

OBTAINING INITIAL INFORMATION FOR BEHAVIORAL SOFTWARE MODELS' PROCESSING

Olena Chebanyuk, Oleksii Dyshevyy, Valentyna Skalova

Abstract: *In software development process, following AGILE approach, operations with software models take a great role. The paper is devoted to important aspect of software modeling process, namely to analysis of XMI representation of UML diagram. Preparing quality information for different software models processing operations is an important step for effective software development processes performing.*

Paper is devoted to improvement of an approach of text to model transformation. The essence of the proposed approach is to decompose data stream into chains and mark conditional and cycle operations. Pointing of such operations is a necessary condition for preparing quality initial information for software models refinement and comparison.

A notation for an analytical representation of behavioral operations, namely conditional and cycle ones is proposed in this paper.

The example of representing of different types of behavioral software models according to the proposed notation is shown.

Keywords XMI, UML, behavioral software model, graph representation,

ITHEA classification keywords: D2 software engineering, D 2.0 Tools.

Introduction

According to definition from UML standard, software models are UML diagrams [OMG, 2015]. Analytical representation of behavioral software models is the ground for the successful performing of different Model-Driven Architecture operations. The set of these operations are: software models' comparison, reusing, refactoring, and merging. These operations are performed in different processes in software development lifecycle activities.

Obtaining detailed information about data streams, represented in behavioral software models, allows providing a comparison considering semantic aspects and refinement, based on operations with algorithms.

Well-designed notation for behavioral software models representation provides a background for comparing, transformation or refinement of software diagrams considering semantic aspects.

This paper is a continuation of papers [Chebanyuk] and [Chebanyuk, 2018]. In the paper [Chebanyuk, 2018] an approach to restore software model structure from UML diagram stored in modeling environment is represented. The concept of different types of behavioral software models representation is proposed in the paper [Chebanyuk, 2015].

Used terminology

Let's introduce main concepts of used terminology for describe the proposed approach

Table 1 Analytical denotations for restoring software model structure

Concept	Explanation and analytical representation of concept
Software model (SM)	According to the UML 2.5 standard SM is an UML diagram. Denote it as SM and SM of some type as SM_{type} where type=use case, type=class, etc.
Software model representation	<p>The graph representation is chosen.</p> $SM_{type} = (O_{type}, L_{type}) \quad (1)$ <p>where</p> <p>O_{type} – a set of SM objects that are used in SM_{type} notation. Objects are the elements of SM notations that can be expressed as graph vertexes.</p> <p>L_{type} – a set of software model links that are used in SM_{type} notation. Links are elements of SM notation that can be expressed as graph edges.</p> <p>Common definitions for behavioral SM representation are presented in the paper [Chebanyuk, 2015].</p>
Elementary sub-graph	<p>It is a part of a graph, consisting of two linked vertexes.</p> <p>Denote an elementary sub-graph as:</p> $e = (o_1, l, o_2) \quad (2)$ <p>where $o_1, o_2 \in O$ are software model objects linked by link $l \in L$.</p>

Concept	Explanation and analytical representation of concept
Set of elementary sub-graphs	All elementary sub-graphs of SM. Denote this set as A .
Linked Elementary Sub-Graphs (LESG)	<p>Consider two elementary sub-graphs $e_1 = (o_1, l_1, o_2)$ and $e_2 = (o_2, l_2, o_3)$</p> <p>If two elementary sub-graphs are interconnected through an object $o_2 \in O$ these two sub-graphs are considered linked. Consider a pair of elementary sub-graphs e_1 and e_2</p> <p>Determine e_1 as the first linked elementary sub-graph, e_2 respectively as the seconds.</p>
Starting border elementary sub-graph	<p>Elementary sub-graph that has no first linked elementary sub-graph. Consider $e_1 = (o_1, l_1, o_2)$. Usually $o_1 \in O$ is an object from which streams of UML diagram are started. These objects are actors or objects that have no incoming links.</p>
A set of starting border elementary sub-graphs	<p>A set that contains all starting border elementary sub-graphs of software model.</p> <p>Denote this set as $START$.</p> $START = \{e_{start,1}, e_{start,2}, \dots, e_{start,k}\}, k = START \quad (3)$
Switching elementary sub-graphs	<p>Consider two linked elementary sub-graphs. $e_1 = (o_1, l_1, o_2)$ and $e_2 = (o_1, l_2, o_3)$. They are started from the $o_1 \in O$. An elementary graph located on SM before e_1 and e_2, $e_0 = (o_0, l_0, o_1)$ is determined as a switching elementary sub-graph.</p>
A set of switching border elementary sub-graphs	<p>A set that contains all switching elementary sub-graphs of a SM. Denote this set as $SWITCH$.</p> $SWITCH = \{e_{switch,1}, e_{switch,2}, \dots, e_{switch,p}\}, p = SWITCH \quad (4)$
Finishing border elementary sub-graphs	<p>An elementary sub-graph that has no second linked elementary sub-graphs. Consider $e_1 = (o_1, l_1, o_2)$. Usually $o_2 \in O$ is an object to which streams of UML diagram are ended. Other words these objects have no outgoing links.</p>
A set of finishing border elementary sub-graphs	<p>A set that contains all finishing border elementary sub-graphs of a SM.</p> <p>Denote this set as $FINISH$.</p> $FINISH = \{e_{finish,1}, e_{finish,2}, \dots, e_{finish,t}\}, t = FINISH \quad (5)$
A set MIDDLE	<p>All elementary sub-graphs that are not included to sets $START$, $SWITCH$, and $FINISH$ are included to the set $MIDDLE$.</p>

Concept	Explanation and analytical representation of concept
Software model sub-path	<p>A part of software model, consisting from chain of linked elementary sub-graphs. Denote a sub-path of a SM as <i>chain</i>. Using (2) <i>chain</i> is denoted by the following:</p> $\begin{aligned} chain &= ((o_1, l_1, o_2), (o_2, l_2, o_3), \dots, (o_{n-1}, l_{n-1}, o_n)), n = chain \\ chain &= (e_1, e_2, \dots, e_n) \end{aligned} \quad (6)$ <p>where n is a number of elementary sub-graphs in sub-path.</p> <p>There are several variants of forming chains:</p> <ul style="list-style-type: none"> — starting from a border elementary sub-graph and ending on a switching elementary sub-graph; — starting from a next elementary sub-graph to switching one (the second elementary sub-graph in pair of linked elementary sub-graph) and ending on other switching elementary sub-graph; — starting from a next elementary sub-graph to switching one (the second elementary sub-graph in pair of linked elementary sub-graph) and ending on the finishing border elementary sub-graph; — starting from a starting border elementary sub-graph and ending on finishing border elementary sub-graph.

The contribution of this paper: is a modification of the text to model transformation approach for restoring behavioral software model structure decomposing models into chains considering main operations of algorithms represented on them.

Such decomposition simplifies the further data stream analysis as well as comparison and refinement operations. In order to compare software models it is possible to analyze combination of processes represented in reference software models and ones used in development of concrete projects. From the other hand obtaining information about operations represented in behavioral software models provides a background for their processing considering semantic aspects.

Investigation of the existing approaches drawbacks

In order to ground the necessity of the existing approaches improvement, it is necessary to illustrate the drawbacks of the previous methods. Doing this, consider two examples for different type of software models, namely UML Use Case and Communication Diagrams.

Let's remember that the process forming of elementary sub-graphs chains uses graph representation of software models. Such a representation considers software model as a graph that consists from a set of objects and links.

In the beginning graph objects are divided into several types [Chebanyuk, 2018].

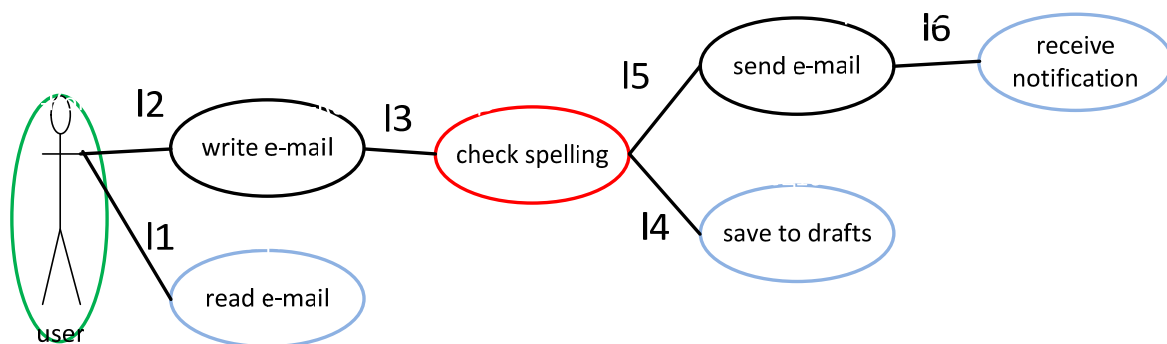


Figure 1 Different types of UML diagram objects.

Let's remember the classification of UML diagram objects. It is proposed to divide software model elements into groups:

- Starting border element. Such element has no incoming links. In the figure 1 it is "user" (marked by green).
- Switching element. Such element has several outgoing links and at least one incoming link. In other words several linked objects can start from it. In the figure 1 it is "check spelling". (This element is marked by red).
- Finishing border element. Such element has no outgoing links. In the figure 1 they are "receive notification", "save to draft", and "read e-mail". (Elements are marked by blue).

Such representation of elements help to compose rules to form chains of software model elements that are linked directly [Chebanyuk, 2018].

An example of forming the set A for small Use-Case Diagram (Figure 2) is represented in the Table 2.

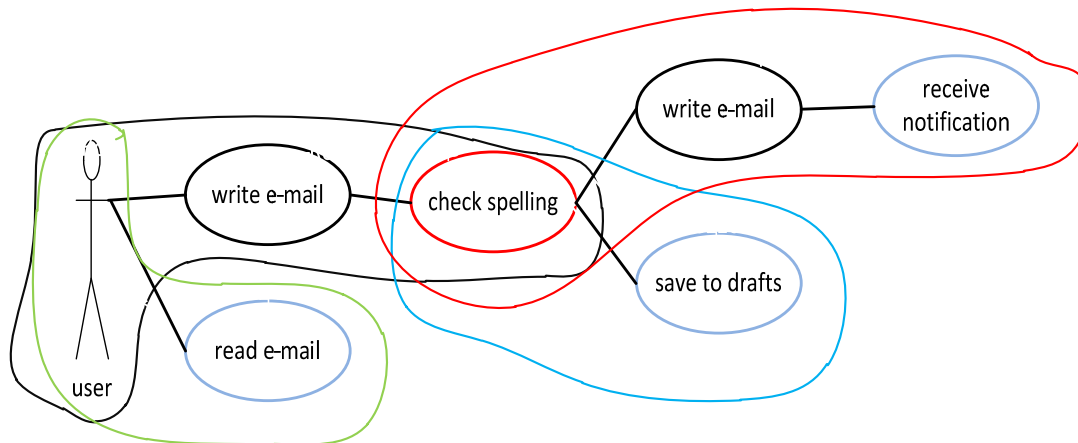


Figure 2. Illustration decomposition of software model to linked chains

Table 2. Description of forming CHAIN process from UML Use-Case diagram, shown in the figure 2

Use Case diagram sets	Explanation
$START = \{(el_1, l_1, el_2), (el_1, l_2, el_3)\}$ $SWITCH = \{(el_3, l_3, el_4)\}$ $FINISH = \{(el_4, l_4, el_7), (el_5, l_6, el_6)\}$ $MIDDLE = \{(el_4, l_5, el_5)\}$	$chain_1 = (el_1, l_1, el_2)$ There is no elementary sub-graph starting from el_2 . That's why the forming of the $chain_1$ is finished on the first step. Elementary sub-graph (el_1, l_1, el_2) is deleted from the set START. In the figure 2 this chain is marked by green.
$START = \{(el_1, l_1, el_2), (el_1, l_2, \boxed{el_3})\}$ $SWITCH = \{(\boxed{el_3}, l_3, el_4)\}$ $FINISH = \{(el_4, l_4, el_7), (el_5, l_6, el_6)\}$ $MIDDLE = \{(el_4, l_5, el_5)\}$ In the figure 2 this chain is marked by black	el_3 is a common element for (el_1, l_1, el_3) and (el_3, l_3, el_4) . According to the proposed approach if $ref \in SWITCH$ Forming of $chain_2$ is finished. $chain_2 = ((el_1, l_2, el_3), (el_3, l_3, el_4))$ Elementary sub-graphs (el_1, l_1, el_3) and (el_3, l_3, el_4) are deleted from the sets START and SWITCH.

Use Case diagram sets	Explanation
<p>$START = \emptyset, SWITCH = \emptyset$ $FINISH = \{(el_4, l_4, el_7), (el_5, l_6, el_6)\}$ $MIDDLE = \{(el_4, l_5, el_5)\}$</p> <p>In the figure 2 this chain is marked by red</p>	<p>According to the proposed approach forming of $chain_2$ in this case is started from the set MIDDLE because $START = \emptyset$ and $SWITCH = \emptyset$</p> <p>el_5 is a common element for (el_4, l_5, el_5) and (el_5, l_6, el_6).</p> <p>$chain_3 = ((el_4, l_5, el_5), (el_5, l_6, el_6))$</p> <p>Elementary sub-graphs (el_4, l_5, el_5) and (el_5, l_6, el_6) are deleted from the sets MIDDLE and FINISH.</p>
<p>$START = \emptyset, SWITCH = \emptyset$ $FINISH = \{(el_4, l_4, el_7)\}, MIDDLE = \emptyset$</p> <p>In the figure 2 this chain is marked by blue</p>	<p>$chain_4 = ((el_4, l_4, el_7))$</p>

Similar steps are used to compose the chains from UML communication diagram that is represented in the Figure 3.

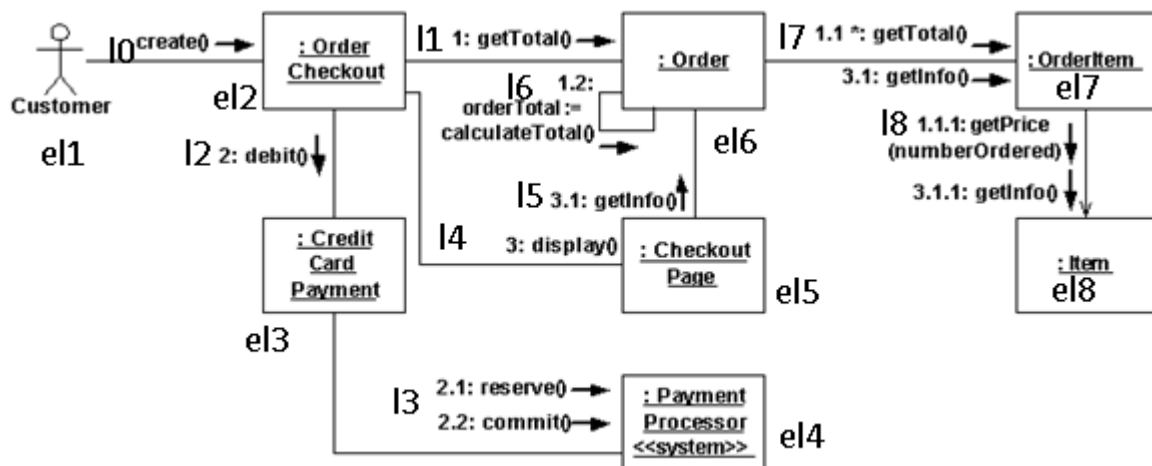


Figure 3 Communication diagram of performing order in internet shop

The figure is taken from AgileModeling site <http://agilemodeling.com/style/communicationDiagram.htm>

An analytical representation according to proposed approach is prepared

1. Form a graph representation of communication diagram

$$\begin{aligned}
 el_1 &= customer, el_2 = order : checkout, el_3 = order, el_4 = checkoutpage, el_5 = creditcardpayment, \\
 el_6 &= orderitem, el_7 = item, el_8 = payment \\
 l_0 &= create : l_1 = 1 : gettotal, l_2 = 2 : debit, l_3 = 2.1reserve(2.2commit), l_4 = display, l_5 = 3.1get inf o, \\
 l_6 &= 1.1ordertotal(calculateTotal), l_7 = 1.1GetTotal(3.1GetInfo), l_8 = 1.1.1getprice(3.1.1get inf o)
 \end{aligned} \tag{7}$$

2. Compose different sets of UML communication diagram sub-graphs.

$$\begin{aligned}
 START &= \{(el_1, l_0, el_2)\} SWITCH = \{(el_2, l_1, el_6)\} \\
 FINISH &= \{(el_3, l_3, el_4), (el_7, l_8, el_8)\} \\
 MIDDLE &= \{(el_2, l_2, el_3), (el_2, l_4, el_5), (el_5, l_5, el_6), (el_6, l_6, el_6), (el_6, l_7, el_7)\}
 \end{aligned} \tag{8}$$

3. Let's form the chains of elementary sub-graphs following the algorithms, described above.

$$\begin{aligned}
 chain_1 &= ((el_1, l_0, el_2)) \\
 chain_2 &= ((el_2, l_1, el_6), (el_6, l_6, el_6), (el_6, l_7, el_7), (el_7, l_8, el_8)) \\
 chain_3 &= ((el_2, l_2, el_3), (el_3, l_3, el_4)) \\
 chain_4 &= ((el_2, l_4, el_5), (el_5, l_5, el_6))
 \end{aligned} \tag{9}$$

Analyzing communication diagram, chains it is pointed that element el_2 is a switch object, but does not appeared in the SWITCH objects set. It appears in the START set. As START set is formed before SWITCH element el_2 was not recognized as SWITCH element.

Research question 1: Why it is important to consider all SWITCH objects?

SWITCH objects have important meaning in defining the type of operations. For example: in considering the conditional operation. It becomes a motivation for authors to improve the algorithm of decomposing software model elements into the chains. The essence of the proposed algorithm is to provide a preparation of initial information for the further analysis after decomposing behavioral software models into chains.

Task: to propose an approach for decomposition of UML diagram chains into operations.

Challenges to prepare initial information for semantic comparison and merging of UML diagrams in convenient form for the further analysis

In order to perform this task the next Research Problems (RPs) should be solved:

- RP1: To define key features of conditional and cycle operations in UML diagrams.
- RP2: To provide changes to algorithm restoring software model structure to define conditional and cycle operations.
- RP3: To propose an analytical form of conditional and cycle operation recording.

Application of the proposed approach is to provide a background for software model comparison, refinement, and transformation considering semantic aspect.

Grounding of the necessity of designing algorithm for decomposing elementary sub-graphs into the chains marking operations

The notation of behavioral software models analytical representation is given in the paper [Chebanyuk, 2015]. In this paper the concept of all operations performed in behavioral software models processing is introduced. We consider two types of operations, namely conditional and cycle ones. These types of operations are defined directly only in sequence diagrams from all behavioral software models. But other types of UML diagrams have more common notations and no notation tool on UML diagrams, except comments, to define operations performed in algorithm in precise form (figure 4).

Description of the proposed approach

Solutions of the research problems given above:

RP1: define key properties to cycle and conditional operations.

In order to define key features of behavioral software models consider communication diagrams as diagrams that are designed to show data streams.

Table 2 illustrates examples of communication diagrams fragments that match to different operations with their elements.

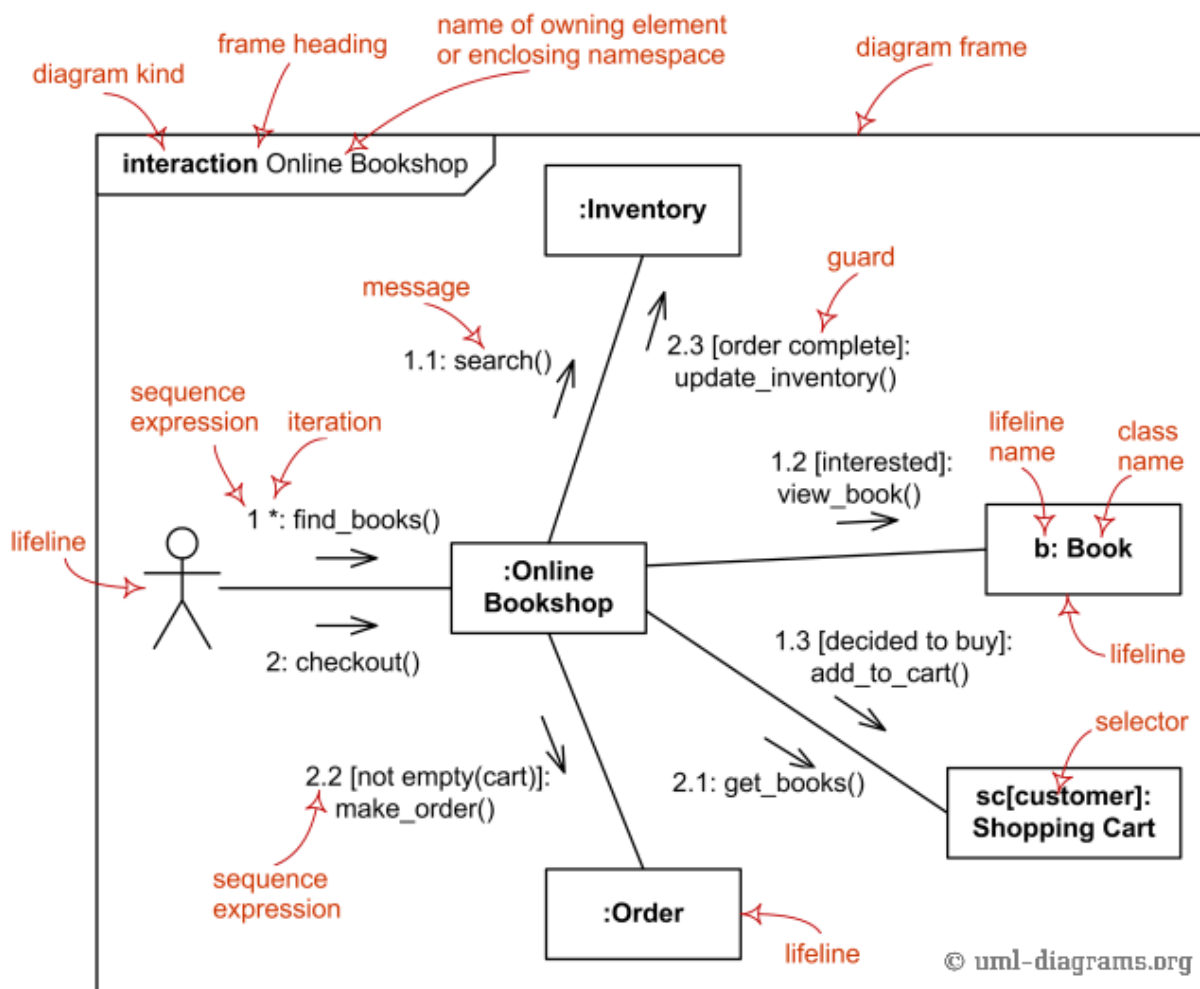
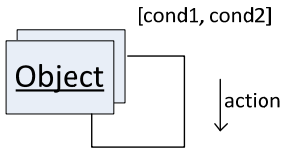
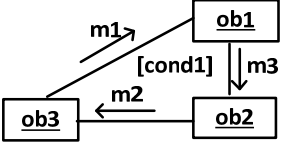
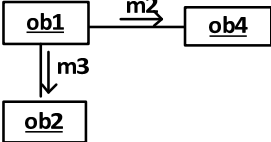


Figure 4 An example of operations description in behavioral software models

<https://www.uml-diagrams.org/communication-diagrams.html>

Table 2 Examples of behavioral software models fragments

Communication diagram fragment	Analytical representation
 <p>Explanation: simple action that is executed under all elements of some collection</p>	<p>The first example of cycle construction</p> $chain = (Object^*, action, Object^*)$ <p>Where $Object^*$ - is a collection of objects</p>

Communication diagram fragment	Analytical representation
 <p>Explanation while cond1 is true actions m_3 and m_2 are executed under objects ob_2 and ob_3 respectively.</p>	<p>The second example of cycle construction</p> $chain = ((ob_1, m_3, ob_2), (ob_2, m_2, ob_3), (ob_3, m_1, ob_1))$ <p>Also such communication diagram fragment can match to conditional statement</p> <p>Pay attention to collision the same communication diagram fragment may point both to conditional and cycle statement.</p>
 <p>Explanation when cond1 is true conditional operation is performed with ob_2 otherwise this operation is executed with ob_4</p>	<p>Another example of conditional construction</p> $chain_1 = \{ob_1, m_2, ob_4\}$ $chain_2 = \{ob_1, m_3, ob_2\}$ <p>Where: ob_1 is a switching object.</p>

Firstly we can assume that switching object points to conditional operation. In order to detect cycle operation let's pay attention to several variants of cycle operation organization.

1. Consider a loop, recorded in one chain. The cycle condition is formulated as follows: chain must starts and ends with the same object (see the first example in the table 2).
2. Consider a loop, represented in two chains, namely $chain_1$ and $chain_2$. The cycle condition is formulated as follows: $chain_1$ starts from the object ob_1 . This object should be the last one in the $chain_2$. The second condition is the next: $chain_1$ ends with the object ob_1 $chain_2$ must starts from the same object.

$$chain_1 = ((ob_1, m_2, ob_4), (ob_4, m_3, ob_6))$$

$$chain_2 = ((ob_1, m_3, ob_2), (ob_1, m_3, ob_2))$$

3. We can extend this condition for representing loop recorded in several chains, $CHAIN = (chain_1, chain_2, \dots, chain_n)$. ob_1 is the first object in $chain_2$ and the last one in the

$chain_2$. In general ob_i is the first object in the $chain_{i+1}$ and the last one in the. The object ob_0 is the last in the $chain_n$ and the first one in the $chain_1$.

It is necessary to point that even when it is possible to combine a cycle from chains it is not always cycle operation. Consider the situation shown in the figure 3. Combination of arrows directions illustrates that only one cycle is represented in the figure 3. (It is started and finished in object el_6) Also we can notice that the object el_2 is a switch one but according to algorithm proposed in paper [Chebanyuk, 2018] elementary sub-graph (el_1, l_0, el_2) participates in forming START set. Repeating actions that are based on proposed algorithm we can skip switch objects from the START set. But definition of them is a very important thing because switching objects point to conditional and probably cycle operations.

RP2: Provide changes to algorithm restoring software model structure to define conditional and cycle operations

It is proposed to take three changes into algorithm of decomposing UML diagram into the chains.

First: additional point is added after elementary sub-graphs are divided into sets START, FINISH, MIDDLE, and SWITCH.

All switching elementary sub-graphs are marked as conditional ones. Conditional elementary sub-graphs are the first sub-graphs in the chain representing conditional operations. The form to consider conditional operations will be introduced later.

Second: in order to avoid situations when elementary sub-graph is not considered as a conditional one (9). It is proposed to perform review of chains after their forming. This review should calculate the number of chains that are started from some object obj . If there is more than one chain starting from the "obj" then "obj" is considered as switching object.(in considering communication diagram in the figure 3 we can define one more switch object. It is object el_2).

"Switch record" is formed.

Third: the analysis of chains is performed in order to define linked chains. Linked chains are those that have common element. This element should be the last in the first chain and the first in the second one. In order to organize cycle the first chain should start and the last chain should finish from the same element.

RP3: Propose an analytical representation of recording conditional and cycle operations.

Analytical representation of "Switching record" to represent conditional operation

$$\Xi = \left[\begin{array}{l} el_1, l_1, el_2 \\ \left[\begin{array}{l} el_2, l_3, el_3 \\ el_2, l_4, el_5 \\ \cdot \\ \cdot \\ \cdot \\ el_2, l_n, el_k \end{array} \right] \end{array} \right. \quad (10)$$

Where: Ξ - denotation of the conditional operation

Analytical representation of cycle operation

$$O = ((el_1, l_1, el_2), \dots, (el_n, l_m, el_k), (el_k, l_1, el_p), \dots, (el_t, l_r, el_w), (el_w, l_s, el_1)) \quad (11)$$

where: O is a denotation of cycle operation.

Case study and evaluation of the proposed approach

Let's analyze the communication diagram represented in the figure 3. From the first view it seems to have two cycles. One of them is started from the object order:checkout another one from the object :order But the analysis of data stream shows that after object order:checkout two separate stream are started. That why this diagram is interesting to investigate and to prove formulated conditions. Modified algorithm of analysis of linked chains allows defining one mode switching object.

Review of related papers

Nowadays there are many investigations directed to analysis of information getting from the UML diagrams, designed in modeling environments. They are divided into two large directions, namely analysis of software development artifacts to get initial information for software model designing (requirement based engineering area). The second area of investigation is aimed to is to extract information from software model XML for the further analysis.

Let us consider papers taking information about software model from requirement specification.

Paper [Sneed, 2018] introduces tool to extract logical test cases or test conditions from requirement specification as they are referred to in the ISO Standard-29119. To archive this goal authors introduce the next terminology base: test condition, action, state, and rule.

A test condition defines a particular feature of the system. This can be an action, a state or a rule. An action would be "to place an order". A state is "the ordered article is available". A rule would be "the

customer can only place an order if the ordered article is available". A rule is as shown here actually a combination of a state with an action.

Information about logical test cases and centrally sequence of them, gathered into logical test cases, serves to expose some domain process in compact view. This information is initial for the analysis of problem domain algorithms and operations. It is possible for example to verify obtained logical test cases with reference ones and to define whether conditional operation was skipped or extra one is added.

Then consider papers proposing approaches that are based on extracting information from software models.

There are investigations related to processing the information about UML diagrams stored in modeling environment. Many researches are concentrated on using ready tools for software analysis. One of them is project SMartyParser. It is aimed to parse UML diagrams stored in XMI formats. Authors of this project [L. A. Lanceloti et al., 2013] use SD Metrics tool for the analysis of XMI. This tool allows performing the next operations:

- **Comprehensive design measurement**

SDMetrics ships with a rich set of object-oriented (OO) design measures covering structural properties of design elements from all UML1.x/2.x diagram types. Investigator may measure all the import design attributes - size, coupling, complexity and more - at all levels of detail, from the model, subsystem, package level down to classes and operations [SDMetrix, 2014].

- **Design rule checking**

Design rules and heuristics automatically check UML design for completeness, consistency, correctness, design style issues such as dependency cycles, and more [SDMetrix, 2014].

- **Early quality feedback**

SDMetrics finds architectural problems at the design stage, before they are committed to source code [SDMetrix, 2014].

- **Extensible set of design measures and design rules**

SDMetrics has a flexible and powerful mechanism to define and calculate new design rules and measures of your own, tailored to your development practices [SDMetrix, 2014].

- **Compare designs**

Calculate size metrics deltas to quantify the growth in size between two versions of a design, identify parts of the system design that have undergone much change, or evaluate alternative solutions to a design problem [SDMetrix, 2014].

The aim of investigation represented in paper [L. A. Lanceloti et al., 2013] is to define differences UML diagrams in software Product Line approach. In order to parse them class XMIReader is used. Then SDMetrics allows representing statistic values about software model. Analyzing these values it is possible to make a conclusion how much UML diagrams are differ.

There are authors [Chebanyuk and Mironov, 2017] taking efforts to design own tools for extracting information about software models elements. The proposed approach implies working with a machine-readable representation of a software model used by modeling environments. XMI is considered as a fitting option due to rich set of features allowing to effortlessly recognizing entities and relations between them. Moreover, XMI is used by notorious modeling environments like IBM Rational Software Architect (RSA) and Eclipse Papyrus and suggests various ways to process and analyze models.

Introduced idea and approach has been expressed as a simplistic framework that works with different types of UML diagrams. It is implemented as a core library and a set of frontends to it. .NET Core is used as a basis for realization because of a powerful XML-parsing features (XDocument, LINQ) and availability on multiple platforms.

The library itself is able to recognize diagrams in XMI format and provide additional data for Class and Use-Case diagrams. It is open to extension and may easily accept new functionality like custom business logic, new frontends, and new kinds of diagram to parse and new metrics to calculate for already implemented ones.

Proposed tool become a start point to design application for analyzing a class diagram in accordance to SOLID design principles [Chebanyuk and Povalyaev, 2017]. The first point of such an analysis is to extract class diagram elements by means of special LINQ requests. Then LINQ requests matching to expression of predicate logic proposed in paper [Chebanyuk and Markov, 2016] were composed.

Summarizing the review

The next conclusion is performed: there is a series of Investigations to analyze xmi representation of UML diagram but extracting information about UML diagram entities does not provide possibility to perform other operation with software model such as transformation, comparing, merging because it is necessary to involve additional information to include semantic aspects in some operation performing. Such semantic aspect is decomposing software model chains into operations.

Conclusion

The proposed approach for software models representation allows decomposing data streams in behavioral software models into chains. Modification of the proposed algorithm lets to consider

conditional and cycle operations that are used in algorithm. Such operations are "skeletons" for analyzing data streams and providing further software models comparison and refinement operations.

Bibliography

- [Chebanyuk, 2014] Chebanyuk, Elena.2014. Method of behavioural software models synchronization. International journal Informational models and analysis. – 2014, №2 P 147-163 <http://www.foibg.com/ijima/vol03/ijima03-02-p05.pdf>
- [Chebanyuk, 2015] Chebanyuk Elena An Approach to Behavioral Software Models Analytical Representation. International Journal "Information Models & Analyses", Volume 4, Number 1, 2015, p 51-79
- [Chebanyuk, 2018] Chebanyuk, Olena An Approach of Text to Model Transformation of Software Models. In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, pages 432-439 ISBN: 978-989-758-300-1
- [Chebanyuk and Mironov, 2017] Chebanyuk Olena and Mironov Yuriy, International Journal "Information content and Processing" Volume 4, Number 2, © 2017,
- [Chebanyuk and Markov, 2016] Chebanyuk E. and Markov K. (2016). An Approach to Class Diagrams Verification According to SOLID Design Principles. In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD*, ISBN 978-989-758-168-7, pages 435-441. DOI: 10.5220/0005830104350441 <http://www.scitepress.org/DigitalLibrary/PublicationsDetail.aspx?ID=HASwCJGMcXc=&t=1>
- [Chebanyuk and Povaliaiev, 2017] Chebanyuk Olena and Povaliaiev Dmytro International Journal "Information Technologies & Knowledge" Volume 11, Number 2, © 2017 p.114-143.
- [Lanceloti et al., 2013] L. A. Lanceloti, J. C. Maldonado, I. M. S. Gimenes, and E. Oliveira Jr. SMartyParser: a XMI Parser for UML-based Software Product Line Variability Models. In *Proc. Variability Modelling of Software-intensive Systems*, pages 10:1–10:5. ACM, 2013. https://www.slideshare.net/edson_ao_junior/edson-va-mos2013
- [SDMetric, 2014], The Software Design Metrics tool for the UML <https://www.sdmetrics.com/FeatOvw.html>
- Sneed, Harry M. "Requirement-Based Testing-Extracting Logical Test Cases from Requirement Documents." International Conference on Software Quality. Springer, Cham, 2018.
- [UML, 2012] Unified Modeling Language 2.5, 2012 Access mode <http://www.omg.org/spec/UML/2.5/Beta1/>
- [XMI, 2015] XML Metadata Interchange access mode <http://www.omg.org/spec/XMI/>

Authors' Information



Olena Chebanyuk – assoc. professor of Software Engineering Department, National Aviation University, Kyiv, Ukraine,

Major Fields of Scientific Research: Model-Driven Architecture, Model-Driven Development, Software architecture, Mobile development, Software development,

e-mail: chebanyuk.elena@ithea.org



Oleksii Dyshlevy – senior lecturer of Software Engineering Department, National Aviation University, Kyiv, Ukraine,

Major Fields of Scientific Research: Software Development, Web Applications, Web Services, Microservices, Software Design, Software Architecture, Software Metrics, Code Quality

e-mail: oleksiy.dyshlevyy@gmail.com



Valentyna Skalova – Software Engineering Department, National Aviation University, Kyiv, Ukraine,

Major Fields of Scientific Research: Software architecture, Software development,

e-mail: valentine.skalova@livenau.net