Formal Foundation, Approach, and Smart Tool for Software Models' Comparison

[1]Olena V. Chebanyuk, [2]Abdel-Badeeh M. Salem

[1]Software Engineering Department, National Aviation University,
Kyiv, Ukraine

[2]Computer Science, Faculty of Computer and Information Sciences,
Ain Shams University, Cairo, Egypt

chebanyuk.elena@ithea.org, abmsalem@yahoo.com

Abstract

Software models' comparison is an operation performed in all software development lifecycle processes. Comparison is one of the steps of software models refactoring, refinement, merging, quality estimation, etc. Such operations take place in different tasks in requirement analysis, software designing, testing, reengineering, etc.

This article proposes analytical foundations for software models' representation and corresponding technique for their comparison. To describe software model structure it is proposed to use graph representation. Software models' comparison technique based on sub-graphs matching is proposed. Following this technique important steps of software tool realization are described, namely, (i) grounding of the choice of development environment and tools for XMI files processing; (ii) the algorithm for extracting software model structure from XMI file; (iii) software models' comparison algorithm realization; (iv) description of software architecture. Peculiarity of proposed software tool is a possibility for visualizing elements of two software models that do not match each other directly in modeling environment. Data are extracted from XMI files by means of LINQ queries. After that they are stored in graph triples. To perform visualization, it is proposed to modify "*.layout" files, designed by Microsoft Visual Studio environment. Other important feature of the tool is a possibility to compare software models, designed in different modeling environments. It is done by means of involving new LINQ queries, considering specifics of storing software models in different modeling environments.

Paper contains case study, explaining the process of software models'analytical representation, and comparison technique implementation.

Keywords: *Agile, Graph Theory,FormalMethods of Software Engineering, LINQ,Software Model, SoftwareModel Formal Representation, XMI, XML, UML Diagram, Use Case Diagram.*

1. Introduction

Software models are central development artifacts in Model-Driven Development approach. Often they are represented as UML diagrams. Many operations in different software development life cycle processes include such tasks as software model merging, transformation, refactoring, and refinement [5],[25],[26]. Software model comparison is an important step for effective processing of all these operations.

Consider requirement analysis software development life cycle process[9]). It consists from such operations as requirement verification, validation, tracing, and refinement. Verification needs comparison of designed requirements with some etalon (for example UML diagrams illustrating problem domain processes) to prove that the requirement specification is represented correctly. Validation also contains some comparison to decide whether system requirements are designed in a proper way. We need to compare different requirements in a requirement list to avoid duplication or establish trace links, etc.[9]).

Consider such software development life cycle process as designing(ISO/IEC/IEEE 42010:2011(E)). It consists from such operations as designing of high level architectural solutions (often represented by means of package diagrams), middle level, and structural representation for concrete components. Comparison operation is performed when class diagrams are estimated in accordance to some etalon structures. In software architecture such structures are design patterns, architectural patterns, anti-patterns, and SOLID principles[15].

Also many other operations from other software development lifecycle processes are based on software models comparison.

2. Related Papers

Model comparison is a central operation in all software models processing operations. There are many scientific papers taking strong contribution in this process. Researches, which consider development of software model comparison approach, are developed in several directions. One of them is processing of UML diagrams text representation. Other papers are directed to development analytical approaches for software model comparison and representation. Consider both of them.

More detailed description of two software models comparison that are stored in XML files is proposed in paper [23]. Authors describe algorithm of two XML files comparison in details considering the fact that software model is stored in XML file as a tree.

Authors of paper [23] propose use DOM specification and consider XML file as no ordered DOM tree. Structures of XML files are compared by means of collaboration of some operations (mostly Insert () and Delete ()) for processing different tree parts. To speed up the comparison process hashing operations are used. If hash meanings are different, than more precise operations for comparison XML files fragments are used. Authors propose detailed analysis of described algorithms effectiveness. But results of two XML files comparison are visualized in plain text. It is not convenient for UML diagram analysis. User should (i) analyze text representation of XML files fragments, (ii) define UML diagram elements that match to this fragments, (iii) compare UML diagram in mind, and use this information for further UML diagrams processing.

Key questions for software model comparison are described in paper [24]. But authors propose just short recommendations related to model comparison process. Software models, represented as UML diagrams, are stored in XML files. Authors propose use composition of methods for XML files processing for analyzing of XML file internal structure.

But many questions still remains open after reading the paper [24]. For example what are steps of algorithms to compare software model.

Paper [11] discussed the requirements for model comparison, composition, and model transformation testing. A prerequisite of composition is the identification of common elements contained in the two sources so that the merged artifact does not contain duplicated information. A rule-based approach for performing automated comparison on diverse models is presented.

Authors [11] consider model comparison as an operation that divide elements into several types, namely: (1) Elements that match and conform, (2) Elements that match and do not conform, (3) Elements that do not match and are within the domain of comparison, and (4) Elements that do not match and are not within the domain of comparison. Matching refers to elements that represent the same idea or artifact, while conformance is additional matching criteria. An example of non conformance in an UML class diagram can be when a class in both models has the same name but one is abstract. So while they likely represent the same artifact, they do 'match enough' or conform to one another [11]. This paper centrally touches deep foundations in mechanism of comparison operations performing. Questions that require more precise recognition of links between model objects need modifying of comparison technique.

In the context of model versioning, in [3] model comparison is decomposed into three phases: Calculation, Representation, and Visualization. However, there is nothing about this decomposition that is specific to model versioning. In the following paragraphs authors elaborate on these phases and provide examples of approaches.

Paper [13] proposes model comparison as preliminary step for performing model versioning, merging, and cloning. Survey summaries software models comparison techniques considering papers [1], [2], [12], [19], [3] and others.

Some papers develop approaches to process textual representation of software models.

Authors [1],[2] use the same comparison stages as [11], but model comparison is based on UML's universally unique identifiers (UUID). During comparison unique elements in two models are determined and added to two separate lists.

Authors [3] propose a survey about software tools for model comparison. Survey describes list of papers that are devoted software model comparison plug-ins (mostly based on Eclipse platform) and software products (for example IBM Rational Software Architect (RSA) [12]). As in IBM RSA, software model comparison serves for model history managing user may compare parts of software models implementing approach proposed in [12].

Other considered techniques for software model comparison use software model graph representation. But comparison is also based on UUIDs processing or introducing some rules for matching software model parts [19]. Some rules designed for structural software models propose top down comparison, matching links defined in MOF standard, comparing structures, detected on meta-level. Software tools designed following this approach are EMFCompare [3], TopCased [8], SmoVer [18], UMLDiff [21] and many others.

To compare behavioral software models, they are represented as graphs or trees. Then different matching algorithms are used. There are different scan algorithms (eScan and aScan) [17]. Comparing fragments of graph structures leads to more precise results of matching software model fragments. Experiments concern sequence diagrams and state chart ones. There are attempts to develop comparison algorithms considering language semantic [14]. Authors (Soto and Munch, 2006) propose an algorithm Delta-P using Resource Description Framework (RDF) for software model representation and its further comparison.

Represented researches are interconnected with concrete modeling environment (plug-ins for concrete software tools) or propose comparison results in text view. Text representation of differences in software models contain some tips for estimation of software model similarities but can't speed up process of software model analysis. These cases make difficult reusing of designed techniques in approach consist from full set of comparison operations starting from analysis of software models designed in different modeling environments and finishing by visualization of these differences in modeling environment.

This idea is repeated by authors of survey [13]. "There is still much room for maturity in model comparison and it is an important area that must be in the minds of MDE supporters, as it has many benefits and is widely-applicable". Thus, that task of design an approach for comparison of software model created in different modeling environments is actual.

3. Task and Challenges

Task: Propose an algorithm and software tool for comparison software models of the same type. Software models may be designed in different modeling environments, namely Papyrus and Microsoft Visual Studio.

Ground the choice of analytical foundation for software model representation to obtain precise comparison results. This foundation should satisfy ***the next challenges***:

- to support description of software model structure considering all its details;
- to provide using of simple logical operations for performing comparison operation;
- to obtain comparison results with high precision.

Ground the choice of software modeling environment for visualization of comparison results. Such software modeling environment should satisfy ***the next challenges:***

- to support application lifecycle management (software modeling, coding, collaboration between stakeholders, etc.).
- to support stack of technologies:
  - for processing of XMI files, in which software model and information about its elements placement are stored;
  - for realization of proposed foundations for software models' comparison;
  - for setting custom visualization features of software models (custom representation for software model objects, possibility to set up custom coloring schemas for software model links and objects).

4. Proposed Approach

Proposed approach is described in the two next points, namely software model representation and software models' comparison technique.

4.1 Software model representation

Main definitions and denotations related to software model representation are described in the Table 1.

Graph representation of software model is not newapproach. It allows choosing necessary software model part for further processing in a flexible manner. In case of performing comparison operation, several elementary sub-graphs (directly linked or not) can be selected. (Definition of sub-graph is represented in the Table 1).

Table 1. Main definitions and analytical representation aimed to perform software model comparison operation

Concept	Explanation and analytical representation of concept
Software model	According to standard UML 2.5 Software Model (SM) is a UML diagram. Denote software model as SM and SM of some type as SM_{type} where type=use case, type=class, ect.
Software model representation	Graph representation [6] is chosen. $SM = (O_{type}, L_{type})$ (1) where O_{type} – a set of software model objects that are used in SM_{type} notation. . Objects are elements of software model notations that can be expressed as graph vertexes. L_{type} – a set of software model links that are used in SM_{type} notation . Links are elements of software model notation that can be expressed as graph edges.
Elementary sub-graph	Part of graph, consisting of two linked vertexes. Denote elementary sub-graph as: $e = (o_1, l, o_2)$ (2) where $o_1, o_2 \in O$ are software model objects linked by link $l \in L$.
Set of elementary sub-graphs	All sub-graphs of software model that contain all its objects and links. $E = \{e_1, e_2, ..., e_n\}, n = \|E\|$ (3)

4.2 Software models' comparison technique

This technique is considered for two software models.

1. The first software model is represented as a set of elementary sub-graphs (3). Denote this set as $E_1 = \{e_{1,1}, e_{1,2}, ..., e_{1,n}\}, n = |E_1|$
2. The second software model is represented as a set of elementary sub-graphs too. Denote this set as $E_2 = \{e_{2,1}, e_{2,2}, ..., e_{2,m}\}, m = |E_2|$
3. The first elementary sub-graph $e_{1,1} \in E_1$ is compared with all elementary sub-graphs $e_{2,j} \in E_2, j = 1, \ldots, |E_2|$.

If $e_{1,1} = e_{2,j},\ j\ in\ \{1, ..., |E_2|\}$ then these two elementary sub-graphs are deleted from the sets E_1 and E_2.

Two elementary sub-graphs $e_1 = (o_1, l_1, o_2)$ and $e_2 = (o_3, l_2, o_4)$ are considered equal if the next three conditions are satisfied:

- o_1 and o_3 are the same objects;
- o_2 and o_4 are the same objects;
- l_1 and l_2 are links of the same type.

4. Previous point is repeated for the all elementary sub-graphs of the set E_1.
5. As a result, the unique elementary sub-graphs remain in the sets E_1 and E_2.

5. Development of Software Models' Comparison Tool

In order to meet challenges to software models' comparison tool it is necessary to perform the next tasks:

- Ground the choice of development environment and tools for XMI files processing;
- Propose the algorithm for extracting software model structure from XMI file;
- Develop of software model comparison algorithm;
- Design software architecture.

5.1 Grounding the choice of development environment and tools for XMI files processing

Many environments store UML diagrams in XMI (XML Metadata Interchange) format. This standard is adopted by OMG for serializing and exchanging UML and MOF models. But different modeling environments adopt it with peculiarities and there is no possibility to use the same compiled plug-ins or tools to process (compare, merge, or perform other operation) software models designed in different modeling environments.

It defines the actuality of task: to design own software tool for software models' comparison.

There are several techniques for XMI filetext representation analysis [22]. Regular expression (Regex) engine provides a special notation for finding necessary elements in text. However, Regex is not convenient tool to extract information from the text with strict and specific structure as the XML one. Other downsides are: low level of extensibility, readability, and therefore increased cost of maintenance.

The other solution is a "Language INtegrated Queries" (LINQ) featured in .NET platform: LINQ queries allow easy manipulation of XML documents via elements of functional programming like Lambda expressions. Syntax of LINQ queries matches predicates expressions structure.

Therefore, to process software model LINQ technology is chosen. This decision defines the overall technological stack of the project - the cross-platform and open source .NET Core Framework.

Being based on a cross-platform technology, application of plug-in for UML diagram verification may be used in many different ways and embedded into various different applications and systems. Therefore, the core functionality should be implemented as a portable library in a .NET Standard format. It provides a full-featured transformation and analysis of Application Program Interface (API) for any types of applications: desktop, web, and command-line. Web application allows users to easily analyze diagrams and manage the output of results.

These technologies are built into Microsoft Visual Studio that is an Application Lifecycle Management environment supporting team development by means of using Team Foundation Server.

5.2 Algorithm for extracting software model structure from XMI file

In order to convert an XML file into a set of elementary sub-graphs the next technique is applied:

1. Determine the modeling environment in which software model was designed. Supported modeling environments are Microsoft Visual Studio and Papyrus modeling environment[16].
2. Determine UML diagram type.
3. Apply special LINQ queries to extract all UML diagram entities when UML diagram is designed in determined modeling environment.
4. Apply special LINQ queries to extract all UML diagram connections between these entities.
5. Compose extracted UML diagram entities and connections between them into set of elementary sub-graphs.

The sequence diagram for algorithm of extracting information from XMI file is represented in the figure 1.

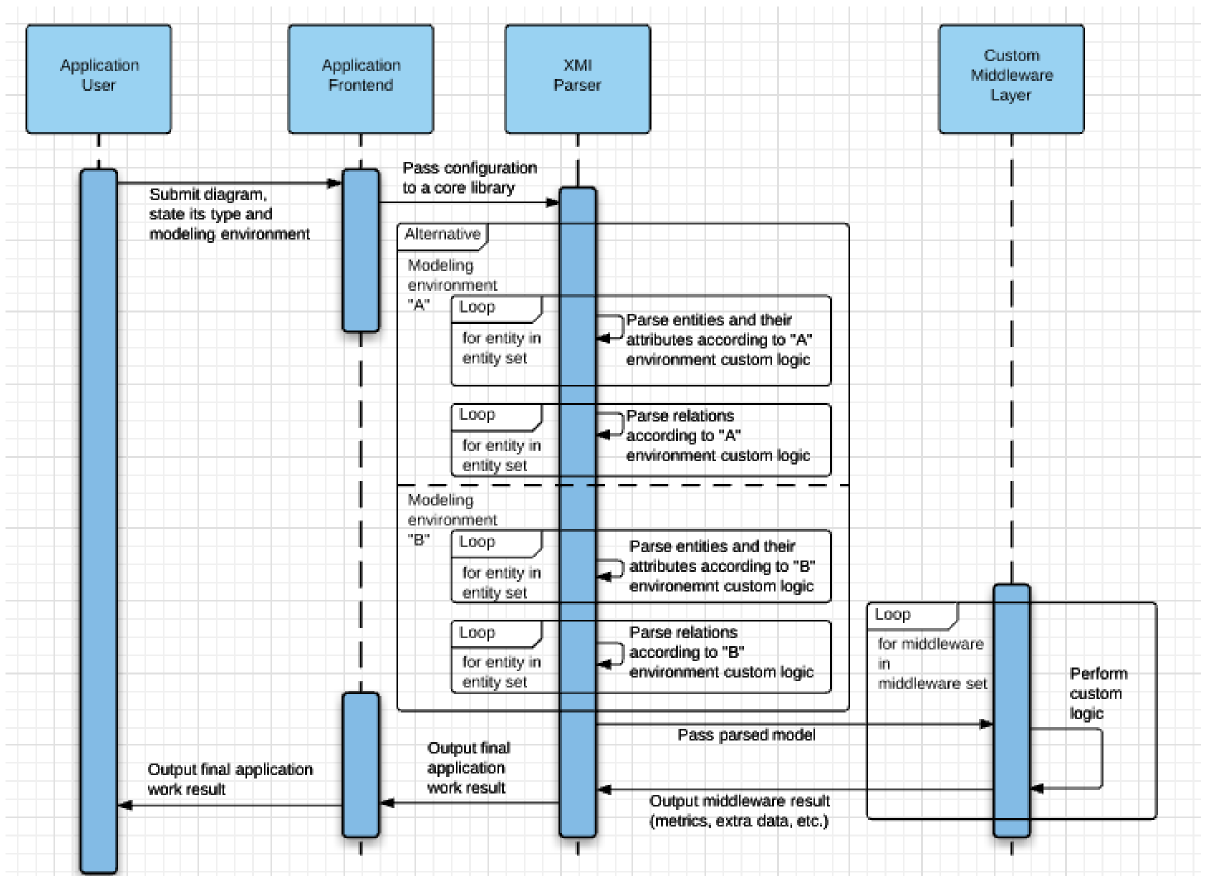

Application User
Application Frontend
XMI Parser
Custom Middleware Layer
Submit diagram, state its type and modeling environment
Pass configuration to a core library
Alternative
Modeling environment "A"
Loop
for entity in entity set
Parse entities and their attributes according to "A" environment custom logic
Loop
for entity in entity set
Parse relations according to "A" environment custom logic
Modeling environment "B"
Loop
for entity in entity set
Parse entities and their attributes according to "B" environemnt custom logic
Loop
for entity in entity set
Parse relations according to "B" environment custom logic
Loop
for middleware in middleware set
Perform custom logic
Pass parsed model
Output middleware result (metrics, extra data, etc.)
Output final application work result
Output final application work result

Figure 1. Sequence diagram for text representation of software model

Figure is taken from [27]

5.3 Development of software models' comparison algorithm.

Software models' comparison algorithm based on proposed technique consists of the next steps:

1. Prove that two software models are the same type.
2. Prove that at least one of the software models is designed in Microsoft Visual Studio. The other software model can be designed in any of Papyrus modeling environment or Microsoft Visual Studio.
3. Form the set of elementary sub-graphs from the first software model.
4. Form the set of elementary sub-graphs fromthe second software model.
5. Compare two sets of elementary sub-graphs according to software model comparison technique.
6. Modify Microsoft Visual Studio "*.layout" file by storing information about software models' differences using custom objects representation or different coloring schemas for UML model links.
7. Open modified UML diagram in Microsoft Visual Studio to watch and analyze comparing results.

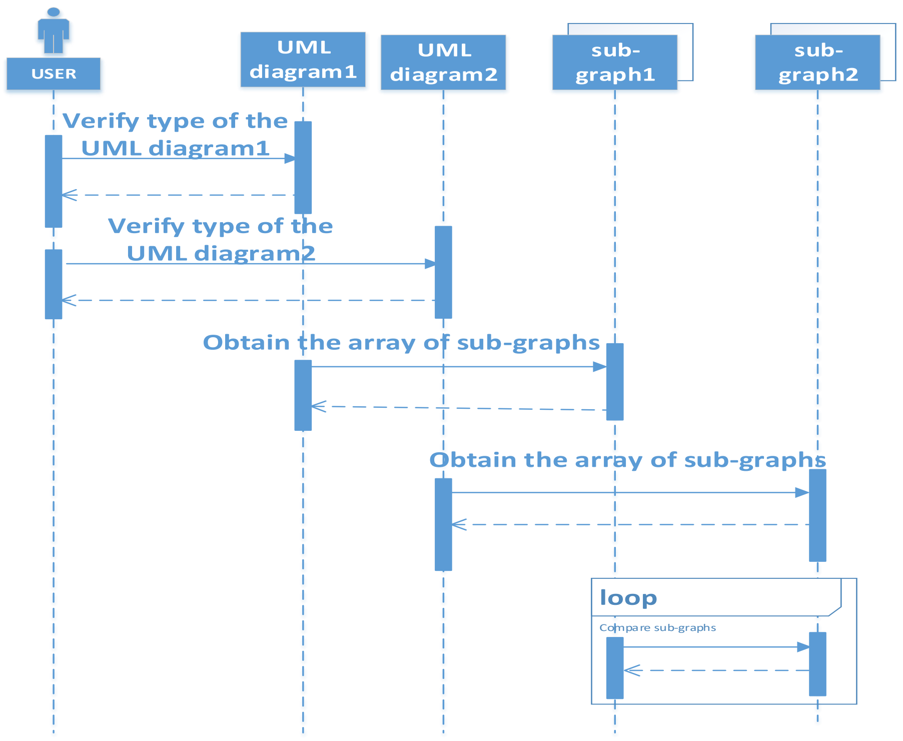

USER
UML diagram1
UML diagram2
sub-graph1
sub-graph2
Verify type of the UML diagram1
Verify type of the UML diagram2
Obtain the array of sub-graphs
Obtain the array of sub-graphs
loop
Compare sub-graphs

Figure 2. Sequence diagram for text representation of software model

5.4 Designing software architecture for realization of software models' comparison technique

Software architecture of models' comparison tool is represented on the Figure 3. Package "UML_models" stores information about software models. For performing comparison operation, software model is represented as a set of elementary sub-graphs. Class "UML_diagram" storesinformation about model in a List of elementary sub-graphs. In constructor of class “UML_diagram”, list of elementary sub-graphs by means of LINQ queries are formed.These LINQ queries are stored in the classes “Parse_Use_Case_Visual_Studio”and “Parse_Use_Case_Papyrus”. In order to satisfy Liskov Substitution SOLID design principle [15], class “UML_diagram” does not link directly with these classes. The interconnection is performed through base class “ParseUML” that has virtual methods “ConstructSub_Graphs()”,”GetConnectors()”, and ”GetObjects()”. These operations are universal for processing of UML diagrams of any type. But concrete LINQ query will be placed in the class inheritor in order to satisfy Open-Closed SOLID design principle[15]. Package “Compare” contains class “SM_Compare”that performs comparison operation for two software models. Comparison operation is performed in the constructor of the class “SM_Compare”. This constructor takes two UML models as input parameters and forms two lists of elementary sub-graphs that are unique for the first and seconds software models. Class “Visual” stores changes to “*.layout” file of UML diagram.

Proposed architecture is extensible. In order to add new types of UML diagrams, theenumerations “Graph_Objects” and “Graph_Links” are completed by new elements. Then,new classes with specific LINQ queries are inherited from “ParseUML” class.

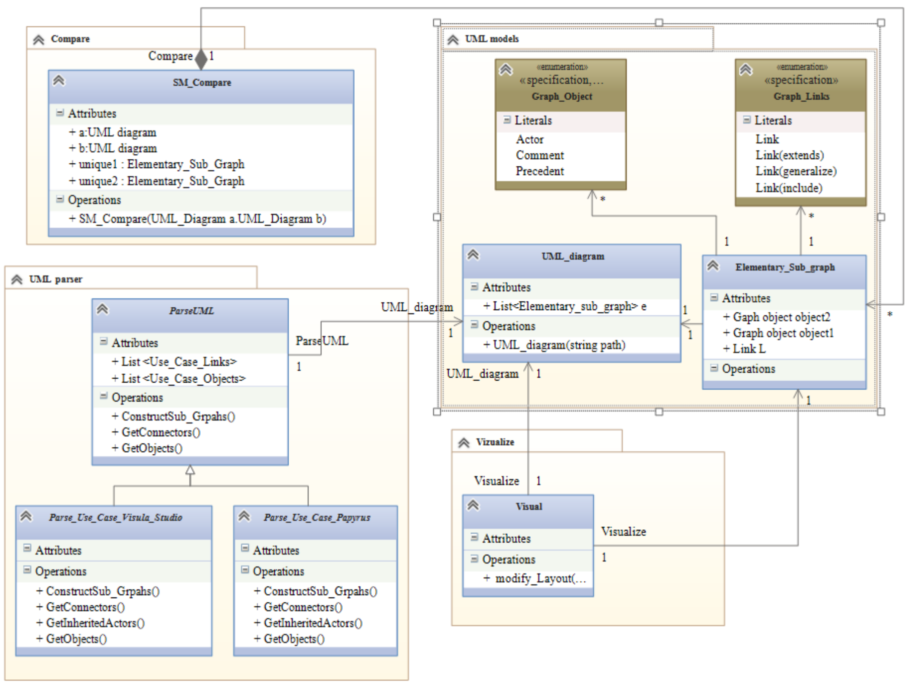
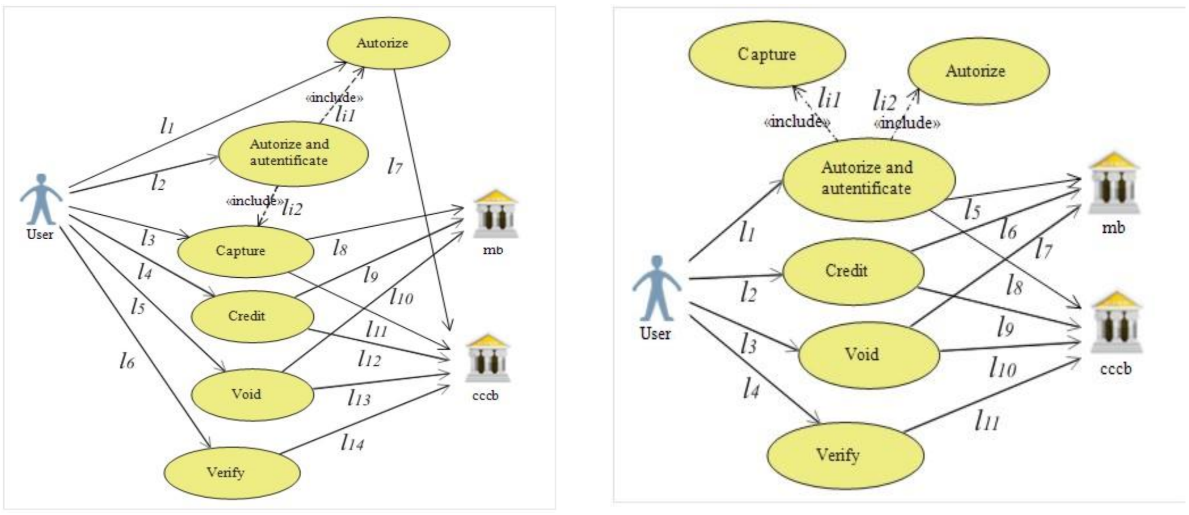


Figure 3: Software architecture of model comparison tool

6. Case Study

Consider two use-case diagrams that are obtained after two SCRUM-meetings (Figure 4).



a) b)

Figure 4 Examples of Use Cases obtained in two different scrum iterations

The first Use Case (Figure 4.a) was obtained in the previous software development iteration. The second one (Figure 4.b) is obtained after scrum meeting in the current software development iteration. Use Cases on the figure 4 are designed using template represented by reference http://www.uml-diagrams.org/examples/online-shopping-credit-cards-use-cases-example.html

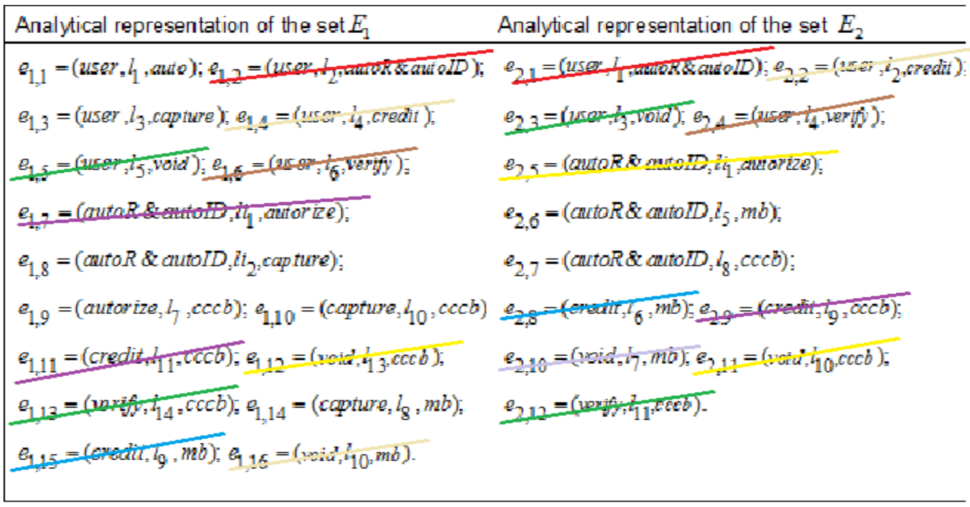
Compare these two Use Case diagrams.

1. Form sets E_1 and E_2 from the first and second use case diagrams (Table 2). *Note:* the use case "Autorize and Autotentificate" (Figure 2) further is denoted as "autoR&autoID".

Table 2 Use Case diagrams analytical representation

Analytical representation of the set E_1	Analytical representation of the set E_2
$e_{1,1} = (user, l_1, auto)$; $e_{1,2} = (user, l_2, autoR\&autoID)$;	$e_{2,1} = (user, l_1, autoR\&autoID)$; $e_{2,2} = (user, l_2, credit)$;
$e_{1,3} = (user, l_3, capture)$; $e_{1,4} = (user, l_4, credit)$;	$e_{2,3} = (user, l_3, void)$; $e_{2,4} = (user, l_4, verify)$;
$e_{1,5} = (user, l_5, void)$; $e_{1,6} = (user, l_6, verify)$;	$e_{2,5} = (autoR\&autoID, li_1, autorize)$;
$e_{1,7} = (autoR\&autoID, li_1, autorize)$;	$e_{2,6} = (autoR\&autoID, l_5, mb)$;
$e_{1,8} = (autoR\&autoID, li_2, capture)$;	$e_{2,7} = (autoR\&autoID, l_8, cccb)$;
$e_{1,9} = (autorize, l_7, cccb)$; $e_{1,10} = (capture, l_{10}, cccb)$;	$e_{2,8} = (credit, l_6, mb)$; $e_{2.9} = (credit, l_9, cccb)$;
$e_{1,11} = (credit, l_{11}, cccb)$; $e_{1,12} = (void, l_{13}, cccb)$;	$e_{2,10} = (void, l_7, mb)$; $e_{2,11} = (void, l_{10}, cccb)$;
$e_{1,13} = (verify, l_{14}, cccb)$; $e_{1,14} = (capture, l_8, mb)$;	$e_{2,12} = (verify, l_{11}, cccb)$.
$e_{1,15} = (credit, l_9, mb)$; $e_{1,16} = (void, l_{10}, mb)$.	

Perform comparison operation as it is described in the point “Software models’ comparison technique” by means of deleting the same elementary sub-graphs from two sets E_1 and E_2 .In analytical representation deleting of the elementary sub-graphs will cause in the case logical operation “and” with one elementary sub-graph from the one setgets “one” with any elementary sub-graph from the other set. Other words the same elementary sub-graphs from different sets are deleted. Elementary sub-graphs considered equal in case names and types of objects and links are the same. Visual interpretation of deleting is shown in the figure 5.



Analytical representation of the set E_1	Analytical representation of the set E_2
$e_{1,1} = (user, l_1, auto)$; ~~$e_{1,2} = (user, l_2, autoR \& autoID)$~~;	~~$e_{2,1} = (user, l_1, autoR \& autoID)$~~; ~~$e_{2,2} = (user, l_2, credit)$~~;
$e_{1,3} = (user, l_3, capture)$; ~~$e_{1,4} = (user, l_4, credit)$~~;	~~$e_{2,3} = (user, l_3, void)$~~; ~~$e_{2,4} = (user, l_4, verify)$~~;
~~$e_{1,5} = (user, l_5, void)$~~; ~~$e_{1,6} = (user, l_6, verify)$~~;	~~$e_{2,5} = (autoR \& autoID, li_1, autorize)$~~;
~~$e_{1,7} = (autoR \& autoID, li_1, autorize)$~~;	$e_{2,6} = (autoR \& autoID, l_5, mb)$;
$e_{1,8} = (autoR \& autoID, li_2, capture)$;	$e_{2,7} = (autoR \& autoID, l_8, cccb)$;
$e_{1,9} = (autorize, l_7, cccb)$; $e_{1,10} = (capture, l_{10}, cccb)$	~~$e_{2,8} = (credit, l_6, mb)$~~; ~~$e_{2,9} = (credit, l_9, cccb)$~~;
~~$e_{1,11} = (credit, l_{11}, cccb)$~~; ~~$e_{1,12} = (void, l_{13}, cccb)$~~;	~~$e_{2,10} = (void, l_7, mb)$~~; ~~$e_{2,11} = (void, l_{10}, cccb)$~~;
~~$e_{1,13} = (verify, l_{14}, cccb)$~~; $e_{1,14} = (capture, l_8, mb)$;	~~$e_{2,12} = (verify, l_{11}, cccb)$~~.
~~$e_{1,15} = (credit, l_9, mb)$~~; ~~$e_{1,16} = (void, l_{10}, mb)$~~.	

Figure 5Visualizing of analytical process of deleting the same elementary sub-graphs

3. Those elementary sub-graphs that are unique for E_1 and E_2 remain in the lists of elementary sub-graphs (Table 3).

Table 3 Unique elementary sub-graphs of two Use Case diagrams

Unique elementary sub-graphs of E_1	Unique elementary sub-graphs of E_2
$e_{1,1} = ((user, l_1, autorize))$;	$e_{2,6} = (autoR \& autoID, l_5, mb)$;
$e_{1,3} = (user, l_3, capture)$;	$e_{2,7} = (autoR \& autoID, l_8, cccb)$.
$e_{1,9} = (autorize, l_7, cccb)$;	
$e_{1,10} = (capture, l_{10}, cccb)$;	
$e_{1,12} = (capture, l_{11}, mb)$.	

4. Visualization of Use case diagrams difference is represented at the Figure 5. Unique parts of the first use case diagram are marked as blue, the seconds use case diagram respectively green. Notice hat results of analytical comparison (Table 3) and software tool working (Figure 6) are the same

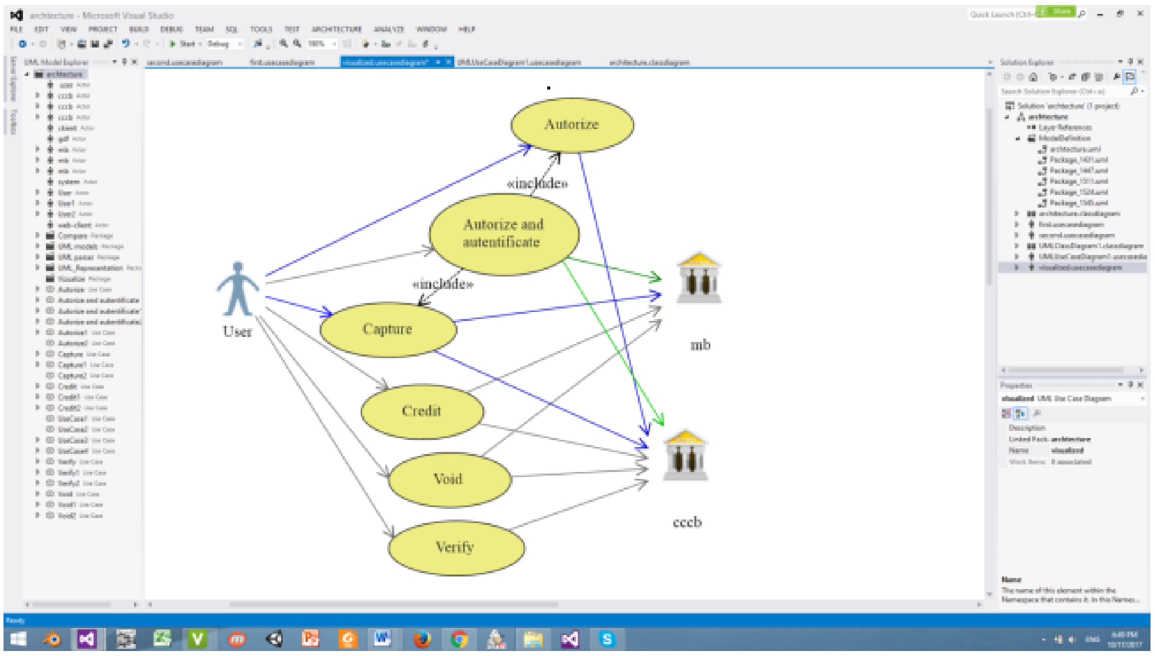


Figure 6Visualization of software model comparison technique results

7. Conclusion

Solution proposed in this paper integrates both analytical foundations of software model representation and comparison as well as software tool designed to implement proposed approach.

To describe software model graph representation (1)-(3) is used [6]. Such representation allows (i) considering all details of software model; (ii) providing simple logical operations for elementary sub-graph comparison; (iii) obtaining comparison results with high precision.

Then approach for software model comparison by means of elementary sub-graph analysis of two different software models is represented. It is based on performing sequence of operations for elementary sub-graphs matching.

Software tool supports the next features: (i) restores information of software models' elementary sub-graphs, designed in Papyrus of Microsoft Visual Studio; (ii) performs software models' comparison; (iii) provide visual representation of non-matching elements in both software models in Microsoft Visual Studio.

Visualization algorithm modifies "*.layout" file of UML diagram in Microsoft Visual Studio. Modification means reaching UML diagram by specific denotations, for example adding custom pictures for drawing software model objects or color schemas for marking different UML diagrams elements (Figure 3).

Using this tool simplifies the process of analyzing software models in comparison with approaches that represent only textual description of UML diagrams' differences in plain text [1], [2],[3],or in analytical expressions. Proposed tool will speed up performing many operations in requirement analysis, software designing, testing, and reengineering software development life cycle processes.

Grounding the choice of software technologies stack, used in software tool, is represented. Concluding: (i) LINQ queries for processing software model XMI files are used; (ii) software tool using Microsoft Visual Studio for models creation, storing, implementing comparison technique, and visualizing of comparison result, is designed, (iii) .Net Core, as target cross-platform technology supporting cross-platform compilation, is chosen.

8. Further Research

Modify proposed approach and software tool for comparing software models of different types that have common elements in their notations (For example notations of Use Case, Communication, and Sequence diagrams contain “Actors”). Develop visualization technique to consider differences in more than two software models allowing user to consider whether to see whole UML diagram or some parts of it, according to history of changing, considering limitations discussed in paper [4]. Develop a software tool based on proposed visualization technique for software model versioning.

References

[1]. Alanen, M. and Porres, I. (2003). Difference and union of models. In UML, pages 2–17.

[2]. Alanen, M.and Porres, I. (2005).Version control of software models. Advances in UML and XML-Based Software Evolution, pages 47–70.

[3]. Brun, C.and Pierantonio, A. (2008). Model differences in the Eclipse modelling framework. The European Journal for the Informatics Professional, pages 29–34.

[4]. Chebanyuk E., Markov K., 2015. Software model cognitive value. International Journal “Information Theories and Applications”, Vol. 22, Number 4, ITHEA 2015, p. 338-356. http://www.foibg.com/ijita/vol22/ijita22-04-p04.pdf

[5]. Chebanyuk E., Markov K., 2016. Model of problem domain “Model-driven architecture formal methods and approaches” International Journal “Information Content and Processing”, Vol. 22, Number 4, ITHEA 2016, p.202-222. http://www.foibg.com/ijicp/vol03/ijicp03-03-p01.pdf

[6]. Chomsky, N. 1957. The book. Syntactic Structures. Mouton publishers, Eilenberg: Mac Lane The, Hague, 1945 - 1957. ISBN 90 279 3385 5. p.107.

[7]. Control in Model Driven Software Development. OOPSLA/GPCE: Best Practices for Model-Driven Software Development

[8]. Farail, P., Gaufillet, P., Canals,A., LeCamus, C., Sciamma, D., Michel, P., Cregut, X., and Pantel, M. (2006). The top cased project: a toolkit in open source for critical aeronautic systems design. ERTS, pages1–8, electronic.

[9]. ISO/IEC 12207:2008(E) Systems and software engineering — Software life cycle processes

[10]. ISO/IEC/IEEE 42010:2011(E)Systems and software engineering — Architecture description http://cabibbo.dia.uniroma3.it/asw/altrui/iso-iec-ieee-42010-2011.pdf

[11]. Kolovos, D., Paige, R., and Polack, F. (2006). Model comparison: a foundation for model composition and model transformation testing. In IWGIMM, pages 13–20.

[12]. Letkeman, K. (2007). Comparing and merging UML models in IBM Rational Software Architect: Part 7. http://www.ibm.com/developerworks/rational/library/07/0410letkeman/

[13]. Matthew S. and Cordy J. (2013). A Survey of Model Comparison Approaches and Applications. In Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development (MODELSWARD-2013), pages 265-277 ISBN: 978-989-8565-42- DOI: 10.5220/0004311102650277

[14]. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., and Zave, P.(2007). Matching and merging of state charts specifications. In ICSE, pages 54–64.

[15]. Ocampo, (2009) Joe Ocampo, Jason Meridth, Chad Myers, Sean Chambers, Ray Houston, Jimmy Bogard, Gabriel Schenker, and Derick Bailey, Pablo's solid software development, 2009 e-book access mode : https://lostechies.com/wp-content/uploads/2011/03/pablos_solid_ebook.pdf

[16]. Papyrus, (2012) Papyrus, 2012. Available from World Web:www.papyrusuml.org. Unified Modeling Language (UML), 2012.

[17]. Pham, N., Nguyen, H., Nguyen,T., Al-Kofahi, J., and Nguyen, T.(2009). Complete and accurate clone detection in graph-based models. In ICSE, pages 276–286.

[18]. Reiter, T., Altmanninger, K., Bergmayr, A., Schwinger, W.,and Kotsis, G. (2007). Models in conflict-detection of semantic conflict sinmodel-based development. In MDEIS, pages29–40.

[19]. Selonen, P. and Kettunen, M. (2007). Metamodel-based inference of inter-model correspondence. In ECSMR, pages 71–80.

[20]. Soto, M. and Munch, J. (2006). Process model difference analysis for supporting process evolution. Software Process Improvement, pages 123–134.

[21]. Xing, Z. and Stroulia, E. (2005). UML Diff: an algorithm for object-oriented design differencing. In ASE, pages 54–65.

[22]. XMI, 2015 XML Metadata Interchange http:// www.omg.org/spec/XMI/2.5.1/

[23]. Yuan Wang, David J. DeWitt, Jin-Yi Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In: Proceedings 19th International Conference on Data Engineering, Bangalore, India, India 5-8 March, 2003 Page(s):519 - 530 http://pages.cs.wisc.edu/~yuanwang/papers/xdiff.pdf DOI 10.1109/ICDE.2003.1260818

[24]. Yuehua Lin , Jing Zhang , Jeff Gray Model comparison: A key challenge for transformation testing and version control in model driven software development (2004)

[25]. El-Licy, F. A. (2016). Paired Scrum for Large Projects. *Egyptian Computer Science Journal (ISSN-1110-2586)*, *40*(1).

[26]. Sanaa, H., Afifi, W. A., & Darwish, N. R. (2016). The Goal Questions Metrics for Agile Business Intelligence. *Egyptian Computer Science Journal*, *40*(2).

[27]. Chebanyuk O., Mironov Yu.(2017) An approach of obtaining initial informationfor software models analysis. *International journal. Informationalcontent and processing*. Vol. 4, number 2, p.114-143