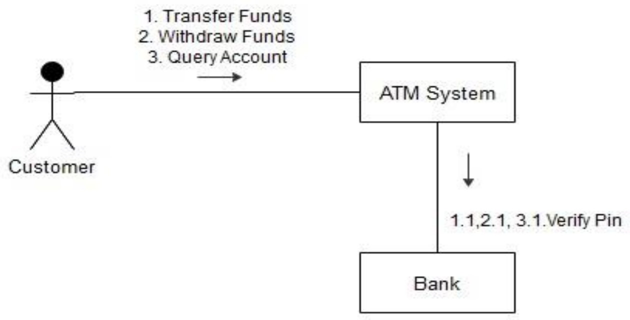

Fig. 4: Communication diagram of ATM Machine.

The conversion with the use of our proposed Algorithm 01 is simplified in table 2, where the actors are converted to objects and Use-Cases are converted to a set of interactions between the objects.

Table 2: Conversion table of Use-Case diagram

Use-Case diagram	Communication diagram
Actors	Objects
Customer ATM Bank	Customer ATM Bank
Use-Cases	Interactions
Withdraw Funds Transfer Funds Query Account	All Use-Cases will be converted into communication messages

The communication between the Actors and Use-Cases are made clearer and sounder with the help of communication messages. This is the one reason to choose a communication diagram as a mid-path between the Use-Case and component. The behavioral Use-Case diagram lacks communication messages, while the Communication diagram includes sequential listings of messages to enhance readability. Conversion table 2 clearly shows the relation between the Use-Case and Communication diagram. Our study is focused on the conversion of a Behavioral diagram into a Structural diagram.

So, we will be using the Component diagram (Structural diagram) into which the behavioral constraints of the communication diagram can be easily transformed as shown in Fig. 5.

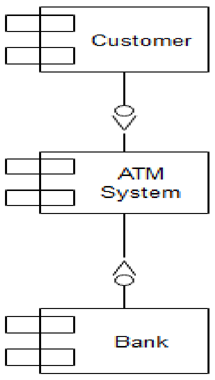

Fig. 5: Component diagram of ATM Machine.

The Component diagram clearly shows how the structural components of an ATM Machine collaborate. The three main components remained the same as in Use-Case, Communication diagrams shown earlier. However, the other artifacts shown in Fig. 5, depict the various structural dependencies of the system.

The conversion table for Communication diagram is given in table 3.

Table 3: Conversion table of Communication diagram.

Communication diagram	Component diagram	
Object	Component	Dependency
Customer ATM Bank	Customer ATM Bank	Dependency is the interface requirement of the components. It is calculated from the Message Directions from the Communication diagram.

Table 3 shows the elements of the Communication diagram on the left side and the elements of the Component diagram are shown on the right side.

Therefore, it is made clear with the figures and conversion tables that systematic transformation helps reduces the inconsistencies and ambiguities between the diagrams.

7. Validation with Cellular Network Case Study

The second case study we have used in our system is a cellular network. The System only incorporates the Phone and

Message passage between the user and network. The Use-Case diagram of the system is given in Fig. 6.

With the Use-Case diagram and our proposed Algorithm 01, we get the communication diagram in Fig. 7.

With the communication diagram and our proposed Algorithm 02, we get the component diagram in Fig. 8.

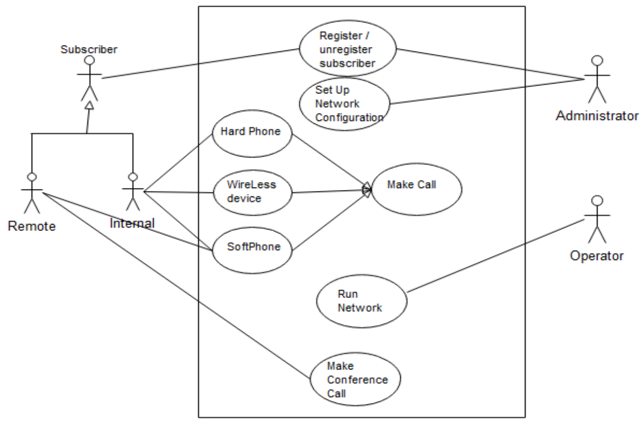

Fig. 6: Use-Case diagram of Cellular Network.

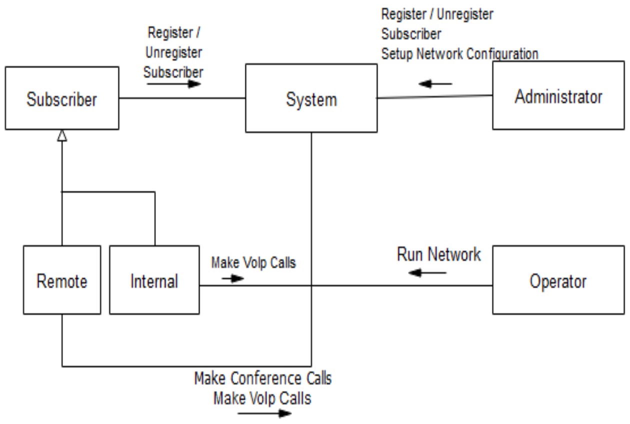

Fig. 7: Communication diagram for Cellular Network.

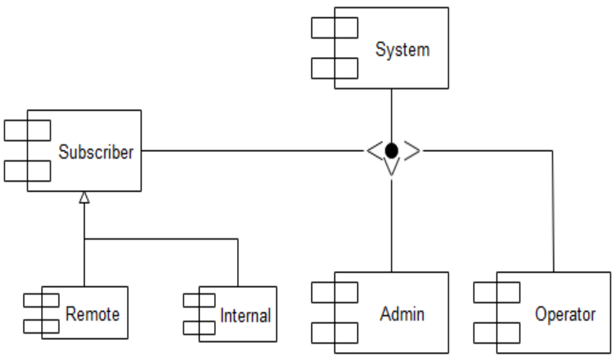

Fig. 8: Component diagram of Cellular Network.

8. Conclusion

In this paper, we have provided a novel universal algorithm that converts one UML diagram to another automatically. This study focuses at converting the Use-Case diagram into a component diagram. Communication diagram is chosen as a mid-path. This conversion shows that the gaps and inconsistencies between the UML diagrams can be fulfilled if proper procedures are followed for going from one diagram to another. Similarly, the data flow inconsistencies, verification and validation problems can also be removed. The automatic conversions between diagrams in software engineering will be helpful with the multifaceted system, where a slight inconsistency in diagram structure will yield poor effects on the overall behaviour of the system. These conversions will be helpful in Software testing mechanisms. The runtime conversions of the case studies have also proven the point.

In the Future, this study can be enhanced by including formal specifications of the UML diagrams.

References

[1] B. Padmanabhan, "Unified Modeling Language (UML) Overview", February, 2012.

[2] I. Jacobson, G. Booch and J. Rumbaugh, “The Unified Modeling Language Reference Manual”, Addison-Wesley, 1999.

[3] S. Shakil and B. Hazela, "Formalization of UML Class Diagram", International Journal of Engineering Science and Computing, vol. 6, no. 5, 2016.

[4] A. Kalnins, J. Barzdins and K. Podneiks, “Modeling Languages and tools: state of the art”, Order 18, no. L12, 2000.

[5] T. Corp., "Tricentis.com" December 2018. [Online]. Available: https://www.tricentis.com/blog/real-life-examples-of-software-development-failures/.

[6] A. Jilani, M. Usman and A. Nadeem, “Comparative Study on DFD to UML Diagrams Transformations”, World of Computer Science and Information Technology Journal, vol. 1, no. 1, pp. 10-16, 2011.

[7] B. Hnatkowska, Z. Huzar and L. Kuzniarz, "Refinement relationship between collaborations", Workshop on Consistency Problems in UML-based Software Development II, 2003.

[8] C. Atkinson and T. Kuhne, "Model driven development: A metamodeling foundation", IEEE Software, vol. 20, no. 5, pp. 36-41, 2003.

[9] N.A. Zafar, "Formal Specification and Verification of Few Combined Fragments of UML Sequence Diagram", Arabian Journal for Science and Engineering, vol. 41, no. 8, pp. 2975-2986, 2016.

[10] S.J. Niepostyn and I. Bluemke, "The Function-Behaviour-Structure Diagram for Modelling Workflow of Information Systems", Advanced Information Systems Engineering Workshops, Berlin, Heidelberg, 2012.

[11] G. Spanoudakis and A. Zisman, "Inconsistency Management in Software Engineering: Survey and open research issues", Handbook of Software Engineering and knowledge Engineering, vol I, pp. 329-380, 2001.

[12] L. Baresi, MM. Pourhashem and M. Rossi, "Flexible Modular Formalization of UML Sequence Diagrams", Proceedings of the 2nd FME Workshop on Formal Methods in Software Engineering, June 2014.

[13] D. Torre, "Verifying the consistency of UML models", IEEE 27th International Symposium on Software Reliability Engineering Workshops, Ottawa, Canada, pp. 53-54, 2016.

[14] D. Thomas, "Revenge of the modelers or UML utopia", IEEE Software, vol 21, no. 3, pp. 15-17, 2004.

[15] M. Nelson and M. Piattini, "A Systematic Literature Review on the Quality of UML Models", Innovations in database design, web

applications and information systems managements, vol. 22, no. 3, pp. 310-334, 2012.

[16] A. Bucchiarone, J. Cabot and RF. Paige, "Grand Challenges in model-driven engineering: an analysis of the state of the research", Software and Systems Modeling, vol 19, no. 1, pp. 5-13, 2020.

[17] D. Moody, "Theoretical and Practical issues in evaluating the quality of conceptual models: current state and future directions", Data and knowledge Engineering, vol. 55, no. 3, pp. 243-276, 2005.

[18] M. Genero, M. Piattini and C. Calero, "A survey of metrics for UML Class Diagrams", Journal of Object Technology, vol. 4, no. 9, pp. 59-92, 2005.

[19] F.J. Lucas, F. Molina and A. Toval, "A systematic review of UML Model consistency management", Information and Software Technology, vol. 51, no. 12, pp. 1631-1645, 2009.

[20] W.Q. Liu, S. Easterbrook and J. Mylopoulos, "Rule-based detection of inconsistency in UML models", Workshop on Consistency Problems in UML-Based Software Development, vol. 5, 2002.

[21] E. Astesiano and G. Reggio, "An Algebraic Proposal for Handling UML Consistency", Workshop on Consistency Problems in UML based Software Development II, San Francisco, USA, 2003.

[22] Z. Chen and G. Motet, "A Language- Theoretic View on Guidelines and Consistency Rules of UML", European Conference on Model Driven Architecture- Foundations and Applications. Springer, 2009.

[23] D. Rajagopal and K. Thilakavalli, "A Study: UML for OOA and OOD", International Journal of Knowledge Content Development & Technology, vol. 7, no. 2, pp. 5-20, 2017.

[24] MB. Tuncel, "Using Collaboration Diagrams in Component Oriented Modeling", MS Thesis, Middle East Technical University, 2006.

[25] M.N. Arifin and D. Siahaan, "Structural and Semantic Similarity Measurement of UML Use Case Diagram", Lontar Komputer, vol. 11, no. 2, pp. 88-100, 2020.

[26] M. Elallaoui, K. Nafil and R. Touahni, "Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques", Procedia Computer science, vol. 130, pp. 42-49, 2018.

[27] X. Liu, "Identification and check of inconsistencies between UML diagrams", International Conference on Computer Sciences and Applications, Luoyang, China, 2013.

[28] H.O. Salami and M. Ahmed, "A framework for reuse of multi-view UML artifacts", The International Journal of Soft Computing and Software Engineering, vol. 3, no. 3, pp. 156-162, 2014.

[29] A.D. Marco and R. Mirandola, "Model Transformation in Software Performance Engineering", in International Conference on the Quality of Software Architectures, Berlin, Heidelberg, pp. 95-110, 2006.

[30] P. Selonen, K. Koskimies and M. Sakkinen, "Transformations Between UML Diagrams", International Journal of Database Management, vol. 14, no. 3, pp. 37-55, 2003.

[31] S. Kuske, M. Gogolla and R. Kollmann, "An Integrated Semantics for UML Class, Object and State diagrams based on Graph Transformation", International Conference on Integrated Formal Methods, Springer, Berlin, Heidelberg, pp. 11-28, 2002.

[32] J. Araujo, J. Whittle, A. Toval and R. France, "Integration and Transformation of UML Models", European Conference on Object Oriented Programming, Springer, pp. 184-191, 2002.

[33] OM. Group, "Unified Modeling Language Specification Version 2.5.1", https://www.omg.org/spec/UML/2.5.1/, December 2017.