VERIFICATION OF BEHAVIOR PRESERVATION IN UML SEQUENCE DIAGRAMS USING GRAPH MODELS

CHITRA M T
Research Scholar, University of Kerala
Thiruvananthapuram, Kerala, 695581, India
chitra.mt@gmail.com

ELIZABETH SHERLY
Professor, Indian Institute of Information Technology and Management-Kerala
Thiruvananthapuram, Kerala, 695581, India
sherly@iiitmk.ac.in

Abstract.-.The verification of model transformations is gaining significant attention recent years. This paper presents an approach for verifying the behavioral preservance property of UML behavioral models that have been subjected to a model refactoring process. Graph based models have been proposed for model verification to analyze the behavior preservation property of the models that have been refactored.

Keywords**:** model transformation, graph isomorphism, verification, refactoring**.**

1. Introduction

Model based software development is becoming very popular in the software industry with the advancements of the visual modeling languages like Unified Modeling Language. The UML is a standard modeling language which provides several standard notations and diagrams to model different views of a system [1]. The models constructed using UML have conformed to their respective modeling standards. Model transformations are fundamental in model driven architecture. It is a disciplined technique of applying some transformation that can convert source model to a target model by maintaining its behavior preservance property [8]. It helps in transforming the UML models to facilitate various advancements like model refinement, automatic code generation etc. Model Refactoring is a model transformation approach which involves the process of changing its structure while preserving its behavior of a system in order to ensure their overall consistency of the model [5, 6]. Model transformation enables us to employ software models beyond a mere design by using it for various approaches such as system simulation and code generation. It greatly improves the productivity of developers and the quality of the models. In order to determine the correctness of the transformed model and the quality and reliability of transformation process applied, verification of the model transformations is very important.

According to Boehm, Verification is defined as *building the thing right* [3]. This paper concentrates on verifying the behavioral preservance properties of the source UML model with the target models that have been subjected to a model transformation process. Given a source model of a system designed using UML2.0 Sequence diagram and its transformation model obtained through an automated refactoring process, the verification process is being implemented to evaluate the correctness of the model transformation applied in to the design model using the proposed graph model rules.

The work proposes a graph model representation to show the effect of model refactoring on the behavioral model design of complex systems represented using UML2.0 Sequence diagrams and Object Constraint language (OCL) [4]. Graph theory approaches have a distinguished background in terms of their application to model transformation. For a graph-based metamodel M, the set Refactoring, R of the graph describes a set of rules describing how to transform models which are the instances of the metamodel M, thus producing a refactored graph version SD^R of the original model SD. In the case of refactorings, we change the structure of the system elements in order to improve its performance, which will nevertheless alter the overall behavior of the system. Rather, it will only enrich the model elements by associating the validation properties which help in confirming that the model that was built is the right one as it is concerned with getting a transformation's requirement correct.

Behavior Preservation is one of the essential aspects that have to be performed for proving the correctness of the UML behavioral models, as they visualize the dynamic aspects of a system. It states the fact that the behavior of a model remains the same even if we have applied the model transformation.

One of the commonly used approaches for verification of the behavior preservation of UML models is for checking the consistency of the given model with the refactored models. Several formalisms are used for the specifications of different model transformations. The work proposed is an attempt to verify the behavioral properties of the software models designed using UML2.0 sequence diagrams annotated with constraints using graph model representations [7].

2. Preliminaries

Graphs are the most commonly used complex data structure in computer science. They are the natural representations of models that are inherently graph-based in nature, such as the UML diagrams. According to B´ezivin, a model in model driven engineering is defined as a graph-based structure representing certain aspects of a given system and conforming to the definition of another graph called a metamodel [2]. It helps in depicting the various aspects of a system such as their structure, different modeling views, states, behaviors, computations, and several other entities of interest in a two-dimensional space.

Refactoring is one of the major techniques commonly used for code restructuring- a fundamental area of model transformation. The term refactoring was originally introduced by William Odypke in his PhD dissertation where the refactoring process is defined as restructuring an existing model by altering its internal structure without changing its external behavior [6]. A model refactoring approach has been proposed to implement the model restructuring of UML sequence diagrams by associating the static and temporal-safety related constraints into it [4]. Graph models are used here to represent the behavioral modeling for checking the preservation properties of the source UML2.0 sequence diagram model with the target model obtained after the proposed model transformation approach.

Graph models can be used to give unified formalism for the problems with almost every conceivable discipline. Formally, a directed graph (or digraph) G=<V,G> is a pair of non-empty sets of vertices or nodes (V) and a set of directed edges or connection arcs (E), showing the direction flow of communication between two vertices, called its endpoints. Each directed edge is connected with an ordered pair of vertices [9].

Definition 1. *A vertex bijection f: VG → VH between two graphs G and H is structure-preserving if it preserves adjacency and non-adjacency.*

For every pair of vertices in G, x and y are adjacent in G $\Longleftrightarrow$ f(x) and f(y) are adjacent in H. This leads us to a formal mathematical definition of our notion of "same" graph. Hence we can add that two graphs are isomorphic if there exist a structure-preserving bijection.

Formally, two graphs G and H with vertices V_n={1,2,…,n} are said to be isomorphic i.e., G ~= H, if there is a permutation P of V_n such that {x,y} is in the set of graph edges E(G) if and only if {P(x), P(y)} is in the set of graph edges E(H). From any given graph, two graphs namely, the representative graph and the graph obtained after the model transformation process are generated. We should also prove the structural equivalence of these two graphs by stating that it preserves adjacency.

A vertex function f: VG → VH preserves adjacency if for every pair of adjacent vertices x and y in a graph G, the vertices f(x) and f(y) are adjacent in graph H. Similarly, the function preserves non-adjacency if f(x) and f(y) are non-adjacent whenever x and y are non-adjacent.

Another important property of isomorphism is to determine whether there exists a one-to-one mapping of the vertices of one graph G onto the vertices of another graph H such that adjacency is preserved.

Definition 2,*Two vertices x and y in an undirected graph G are said to be adjacent in G if x and y are endpoints of an edge of G. If e is an edge associated with {x, y}, the e is called incident with the vertices x and y. The edge e is also said to connect x and y. The vertices x and y are called endpoints of an edge associated with {x, y}. To check the no. of edges incident to a vertex, we follow the Definition 3.*

Definition 3.*The degree of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex x is denoted by deg(x).*

The in-degree and out-degree of the graphs is also an important factor to take care of which shows a one-to-one correspondence between the nodes. The in-degree of node x, denoted by deg- (x), is the number of edges with x as their terminal vertex node and the out-degree of x, denoted by deg+(x), is the number of edges with x as their initial vertex node.

Definition 4.*Given two graphs G(V_1, E_1) and H(V_2, E_2) are said to be isomorphic if there exist a one-to-one correspondence (bijection) f : $V_1 \rightarrow V_2$ and g : $E_1 \rightarrow E_2$ which maps each edge (v, e) to f(v), f(e).*

3. UML Sequence Diagram Semantics

We are using the notion of directed graphs in particular for representing the behavioral specifications of the complex software system modeled using UML2.0 Sequence diagrams that have been subjected to a model transformation. Sequence diagrams mainly focuses on showing the interactions between the various objects involved in a particular task or process as messages over a time period. Hence, time flow is the high priority constraint involved in the sequence diagrams which determines the flow of control of activities taking place in the system in an ordered sequence. It helps us in verifying whether the model is satisfying the behavior preservation property or not. Each of the statements, methods or activities involved in the participating objects is represented as vertices and the inter communication between vertices in an interaction, of edges. The edges quantify the ordering precedence in which the execution takes place in the specified scenario.

UML2.0 Sequence diagrams basically focus on visually specifying the interactions between various objects participating in the system in a time sequenced manner. An interaction typifies a message or a sequence of messages in a fragment [1]. An interaction set in a sequence diagram, SD can be generally represented as $I=\{m_1, m_2, \ldots, m_n; n\geq 0\}$, where $m_1, m_2, \ldots, m_n$ are the set of messages in the sequence diagram. If $n = \phi$, represents a null interaction ($\wedge$), which implies no occurrence of a message in the interaction I. if n=1, represents the occurrence of a single message and, n>1 shows the occurrence of a fragment with n messages.

For any two interactions in the sequence diagram, there exists a precedence relation showing the order of the message flows, representing the timing order in the design SD model. The precedence relation between two interactions I_1 and I_2 represented as $I_1 < I_2$ implies that I_1 should immediately occur before the occurrence of I_2, which shows the time order precedence of I_1 over I_2, where I_1 and I_2 are the interactions taking place within the same fragment. The precedence relations play a significant role during the verification of the SD model. For the two operands within the same fragments, the precedence relation should hold the following properties [10]:

(a) Asymmetric: if $I_1 < I_2$, then, I_2 not $< I_1$

(b) Non-transitive: if $I_1 < I_2$, then $I_2 < I_3$, then, I_1 not $< I_3$

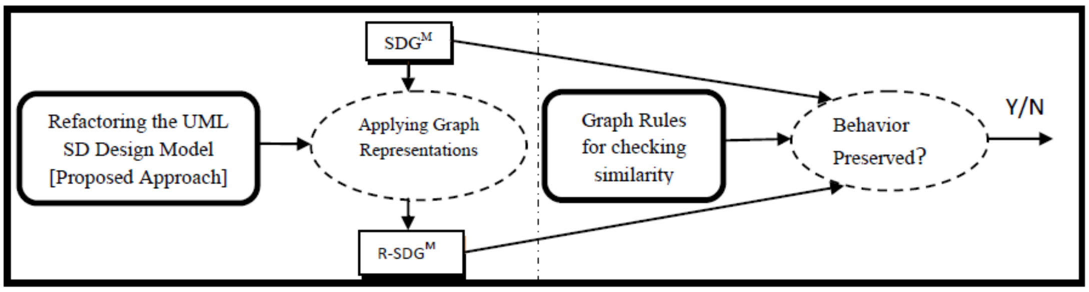
(c) Non-reflexive: I_1 not $< I_1$

4. Representing Behavioral Preservation using Graph Isomorphism

The graph model helps in depicting the representation of the UML models in a manner which can be easily verified to prove the syntactic as well as the behavioral correctness of the design model as well as the source code generated from it. The work uses the flow graph model, a directed graph, to represent the sequence diagram model of the system.

A Flow Graph (FG) is a directed graph S_{fg} = <M, E>; where N is a set of messages captured during a system interaction fragment and E represents the flow control ordering of the messages. All the messages involved in a particular interaction are represented using vertices. Each directed edge in the flow graph is associated with an ordered pair of vertices. For instance, the directed edge associated with the ordered pair M(x, y) is said to start at x and end at y.

Fig.1 depicts the overall behavioral preservation approach proposed in this work using graph isomorphism, a widely used graph matching technique. The flow graph model thus constructed is used to verify whether the model obtained after the proposed model refactoring approach proposed in the work satisfies the behavioral preservation property [4]. The flow graphs constructed are to be formally verified using the graph isomorphism properties [9]. The proof shows that the model obtained after the refactoring approach maintains the behavior preservance property.



SDG^M
Refactoring the UML SD Design Model [Proposed Approach]
Applying Graph Representations
Graph Rules for checking similarity
Behavior Preserved?
Y/N
$R\text{-}SDG^M$

Fig.1 Behavioral Preservation Architecture

Fig.2 depicts the UML2.0 Sequence diagram visualization of the simulation process of a coal pulverizing mill optimization process. From the above diagram, we can comprehend that CoalStorage and Pulveriser are the main participating objects in this interaction scenario. The metadata information related to the sequence control flow of this scenario are described as $SD_{pulverizing}$ ={loop, m1, alt}. Since the sequence diagram focuses on the message ordering with respect to time sequences, there must be a precedence relation between them. Hence we strictly adhere to those precedence standards during the execution of metadata elements in the sequence diagrams.

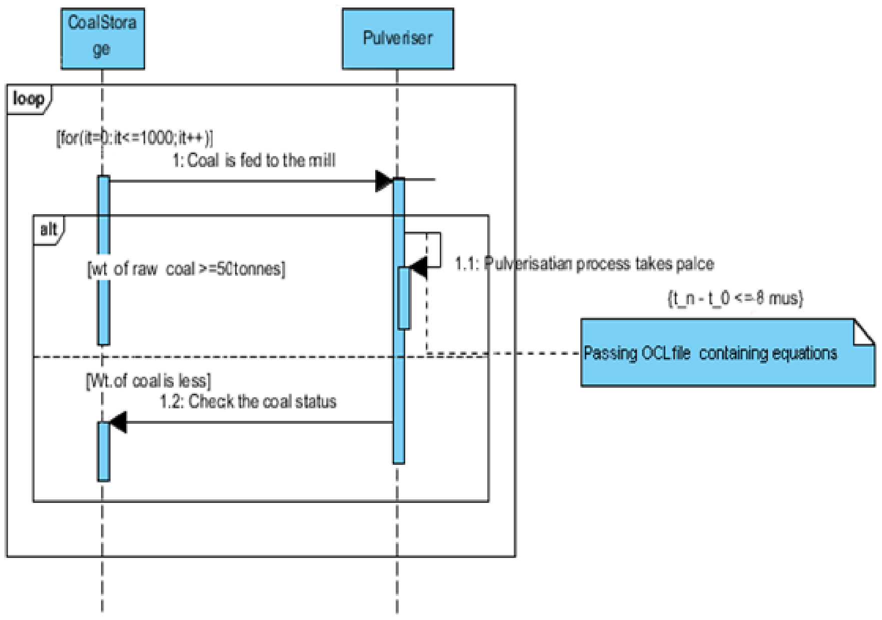

CoalStora ge
Pulveriser
loop
[for(it=0;it<=1000;it++)]
1: Coal is fed to the mill
alt
[wt of raw coal >=50tonnes]
1.1: Pulverisatian process takes palce
{t_n - t_0 <=8 mus}
Passing OCL file containing equations
[Wt.of coal is less]
1.2: Check the coal status

Fig.2 UML 2.0 sequence diagram model for the pulverisation process

For the flow graph obtained to be consistent with the sequence diagram design, one to one correspondence must hold between the type of the control flow in the sequence diagram with that of the edge in the flow graph which represents the precedence order of the fragments in the interaction. The precedence relation among the interactions in the above SD model is observed as P(SD) = {loop < m_1, m_1<alt} U P(SD_{alt}); where P(SD_{alt}) constitutes the set of precedence relations among the interactions that have occurred within the alt fragment Salt, i.e., SD_{alt} = {m_2,selfloop, m_3}. To evaluate P(SD_{alt}), we have to consider a null interaction Λ, which is coming before and after the messages m_2 and m_3 as they are appearing as a single message interaction in the alt fragment, thus forming P(SD_{alt}) = {Λ< m_2, m_2< Λ, Λ< m_3, m_3< Λ}. Thus P(SD) = { Λ< loop, loop < m_1, Λ<loop, m_1<alt, Λ< m_2, m_2< Λ, Λ< m_3, m_3< Λ}.

The precedence relation between two interactions I_1 and I_2 represented as $I_1 < I_2$ implies that I_1 should immediately occur before the occurrence of I_2, which shows the time order precedence of I_1 over I_2, where I_1 and I_2 are the interactions taking place within the same fragment.

Fig3. shows the flow graph model corresponding to the UML sequence diagram model for the coal pulverization process optimization. The messages illustrate the communication between the objects, that are represented as vertices and their execution order typifies the edges in the flow graph representation.

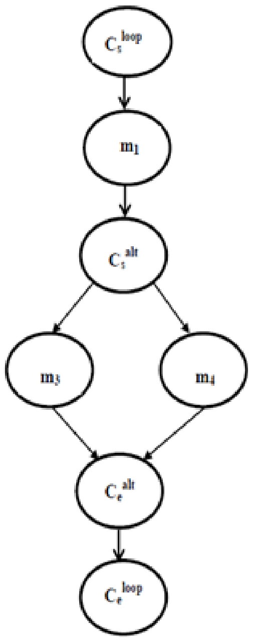


Fig. 3. Flow Graph Model for the UML SD model for Coal Pulverisation Process.

In this work, we consider graph isomorphism as an underlying formalism for the specification and analysis of system behavior and for the verification process of the model refactoring applied on to the design model. Refactoring is applied in the model in order to associate the constraint validations into the model so that we can perform the model verification and validations from the design stage itself. Refactoring ensures the behavior transformation during the design in which its behavior preserving properties are defined based on UML constraints at the metamodel level. In order to depict the dynamic behavioral properties out of UML models, sequence diagram is well suited as it can express more precisely the object interactions as scenarios.

Graphs can be typically represented as matrices which help in simplifying the computation process involved in it. One way of representing graphs models as matrices is based on the adjacency of vertices where elements are usually represented based on the ordering chosen for the vertices.

The adjacency matrix so obtained for the flow graph of the sequence diagram model designed for the coal pulverizing optimization process is as follows:

$$Adj(\mathrm{SD}_{br}) := \begin{vmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Fig.4 shows the flow graph of the sequence diagram model which had undergone the model refactoring process. Here we can clearly understand that additional modifications that have taken place in the model when compared with the flow diagram in Fig3. The block **a** and block **b** corresponds to the structural modifications made to the message fragments/methods with the association of constraints during the refactoring process. The adjacency matrix obtained for the refactored UML2.0 sequence diagram model is:

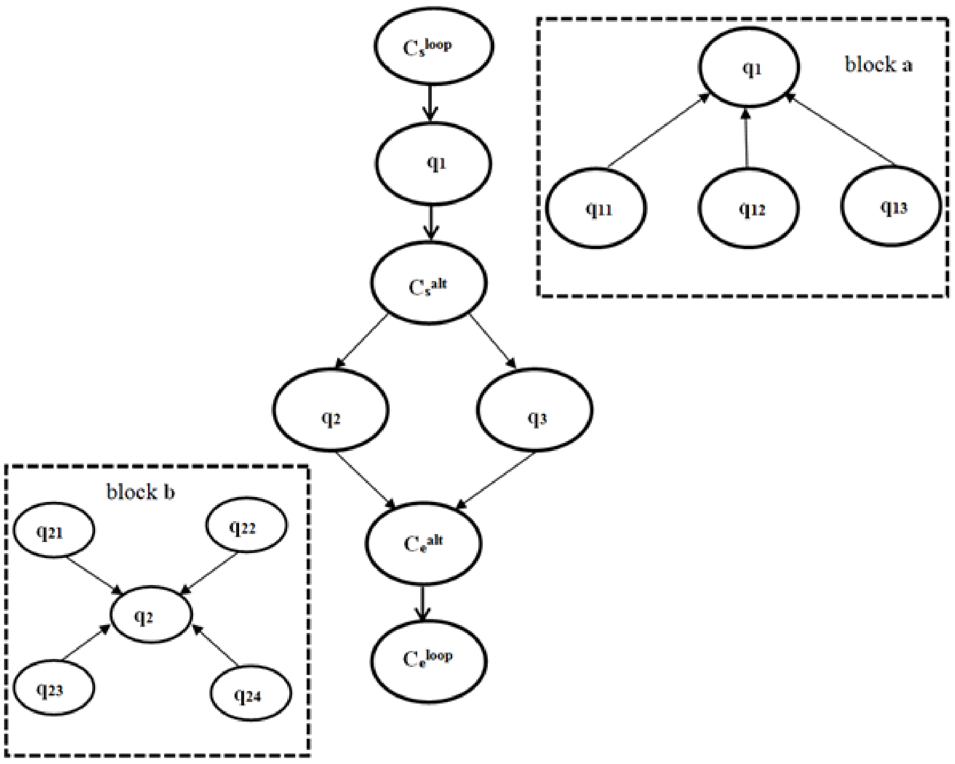


Fig. 4. Flow Graph Model for the refactored UML SD model

$$Adj(\mathrm{SD}_{ar}) := \begin{vmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Since the blocks shown in the flow graph correspond to the extension of the nodes q1 and q2 that will not make any alterations to the overall behavioral specification of the system, hence we are treating it as a single activity. Here, the two flow graphs obtained have exactly the same form, as there is a one to one correspondence between their vertex sets that preserves edges. Hence we can undoubtedly prove that these two graphs are isomorphic.

If G and H are isomorphic graphs, then both have the same degree of sequence. The in-degrees of each of the vertices in both the graphs conform to the graph isomorphic property. The degree of vertices in graph G represented as g(i) are deg(g0) = 0; deg (g1) = 1; deg (g2) = 1; deg (g3) = 1; deg (g4) = 1; deg (g5) = 2; deg (g6) = 1 respectively. Similarly the degree of the vertices in graph H represented as h(i) are deg (h0) = 0; deg (h1) = 1; deg (h2) = 1; deg (h3) = 1; deg (h4) = 1; deg (h5) = 2; deg (h6) = 1.

As per the definition of the isomorphism of graphs, there must be a one-to-one correspondence function *f* between these vertices showing the adjacency. Hence we found that the function f(g0) = h0;f(g1) = h1; f(g2) = h2; f(g3) = h3; f(g4) = h4; f(g5) = h5; and f(g6) = h6.

In order to prove the consistency between the graphs we have proved that the two graphs generated satisfy graph isomorphism. Based on the above definitions, we are finding out the in-degree and out-degree of the vertices of the given graph S_{fg}. In order to show that the two models satisfy the behavioral preservance properties, we have shown that inflow (in-degree) and outflow (out-degree) of the nodes in the graph in both the source and target sequence diagram models are the same.

Conclusion

Behavior Preservation is one of the essential properties that have to be verified for proving the correctness of the UML behavioral models, as they visualize the dynamic aspects of the system. The work uses a graph matching technique, graph isomorphism to quantify the behavioral aspects of the sequence diagram model along with the refactored model generated from it. The graph isomorphism property is utilized to check the consistency of the model and its transformation. The behavioral preservance property of the model has been verified from the adjacency matrices obtained from the flow graph of the model as well as the transformation model.

References

[1] Booch, Grady. The Unified Modeling Language User Guide. Pearson Education India, 2005.
[2] B´ezivin, J.(2005). Model driven engineering: Principles, scope, deployment and applicability. In: Proc. Summer School on GTTSE 2005, Springer.
[3] Boehm, B W., et al. (1976). Quantitative evaluation of software quality. Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press.
[4] Chitra, M. T., and Elizabeth Sherly. (2014) Refactoring sequence diagrams for code generation in UML models. Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on. IEEE, 2014.
[5] Fowler, M. (1999) Refactoring: Improving the Design of Existing Code. Addison-Wesley.
[6] Opdyke, W.F.: Refactoring (1992) A Program Restructuring Aid in Designing Object-Oriented Application Frameworks. PhD thesis, University of Illinois at Urbana Champaign.
[7] Kempen, M., et al(2005). Towards proving preservation of behaviour of refactoring of UML models. In: Proc. SAICSIT.
[8] Kleppe, A., et al. (2003). MDA Explained; The Model Driven Architecture: Practice and Promise, Addison Wesley.
[9] Rosen K.H (2007). Discrete Mathematics and its Applications with Combinatorics and Graph Theory, 6th Edition, The Mc Graw Hill Companies.
[10] Kundu, Debasish, Debasis Samanta and Rajib Mall... Automatic code generation from unified modelling language sequence diagrams. IET Software, 7(1), 2013.