Reverse Engineering of UML Use Case Model from Website Usage Records

Vaidotas Drungilas
Department of Information Systems,
Kaunas University of Technology
Informatics faculty
Kaunas, Lithuania
vaidotas.drungilas@ktu.lt

Lina Čeponienė
Department of Information Systems,
Kaunas University of Technology
Informatics faculty
Kaunas, Lithuania
lina.ceponiene@ktu.lt

Mantas Jurgelaitis
Department of Information Systems,
Kaunas University of Technology
Informatics faculty
Kaunas, Lithuania
mantas.jurgelaitis@ktu.lt

***Abstract*—Though UML is rather commonly used for modelling various software systems, if not properly maintained, UML models could lose their practical value. Fixing the mismatch between documentation and the current state of software, requires significant effort from development team. This also applies to systems that have no documentation at all or legacy systems, which documentation is not available. Reverse engineering can be used for generating UML diagrams for existing systems. In this paper we present a method for reverse engineering UML Use Case model from websites. This method enables generating UML Use Case and Activity diagrams from the recorded user activity in the website.**

Keywords—UML; reverse engineering; website; Use Case diagram; Activity diagram.

I. Introduction

Unified Modelling Language (UML) is rather commonly used for modelling various software systems [1]. UML is applied not only during development of complex software systems but also during maintenance of the systems in use. As for deployed systems that require support and updates, models help to analyze and understand inner structure and functionality of a system [2]. Most of the models and documentation are usually created during initial software development stages. If not properly maintained these models lose their practical value. An example of improper maintenance could be a situation when the final product receives updates and new features, without updating documentation and leaving it obsolete. While using this kind of obsolete documentation, maintenance of software and introducing new features becomes more difficult. Fixing the mismatch between documentation and the current state of software, requires significant effort from development team. This also applies to systems that have no documentation at all or legacy systems, which documentation is not available.

Websites more than any other type of software demand constant updates and fixes to meet changing user demand and to beat harsh competition [3]. This demand and competition puts pressure on web developers to implement changes as fast as possible, without wasting valuable resources and time. As demand grows, website developers tend to concentrate on maintaining created and implementing new software features rather than spending time for updating the documentation. To match the tendency of directing most of the effort into implementation stage, Agile project management methodologies have a tendency to be rather popular among web developers. Software products that are built using Agile methodology, usually do not concentrate on having detailed documentation. Consequently, in this paper we tackle a problem of increasing efficiency of modelling process for websites and suggest reverse engineering as a possible solution.

Reverse engineering is the process of analyzing a system to identify its structure and behavior in order to create its visual representation [4]. Reverse engineering can be used to understand how software works and to transform some kind of static information, like program code, into models and documentation [5].

Reverse engineering can also be used for generating UML diagrams [6] [7] [8]. In this paper we present a method for reverse engineering UML Use Case model from websites. This method enables generating UML Use Case and Activity diagrams from the recorded user activity in the website and websites' HTML files.

The rest of the paper is organized as follows. The second section presents related work in the field of reverse engineering UML diagrams. The third section is dedicated to describing our proposed method for generating UML diagrams from registered user actions. Section four describes the prototype developed for our proposed method. The fifth section presents the results of evaluation of the prototype by applying it for a particular website. The last section summarizes the findings of our research and discusses the future work.

II. Related work

Reverse engineering of UML models is rather common in the field of software engineering. Reverse engineering can greatly reduce the effort required to construct UML diagrams. UML diagrams can be divided into two categories, one describes structure of the software system, and the other defines its behavior [9].

Structural UML diagrams can be reverse engineered from code or other static structure. Class diagrams can be easily transformed from static code [8], [10]. Many tools support this option through plugins or default functionality, e.g. Eclipse [11] or Visual Paradigm [12].

On the other hand, reverse engineering of behavioral diagrams, like Use Case and Activity diagrams, is not so commonly implemented and used. Nevertheless, there have been significant effort to create a working method for reverse engineering of UML behavioral diagrams [6] [7] [13] [14].

El-Attar and Miller proposed a method to reverse engineer Use Case models from structured Use Case descriptions [7]. This method requires structured text as an input which is analyzed and transformed into the Use Case diagram. This method if used correctly could greatly improve consistency of software documentation and would allow creating precise and unambiguous Use Case models. The best result using this method could be in early development stages. On the other hand, while using Agile methodologies this method would not be the best choice because it requires additional effort for creating and formatting Use Case specifications.

Another method for reverse engineering UML diagrams has been proposed by Muhairat [6]. It uses event table as an input for generating Use Case diagram. Event table, as defined in [6], has four main elements: event, source of event, action and object. These main elements are later transformed, using the proposed process, which consists of actor identification, relation between actors' identification, use case identification, relation between use cases identification and integration of all found elements. Just like [7] method, this method requires a lot of effort for creating sufficiently detailed event table to generate informative Use Case model. This method could be the most useful during requirement analysis phase.

Much can be learned not only form reverse engineering Use Case diagrams, but also from reverse engineering other behavioral diagrams. An excellent example of reverse engineering behavioral diagrams is presented in research by Ziadi, da Silva, Hillah, and Ziane [13]. They proposed an approach how to fully dynamically reverse engineer UML Sequence diagrams. This dynamic method is intended for the systems where static code analysis is not applicable directly. Approach also defined how to extract the traces of a working system. This idea is used as a basis in one of the steps of our proposed method for extracting website usage information.

Di Lucca, Fasolino and Tramontana proposed a specialized tool specifically intended for website reverse engineering, called WARE [14]. This tool is capable to reverse engineer UML Use Case and Sequence diagrams as well as Class diagrams. But instead of working with dynamic content, this tool uses static code as an input for reverse engineering. This tool is only applicable in situations where the full access to source code is granted. In contrast, our research focuses on reverse engineering UML behavioral diagrams independently of the availability of the websites' source code.

Most of the research conducted in the area of reverse engineering Use Case model is not intended particularly for websites. Our approach focuses on reverse engineering UML behavioral diagrams from websites, specifically from the information recorded during website usage. Our proposal is based on the idea that websites use a common architecture which can be used to extract information about user activities.

III. Proposed method for Reverse Engineering UML Use Case model

As UML Use Case model provides detailed overview of systems functionality, it is one of the main components of high quality system documentation [15]. Our approach in reverse engineering UML Use Case model should provide ability to flexibly analyze web applications, and transform analysis results into UML Use Case model. The created Use Case model should include Use Case diagram along with each Use Case specified by an Activity diagram defining the functionality of that Use Case.

The proposed method consists of two main steps for reverse engineering UML Use Case model from the selected website (Fig. 1):

1) registering usage of the analyzed system;
2) transforming registration results into XMI file.

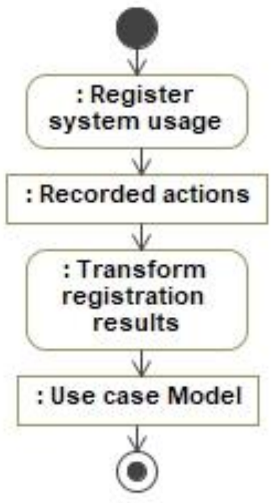


Fig. 1 Structure of the proposed method

During the first phase of reverse engineering UML Use Case model, user opens the system that will be analyzed. He inputs his role in the system and activity that he will be performing. User then continues to use system while his activity is being recorded by reverse engineering system in the background. User can input as many roles and activities as it is required to completely represent his usage of the system. Registering of user actions should be performed by a number of users that is required to cover all functionality of system. After all users register their activities, they should be able to export result files and send them to the system analyst. System analyst then should be able to merge all the result recordings together, into one full structure that represents actions performed in the analyzed system.

A. System usage registration process

The component that will be used for registering system usage should not interfere with system functionality by any way. The recording component should be able to read HTML files that user is interacting with. It should work as a background process that captures user input events, such as clicks and form submits. To describe these events correctly, recording component should also store information about HTML elements that user interacts with. These elements should be uniquely described with an identifying element. Registration component should allow user to define what kind of role he is performing in the system and to define what kind of activity he will be performing. The process of registration consists of initialization and recording steps as can be seen in Fig. 2

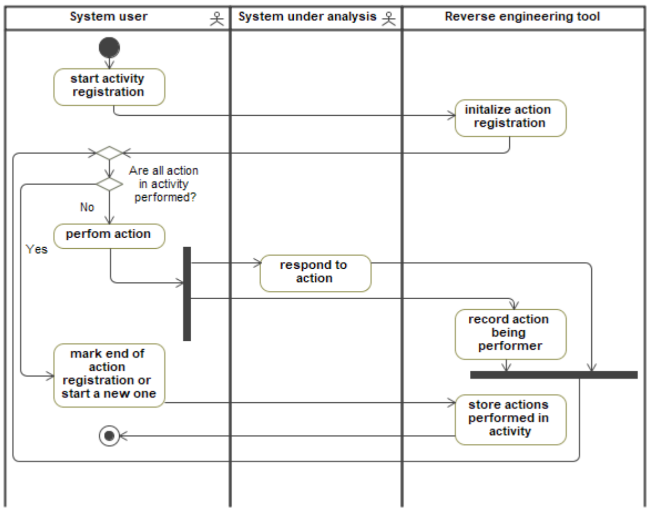

System user
System under analysis
Reverse engineering tool
start activity registration
initalize action registration
Are all action in activity performed?
No
Yes
perfom action
respond to action
record action being performer
mark end of action registration or start a new one
store actions performed in activity

Fig. 2 Activity diagram representing the process of system usage registration

B. Registration result

Registration results are then provided as an input for the component that transforms user actions into UML Use Case model. This input should be stored in a structure that has elements described in Fig. 3. This structure should store all Website URLs that user interacted with during time of recording. In addition, user should provide the name of the Role, which exists in given Website. Each Role will be performing some kind of Activity. Each Activity should be defined by Webpage that it was performed on and events that were performed in that same Webpage. Each event is described with detailed information about type of Action, and information about HTML element that was in use during that event.

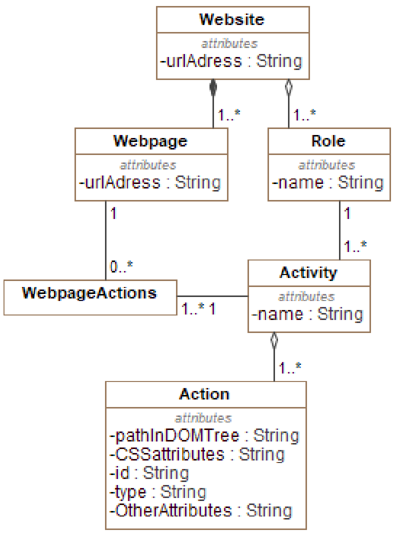


Fig. 3 The structure of the user activity registration result

C. Transformation process

Transformation process is defined in Fig. 4. In order to create a detailed UML model, transformation step should be performed. During this step, registration input is being analyzed for detecting relations between actors and use cases, also between use cases themselves.

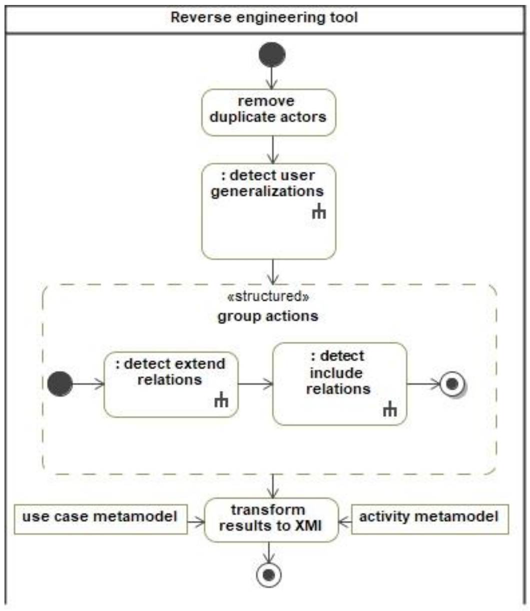


Fig. 4 Activity diagram representing the process of transformation to XMI file

Activity diagram defining the transformation (presented in Fig. 4) specifies what actions are required to transform registered results to XMI. The first action removes duplicate actors, to keep the model concise. The second action is required to detect and create generalization relations between actors. The third action performs grouping operations with the registered data. These grouping operations consist of detecting extend and include relations between use cases. The final step takes the results from all relation detection steps and creates XMI file basing on the structure of metamodels of UML Use Case and Activity diagrams.

D. Generalization relation detection

Generalization relation detection starts by scanning all registered use cases and searching for two or more actors which have matching use cases. Algorithm also checks whether it needs to create a new user, in order to display a generalization correctly. If the user creation is required, the system analyst should be prompted to input actor name for new actor. After these steps, the system creates generalization relations between the actors. As a last step, the system maps all required use cases to required actors. Generalization relation detection step is the only step that changes configuration with actors in the model, so after this step we will have the final number of actors in the model. The activity diagram describing generalization detection process is defined in Fig 5.

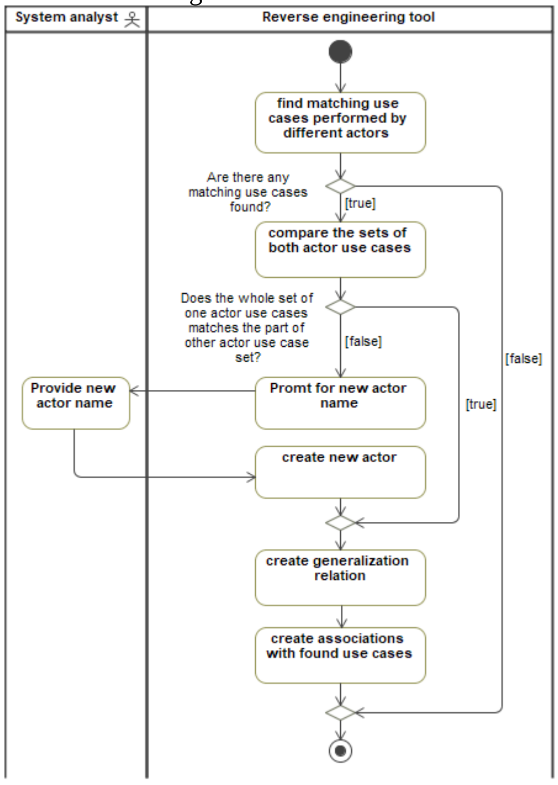

System analyst
Reverse engineering tool
find matching use cases performed by different actors
Are there any matching use cases found?
[true]
compare the sets of both actor use cases
Does the whole set of one actor use cases matches the part of other actor use case set?
[false]
[false]
Provide new actor name
Promt for new actor name
[true]
create new actor
create generalization relation
create associations with found use cases

Fig. 5 The process of detecting generalization relations

E. Detecting relations between Use Cases

Detection of use case relations like extend and include, depends on a data set provided by the user of proposed method. Our method does not detect generalization between Use Cases, only include and extend relationship. To detect extend relations, the user of a system should record the same activity on a website twice or more. If these data sets of the same activity will provide exactly the same information, detection would just discard it. Otherwise, if some differences would be found in these data sets, the proposed algorithm can create a more detailed use case model. Success of relation detection depends on an amount of data that user provides during the recording. The higher amount of recorded data should transform into a model that is more detailed and thus more informative.

1) Detection of extend type relations

To enhance the Use Case model, our proposed approach detects extend relations. All the actions required to detect extend relations are represented in Fig. 6. These extend relations are important in describing alternative scenarios in the model. Extend relation detection starts with checking all recorded action sequences and finding partially repeating actions. From these sequences, the matching parts are extracted, by comparing them to each other. At the beginning and end of the extracted sequence, decision and merge points are created respectively. Afterwards the extracted, remaining and newly created elements are merged together to create the complete activity diagram. For each path that is now separated from the main path of the activity flow by a decision point, a new use case can be created. User provides the names for these use cases and the algorithm creates them in a model. The next step in extension relation detection is creating extension relation between newly created use cases and the use case they originated from. Finally the extension points are created for the use case, which has incoming extend relations.

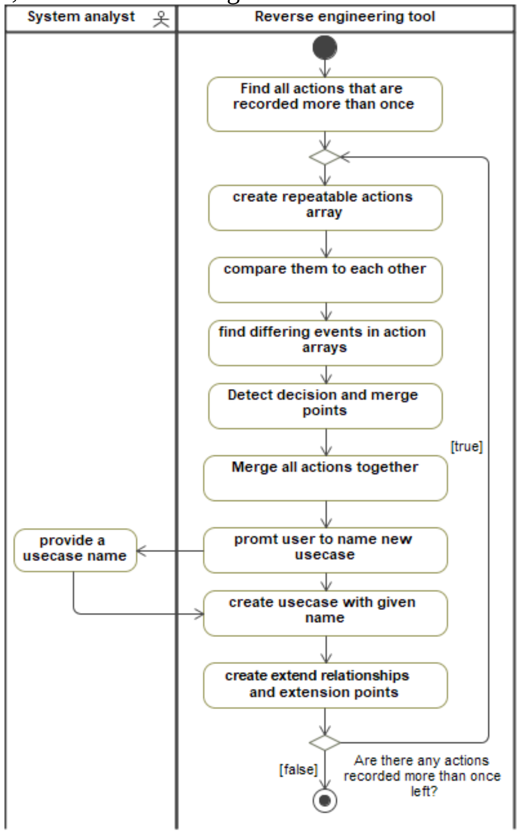

System analyst
Reverse engineering tool
Find all actions that are recorded more than once
create repeatable actions array
compare them to each other
find differing events in action arrays
Detect decision and merge points
[true]
Merge all actions together
provide a usecase name
promt user to name new usecase
create usecase with given name
create extend relationships and extension points
[false]
Are there any actions recorded more than once left?

Fig. 6 The process of detecting extend relations

2) Detection of include type relations

In order to decrease redundancy in use case model, include relations can be used between use cases. As in our method the amount of recorded data should be quite big, include relations help to reduce the number of repeated actions in the model. Include relation detection starts with finding all repeating action sequences where repeated sequence length is higher than user defined number N. For each sequence found, the algorithm starts use case creation by prompting the user to input a use case name and creating use case element with that name. These found sequences are then removed form use cases where they originated. To finalize, include type relations are created by joining newly created use cases and use cases that the sequences originated from. The process of include relation detection is presented in Fig. 7

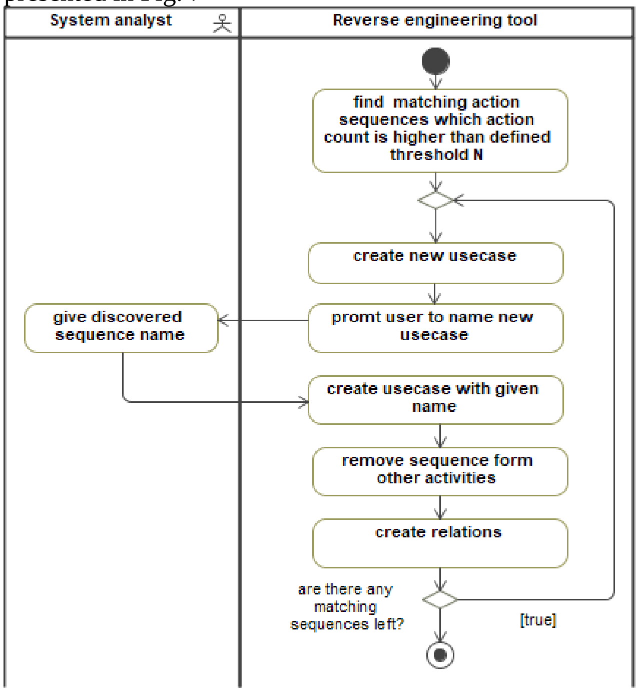

System analyst
Reverse engineering tool
find matching action sequences which action count is higher than defined threshold N
create new usecase
give discovered sequence name
promt user to name new usecase
create usecase with given name
remove sequence form other activities
create relations
are there any matching sequences left?
[true]

Fig. 7 The process of detecting include relations

F. *Transformation to XMI format*

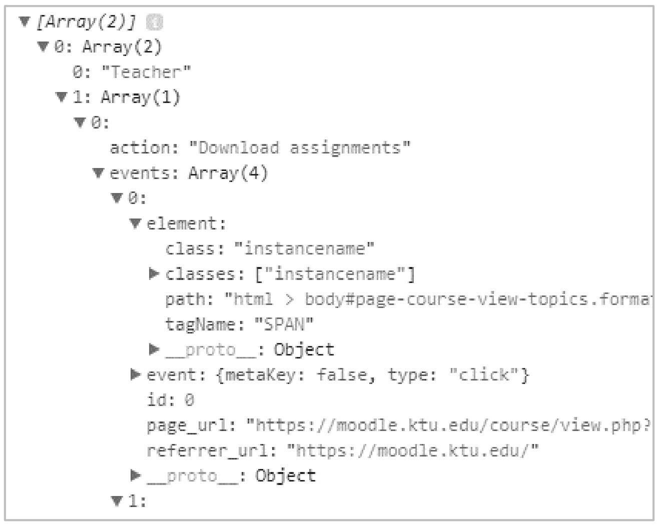
During the transformation to XMI step, all the information gathered in previous steps is transformed into Use Case and Activity diagrams. Transformation starts by creating a Use Case diagram. For each role that the user defined, an actor is created. For each activity, that the user defined, the use case is created. Moreover, the use cases that were found during detection of extension and include relationships are added to the model. All use cases and actors are joined using the detected relationship. For each use case, an Activity diagram is created. In this diagram, the algorithm creates two swimlanes, the first one for the actor that is interacting with the given use case, and the second for the system under analysis. For each recorded action our algorithm creates the action in activity diagram. Actions are named referring to the action naming rules, selecting the verb corresponding to the action performed on a HTML element as well as a noun extracted from HTML element attributes. These rules define how each action on HTML element should transform into semantically correct action name. In a systems swimlane, actions are created describing the opening of new webpages. Initial and final activity nodes are also created and all nodes and activities are joined together in a continuous flow.

G. *Transformation results.*

Results after generation are stored in XMI file. This file consists of Use Case diagram with elements that are described by UML Use Case metamodel [9]. For each use case, an activity diagram is created, representing all actions that the user performs in the system under analysis. Activity diagram is also based on UML metamodel for Activities and Actions [9]. The main elements that this method detects are roles, use cases, and actions. Relations join each of these elements: association relationship joins use cases and actors, generalization is used between actors, include and extend relations can join two use cases, and control flow relations connect the actions in activity diagrams. User can download the generated XMI file and store it as needed. As most of UML modeling tools support XMI as their import format, users should just import this file and have the working version of Use Case model.

IV. The Implemented Prototype of the proposed Method

To test whether the proposed method could be utilized in practice, a prototype has been implemented. Prototype was realized as a Chrome plugin using JavaScript. It enables users to submit information about their role in the website activity they will be performing. As user continues to use the system that is being analyzed, his actions are recorded. Recorded actions are then stored in JSON file. After user indicates that he has ended registration process, he can start transformation process. The system transforms registered JSON structure to Use Case and Activity diagrams. Example of this JSON structure is presented in *Fig. 8*.



```
▼ [Array(2)]
  ▼ 0: Array(2)
      0: "Teacher"
    ▼ 1: Array(1)
      ▼ 0:
          action: "Download assignments"
        ▼ events: Array(4)
          ▼ 0:
            ▼ element:
                class: "instancename"
              ▶ classes: ["instancename"]
                path: "html > body#page-course-view-topics.forma
                tagName: "SPAN"
              ▶ __proto__: Object
            ▶ event: {metaKey: false, type: "click"}
              id: 0
              page_url: "https://moodle.ktu.edu/course/view.php?
              referrer_url: "https://moodle.ktu.edu/"
            ▶ __proto__: Object
          ▼ 1:
```

Fig. 8 An example of JSON structure displaying use case “Download assignments data”

Afterwards this JSON file is transformed into XMI file, which can be imported into MagicDraw CASE tool as a model. This model can later be viewed, analyzed and modified by the analyst.

V. Example of UML diagrams generated using the implemented prototype

An experiment was conducted to verify whether the created prototype is capable of reverse engineering UML Use Case model. Website selected for this experiment was a virtual learning environment Moodle, customized for Kaunas University of Technology. The roles of student and teacher were analyzed.

In order to create Use Case model, student and teacher were asked to perform actions in the analyzed website. Student performed actions for uploading a file into the system. Teacher recorded a process in which he downloaded the students' submitted assignment. Fig. 9 displays the generated Use Case diagram. For this diagram, users provided two roles but the system identified one additional actor. In total three use cases were detected. The first actor was discovered by generalization relation detection step during registration result transformation. As both of the actors had to log into the system a new user named "Guest" was created. This user, as mentioned before, provides the ability to subtract the amount of excess use cases. For each of these use cases, the algorithm created an activity diagram as expected.

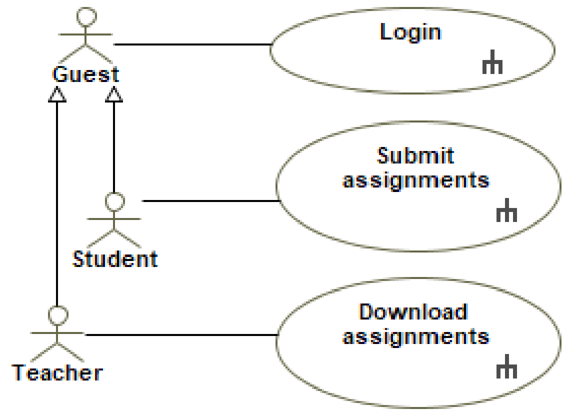
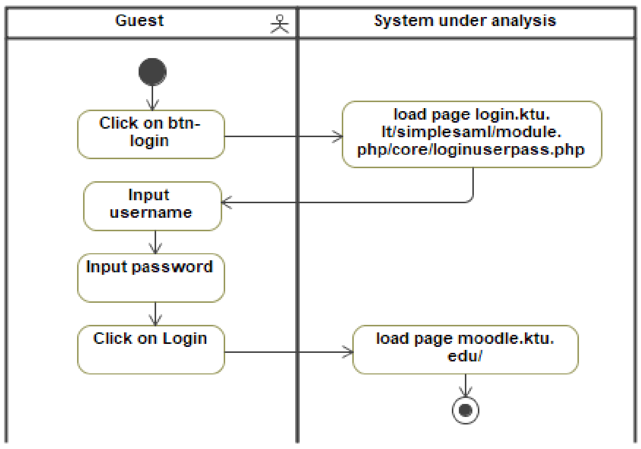


Fig. 9 The generated Use Case diagram

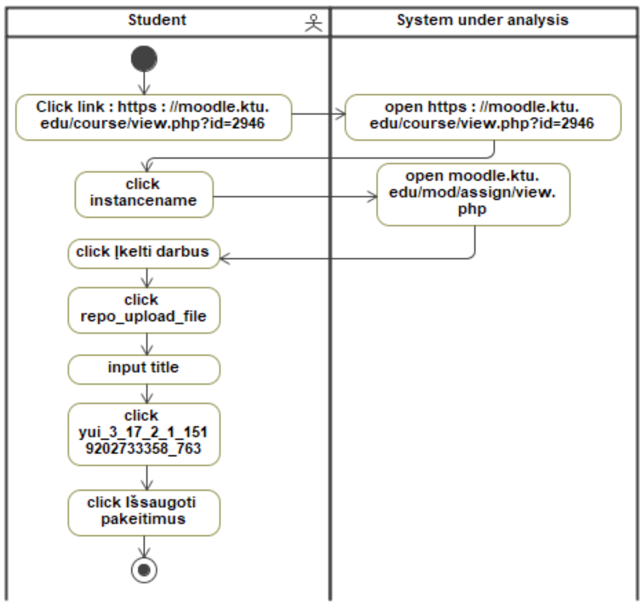
Activity diagram describing Use Case "Login" is presented in *Fig. 10*. In this diagram we can see that each non-authenticated user had to click login button styled by CSS class named "btn-login". After the button is clicked, the system transfers all non-authenticated users to unified authentication system where user fills out login fields and submits the form. To finish login use case, the user clicks Login button and is transferred to the virtual learning environment. The generated Activity diagram demonstrates the prototypes ability to specify common actions, like login and registration.



Guest
System under analysis
Click on btn-login
load page login.ktu.lt/simplesaml/module.php/core/loginuserpass.php
Input username
Input password
Click on Login
load page moodle.ktu.edu/

Fig. 10 Generated activity diagram describing *Login* use case

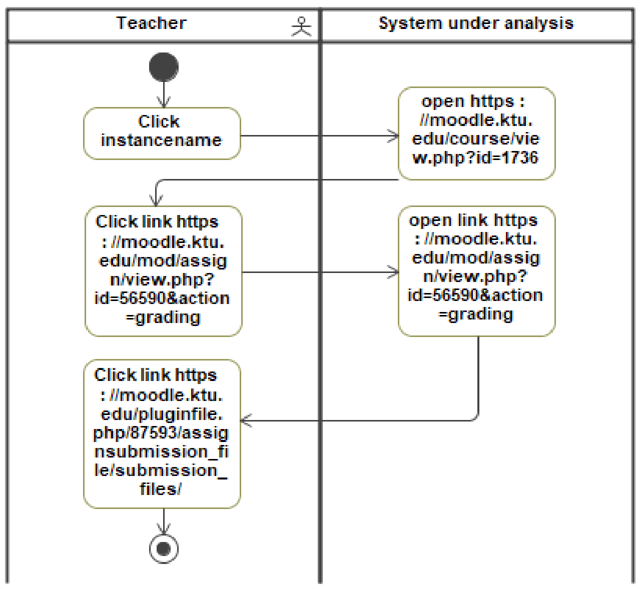
An example of generated activity diagram for use case "Submit assignments" is displayed in Fig. 11. This diagram displays interactions the user performed and webpages he opened. The diagram describes algorithms' ability to record actions, and to transform them correctly. The action naming conventions do not convey the performed action information clearly, it heavily depends on the systems configuration. Most of the students' actions in this use case were navigation through the website. One downside of using just URL changes to describe the opening of new windows in the system, is that it cannot record the opening of a modal window. The algorithms' ability to detect modal windows should be improved in the future.



Student
System under analysis
Click link : https : //moodle.ktu.edu/course/view.php?id=2946
open https : //moodle.ktu.edu/course/view.php?id=2946
click instancename
open moodle.ktu.edu/mod/assign/view.php
click |kelti darbus
click repo_upload_file
input title
click yui_3_17_2_1_1519202733358_763
click Išsaugoti pakeitimus

Fig. 11 Generated activity diagram describing *Submit assignment* use case

The example use case "Download assignment" is described in activity diagram presented in *Fig. 12*. This activity provides visual feedback, demonstrating that some of the URL naming rules should be improved. On the other hand, this activity still provides enough detail to cover all the most important actions of the use case.



Teacher
System under analysis
Click instancename
open https : //moodle.ktu. edu/course/view.php?id=1736
Click link https : //moodle.ktu. edu/mod/assign/view.php? id=56590&action =grading
open link https : //moodle.ktu. edu/mod/assign/view.php? id=56590&action =grading
Click link https : //moodle.ktu. edu/pluginfile. php/87593/assignsubmission_file/submission_files/

Fig. 12 The generated activity describing *Download assignment* use case.

The results of experiment indicate that created prototype is capable of creating UML Use Case model. Both Use Case and Activity diagrams were created describing system usage in great detail. As current prototype is only capable of detecting generalization relations, further iterations of this prototype will only increase the expressivity of generated models. Generation of activity diagrams describing each use case provide even more depth to generated UML Use Case model. As semantic value of these Activity diagrams can still be improved in future releases, generated Activity diagrams still provide enough information about system usage.

VI. Conclusions and Future Work

Reverse engineering of UML diagrams is utilized in various areas of software engineering. There are many applications of reverse engineering, but most of them are for structural UML diagrams. Behavioral UML diagrams can also be reversed and in our work we have proposed the methodology for reversing UML Use Case model from the data recorded during website usage. Our algorithm analyses recorded user activity and transforms it into UML Use Case and Activity diagrams. The prototype of the proposed algorithm was implemented as a Chrome extension. As this prototype is just the first of its kind, it generates use case and activity diagrams, but is not yet capable of detecting extend and include relations.

The results of experiment indicate that the implemented prototype is capable of generating UML Use Case model. Both Use Case and Activity diagrams were successfully generated using the prototype. The experiment results indicated that our method could be successfully utilized in practice. The experiment also provided valuable feedback about required improvements on action naming rules.

In the future, we are planning to implement the extended capability to support other UML modeling tools. The set of supported tools should include at least one open source tool that is free to use, so that the proposed method could be more accessible to wider audience.

References

[1] M. R. Chaudron, W. Heijstek and A. Nugroho, "How effective is UML modeling?," in *Software and Systems Modeling (SoSyM)*, 2012.

[2] E. Arisholm, L. C. Briand, S. E. Hove and Y. Labiche, "The impact of UML documentation on software maintenance: an experimental evaluation," *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 365-381, 2006.

[3] G. Rossi, Ó. Pastor, D. Schwabe and L. Olsina, Web Engineering: Modelling and Implementing Web Applications, Springer-Verlag London, 2008.

[4] E. J. Cross and J. H. Chikofsky, "Reverse engineering and design recovery: a taxonomy," in *IEEE Software, vol. 7, no. 1*,, 1990, pp. 13-17.

[5] J. Hibschman and H. Zhang, "Unravel: Rapid Web Application Reverse Engineering via Interaction Recording, Source Tracing, and Library Detection," in *UIST '15 Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, Daegu, Kyungpook, Republic of Korea, 2015.

[6] M. Mohammad I and E. A.-Q. Rafa, "An approach to derive the use case diagrams from an event table," in *8th WSEAS International Conference*, Cambridge, 2009.

[7] M. El-Attar and J. Miller, "Producing robust use case diagrams via reverse engineering," *Softw Syst Model*, pp. 7-67, 2008.

[8] M. I. Muhairat and A. Abdel, "A New Reverse Engineering Approach to Convert," *International Journal of Software Engineering & Applications (IJSEA)*, 2014.

[9] "UML 2.5 Specification," 1 03 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/PDF.

[10] E. Korshunova, M. P. M. v. d. Brand and M. Mousavi, "CPP2XMI: Reverse Engineering of UML Class, Sequence,," in *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06)*, 2006.

[11] "eclipse," The Eclipse Foundation, 2108. [Online]. Available: http://www.eclipse.org.

[12] "visual-paradigm," Visual Paradigm, 2018. [Online]. Available: https://www.visual-paradigm.com/.

[13] T. Ziadi, M. A. A. d. Silva, L. M. Hillah and M. Ziane, "A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams," in *16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, Las Vegas, United States, 2011.

[14] G. A. D. Lucca, A. R. Fasolino and P. Tramontana, "WARE: a tool for the Reverse Engineering of Web Applications," *Journal of Software Maintenance and Evolution: Research and Practice - Special issue: Web site evolution*, pp. 71-101, 2004.

[15] Richard Soley and the OMG Staff Strategy Group, *Model Driven Architecture*, 2000.

[16] B.A. Nowak, R.K. Nowicki, M. Woźniak, and C. Napoli,. "Multi-class nearest neighbour classifier for incomplete data handling," in *International Conference on Artificial Intelligence and Soft Computing*, pp. 469-480, 2015.