Applied Computer Systems
ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
June 2024, vol. 29, no. 1, pp. 59–67
https://doi.org/10.2478/acss-2024-0008
https://content.sciendo.com

Definition of a Set of Use Case Patterns for Application Systems: A Prototype-Supported Development Approach

Oksana Ņikiforova[1], Kristaps Babris[2*], Aytaj Guliyeva[3]
[1–2]*Riga Technical University, Riga, Latvia*
[3]*Accenture Baltic, Riga, Latvia*

***Abstract* – UML diagrams are a base for the planning of development in most software projects. It is used for representing different artefacts during software development and project structure. The use case is one of the diagrams in Unified Modelling Language (UML), which allows describing the dynamic flow of the system. There are a lot of tools that are used for creating this diagram before starting the actual coding process, and the diagram needs to be specific and easily understandable. Meantime, the creation of a UML use case diagram from scratch for complex systems can be time-consuming and confusing for people, which needs to be optimised. The authors of the paper attempt to solve the addressed problem. Therefore, in this research paper a new definition for UML use case diagrams will be introduced, where the main question will be whether it is possible to formalise use case modelling by introducing pre-defined use case patterns. This is academic research and discussion, which is based on the analysis of advanced UML tools, which use case diagram templates contain. The solution to this research question contains an initial set of UML use case patterns, created by analysing of the existing use case diagram templates. Moreover, in order to validate work, the pre-defined patterns were demonstrated on a developed prototype. The operation principle of the prototype focused on giving the ability to the user to construct a use case diagram by the combination of pre-defined patterns. The prototype can be useful for the development/management process in case of correct implementation. It will allow decreasing spent time on the use case diagram creation as well as avoid creating anti-patterns.**

***Keywords* – Pre-defined patterns, system modelling, UML use case diagram, UML modelling tools.**

I. Introduction

Software engineering has seen much advancement over the years, focusing on making better the process of software development and raising standards for software systems [1]. In this progress, requirement analysis is a significant stage where programmers team up with interested parties to collect and polish functional needs. Use case diagrams are one of the most important elements in Unified Modelling Language (UML) [2], providing a strong way to show these functional requirements. They make clear connection between users and system that is being developed.

However, the UML's use case patterns are not used effectively. This poses a difficulty for developers and also makes it harder to communicate with those who are interested in the project. The problem is that there is not any clear way to define use case diagrams. That causes confusion, misinterpretation and slows development processes. This paper focuses on recognising problems related to defining use cases within UML and suggesting the solution to base use case diagram development on the pre-defined patterns that can be included into an enhanced design model for real-life application in software development.

The research performed by Guliyeva [3] has identified several problems in the definition of use case patterns in previous studies:

- *Absence of Standardization*: As there are no universally recognised standards and rules to define UML use cases, the use cases can be understood in different ways. This gives parties involved a chance to consistently comprehend and apply them;
- *Pattern Definition*: The case patterns of use do not have exact explanations, which causes communication issues and confusing situations impacting the cycle for software development;
- *Inconsistency*: When developers do not use case patterns consistently, it makes it difficult to combine development processes and tools – this also leads to more standardization issues.

Additionally, not paying enough attention to details in UML diagrams can make these problems worse. This might put project quality and effectiveness at risk. Even though using documented templates could help reduce mistakes made manually, it is not an easily expandable fix because of each project unique needs that require customisation.

Use case diagrams are very important in software projects because they help define and comprehend the system requirements. They offer a summary of the structure of system at high level, explaining how actors interact with software and

* Corresponding author's e-mail: kristaps.babris@rtu.lv
Article received 2024-04-12; accepted 2024-07-22

providing points for reference during development as well as testing phases.

This is academic research, which is based on the analysis of advanced UML tools, which use case diagram templates contain. The goal of the paper is to offer a set of use case patterns for the most common application cases, making it possible for users to choose the particular fragment of ready use case diagram from the list of pre-defined patterns when creating or refining the use case diagrams. This paper is organised in such a way that it overviews the theoretical background, goes through existing modelling tools and methods summarising the use case diagram templates suggested. It also offers a set of pre-defined UML patterns, shows the supporting prototypes, and finally examines modelled outcomes using test situations. To solve the problems and supply reusable design patterns, this study aims at providing consistency, clarity and ease of use in UML-based software making. It aspires to enhance efficiency and error forgiveness in systems that are developed via UML.

II. Background and Related Work

In the business setting of today, big enterprise systems are basic for running businesses. These systems do many tasks to keep operations going smoothly. This makes it very important that the structure of these systems is clear, matching with organisation's goals. At present, it is very important for companies to create a system model before starting development. The drawing representation of the system makes it easy for people to examine possible problems and differences. In software projects, such models help teams understand the system well and make good plans.

In the words of Object Modelling Group [4], modelling is about designing a system before it gets developed. It is a method that plays an important role in making sure software development projects are successful. When we find and fix issues or shortcomings in the model, changes can be made more effectively and economically. This saves time and resources over the entire duration of project completion.

The first cause for the need of software modelling languages is proper documentation. People who create software frequently face difficulties because there is not a uniform language, and designs or specifications are complex to understand. UML helps solve this issue as it offers an exact and easy-to-understand instrument for modelling software. The idea is about changing text explanations into a sequence of models, making the design more visual. Another benefit that UML provides is the capability for automatic creation of source code or reverse engineering, which makes switching between code and models simpler.

In general, UML is a modelling notation for representing complex systems visually. It provides an all-inclusive structure and model of software systems to effectively handle project demands [4]. As detailed in [5], UML operates as a language for building the blueprint of software systems. The main uses of UML are documentation, visualisation and specification, as well as construction of systems. UML diagrams, in general, can be grouped and they have 14 types [2]. This study is about UML use case diagram, which falls under the category of behavioural diagrams within UML framework. These types of diagrams show the main functions of a system by illustrating actors and use cases with their relationships [6]. UML use case diagrams are well-known in the world of diagrams and hold the third position for being used most often among publications checked for study [7]. Thus, use case diagrams are still being demanded, indicating their lasting importance as a well-liked research subject, which needs more investigation. The significance of use case diagrams as both a research domain and an important instrument in software projects becomes clear, emphasising the need for optimisation, especially regarding the production of complex system diagrams with less effort required.

Modelling, a crucial part of designing software, helps in understanding and describing the needs of a system. This often occurs during planning activities. Therefore, it is very important to make use case diagram fit for this purpose. In general, there are four basic principles for modelling [8]:

1. The models chosen greatly impact how problems are tackled and solutions are made. If a model is more precise, there should be fewer issues during the process of development;
2. The precision of each model can be different, so the level of detail in a use case diagram should match with how complex you want it to be;
3. The best models mirror real-world circumstances, enhancing understanding and applicability;
4. One model for complex systems is not enough. A small collection of almost separate models should be used, with each focusing on a distinct part of the system.

Considering recent occurrences and the application of many use cases, we observe that there are repetitive patterns in use case situations. This presents chances to simplify the modelling procedure. As a result, typical use cases can be used as templates that bring advantages in development stages by lessening repetition, guaranteeing uniformity and saving time and effort across projects.

The UML use case diagram is a mainstay in many IT projects, providing clarity about project needs and maintaining precision of these requirements throughout the life cycle of the project. The study has emphasised its importance from requirement gathering to testing phase [9]. Even though it is very important, there are no strict rules for making UML use case diagrams, which makes it hard to apply in practice. Although UML diagrams were created in 1986, the requirement for simple directions continues to exist [10].

Different ways have come up to handle this problem. For example, Schneider and Winters [11] provide a guide for making diagrams using real-life examples. Then there are creative methods like CASL-based methods that use algebraic structures to describe elements in a diagram [12]. Also, the research contributions of Alberto et al. [13] and Faitelson and Tyszberowicz [14] give ideas and regulations for making good use case diagrams.

Once they understood that use case diagrams had common functionalities across projects, researchers began to concentrate on finding likenesses to make their reuse easier. For example, studies [15] and [16] both apply semantic analysis along with

structural analysis for recognising similarities. These endeavours highlight how patterns are starting to emerge in use case diagrams over time, setting up the possibility for pattern creation.

In addition, studies have also investigated the utilisation of CRUD operations as a foundation to establish use case patterns. For example, studies [17] and [18] both look into CRUD operations and relationships to define patterns, while suggesting semi-automatic answers for scenario creation. In the recent work [19], they make use of Formal Concept Analysis to find similar use cases, adding more to the list of patterns that can be used again in different situations.

Lionel [20] brings out the importance of meta-level management to attain effective reuse, highlighting the significance in identifying and handling use case types. By using design patterns and specific UML models, developers can gather different use cases together in a smart way that allows them to make use of pre-existing implementations. Models represented by UML provide a concise view of interactions between use cases, helping with understanding and finding overlaps within the system.

However, there is a problem with the elaboration of UML model architecture. Ballis [21] emphasises the requirement for strong means to compare patterns of software design with UML models, making better pattern use during design possible.

This paper, by using knowledge gained from past studies, tries to offer fresh thoughts and descriptions of UML use case diagrams while finding usual models. Its goal is to improve understanding about the application of use case modelling in creating software systems.

To sum up, the UML use case diagram uses certain notations that help developers and interested parties to comprehend processes. Past research has thoroughly investigated UML diagrams, how they are put into action, and why design patterns are so important. The focus of this paper is on finding similarities in projects that use software tools and techniques. These studies reveal the need for fresh methods in software development, highlighting the importance of design patterns and the deficiencies of current approaches.

III. Systematisation of Use Case Diagram Fragments in the Popular UML Modelling Tools

In this section, we summarise the most popular tools for UML modelling and templates for use case diagrams that are offered in these tools. The most commonly used UML use case diagram tools are as follows [22]:

1. Draw.IO [23] – an open-source tool, which allows users to create UML diagrams. This tool has also mentioned a template, which can be used as an example for a use case diagram;
2. Lucidchart [24] – offers an option to make diagrams, similar to charts. It also contains several templates for different types of diagrams like UML use case diagram among others;
3. Visual Paradigm [25] – a tool for making UML diagrams that includes many types of diagrams, charts, business process models and more. One of the services it offers is creating use case diagrams;
4. Edraw Max [26] – a tool for modelling of different types of diagrams, charts, models, etc. The most noticeable part about this tool is that it has many templates compared to others we considered;
5. Gliffy [27] – a tool for modelling of various kinds of charts, models and diagrams;
6. Cacoo [28] – a diagramming tool for creation of various diagrams and models, which provides only one UML use case diagram for users.

The total number of templates offered by UML use case diagram modelling tools mentioned above is 120 use case diagrams that range in complexity levels and scenarios. All these templates were analysed in terms of their content and structure of the use case diagram offered and summarised based on their similarities. An example of such an analysis of the template of EdrawMax tool offered in [29] is shown in Fig. 1.

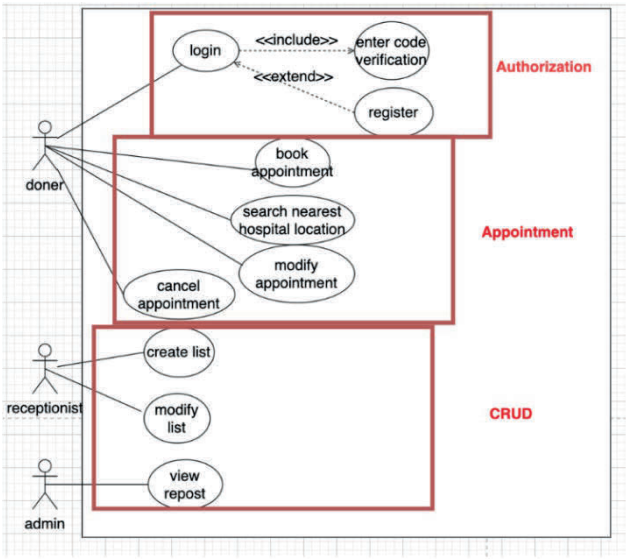

Fig. 1. Division of the use case template into smaller parts suggested for use case patterns.

The template was analysed and classified into parts – potential repetitive functionalities, where it is observed that use case diagram is divided into authorisation, appointment and basic CRUD sub-templates. In the given context, CRUD stands for a simple operation combination with data such as create, modify, view and delete, which comes from technical point of view.

The same analysis was performed for all the templates offered in the most used UML modelling tools and, as a result, the set of eight patterns that is functionally grouped into three classifications is described in the next section.

IV. Patterns of Use Case Diagram for the Most Common Business Applications

In general, the pattern is a reusable part during project development. If we consider that most of the software projects include common reusable sections, then patterns play an important role here. Thus, the patterns help in making the

software development process more efficient and repeatable. The main types of the patterns are as follows:

1. Architectural patterns give details about relationships among components and describe a set of architectural characteristics [30]. Also, there are multiple types in architectural pattern such as Layered, Event-driven, Microkernel and Microservices, Client-Server, etc.;
2. Concurrency patterns are based on multithread and parallel programming [31]. There are numerous kinds of concurrency patterns such as double-checked locking, thread pools, active objects etc.;
3. Design patterns in the context of software development are applied in various phases and paradigms of software development for design of reusable programming components within the best programming practices [32].

During the modelling process, it is important to bring out a wider variety of patterns that are kept for use cases. We do this, so we can think about particular functional needs from user's perspective. Use case patterns, in general, are the most suitable method to provide a more organised and specialised manner of showing the interrelations between actors and the generalised system. Specified patterns can act as the main guidelines or templates, which help in implementing use case diagrams while enhancing consistency across different projects.

The authors defined the eight patterns for the most used application system use cases, where the following list shows the number of the corresponding use case templates offered in the modelling tools, which are 120 use case diagrams in total, storing the most repeated patterns and removing other items from the list:

1. authorisation (45 usages);
2. appointment (15 usages);
3. order (20 usages);
4. bank (16 usages);
5. delivery (12 usages);
6. payment (31 usages);
7. feedback (13 usages);
8. data management (30 usages).

As an example of how the patterns were created, this paper shows in details the definition of the pattern "authorization", which usually has registration/login use cases, and is extensively employed in most projects. The templates offered in the modelling tools are examined in terms of their similarities with the conclusion that authorisation part differs from each other mainly due to its structure. Then, the analysis of the template differences was made according to the authorisation flow. For expressing and examining templates, we have categorised the same authorisation processes into groups based on use case diagram structures shown in Fig. 2.

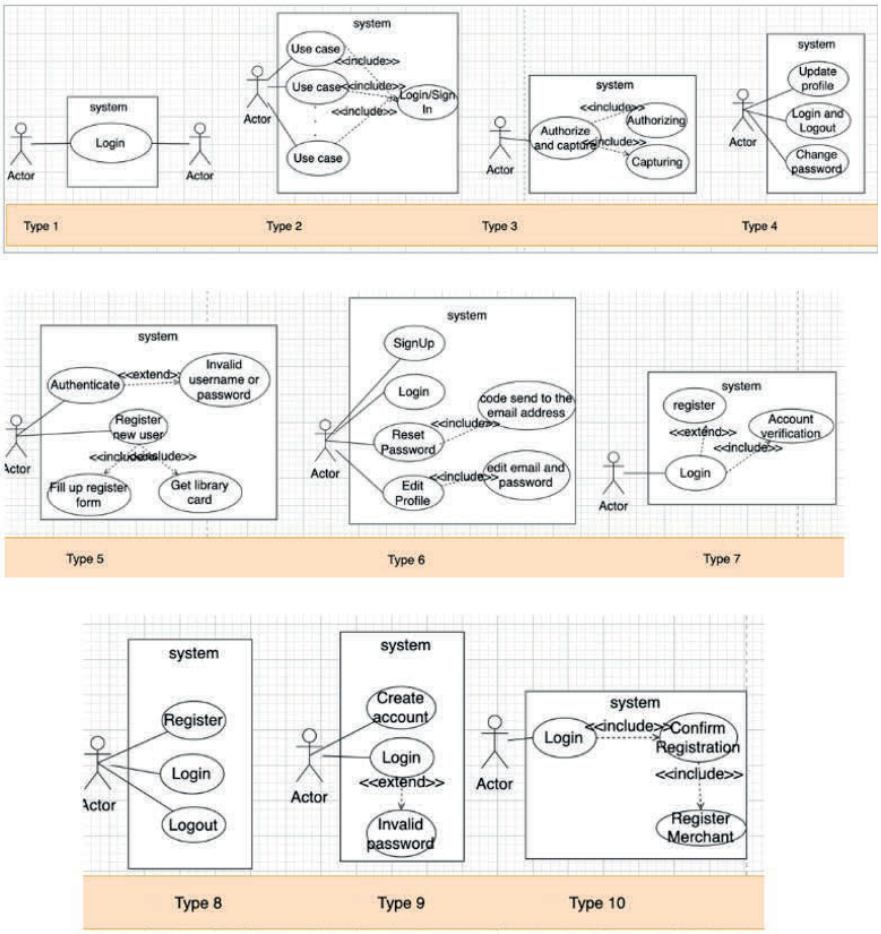

Fig. 2. Template examples for the authorisation use case.

In general, based on the given use case diagrams for authorisation, we can understand that the patterns planned to be used must have a quality of being reusable and easily adjustable for certain needs of a project. Furthermore, instances indicate that insufficient details may not adequately cover important parts within an illustration, while excessive detailing can cause complexities in a diagram. The different types of examples show that the most frequently used use cases for the representation of authorisation diagram are a combination of: Sign Up, Login, Reset Password, Update profile, and Logout. The use cases in the pattern were repeated in various examples. The authorization use case pattern can be represented as shown in Fig. 3.

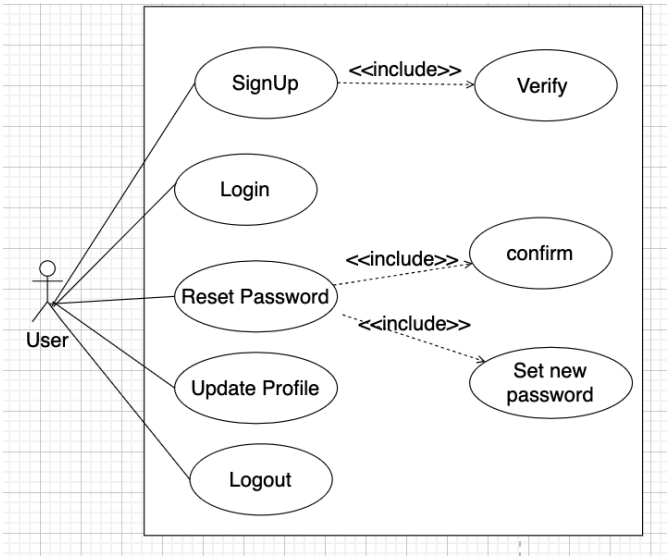

Fig. 3. The use case pattern offered for “authorization”.

The outcome of the analysis has led to the suggested authorisation pattern, which consists of five main use cases and three additional ones. Considering that in different templates and specific projects, the main actor is called by various names, we suggest the common naming “user” that will refer to any person who uses the system. The initial diagram of the proposed pattern says that a user can Sign Up into system; it includes verification for an account plus Login, too. If the user forgets their password, they can reset it by confirming the account and creating a new password. Also, this pattern involves updating of the profile and as a final use case: logout from the system.

The repeated fragments of various use case diagrams serve as a basis to define a set of the patterns for the most common application cases used in the information systems. Each template has its structure and specification. This is because use cases are unique to the system and defined requirements of its own users. The practice to improve and make more effective use case models is by analysing existing templates, which provide useful information along with ready-made frameworks that can be altered for a certain project’s needs.

In general, these patterns can be grouped together in a logical way. For many projects that have been researched, the users begin with authorisation and then proceed to the main functionality of a specific project. So far, the research suggests to categorise use case patterns into the following groups:

1. Initial patterns. For the first flow of software projects, we start from the registration/login flow to the system. Here, actions can be taken. Initial patterns – at first, software projects begin with a login/registration flow to access the system where more actions can be performed;
2. Business logic patterns. They will be the most crucial part of the project after authorisation, where the main idea/services of the system are described;
3. Closure patterns. They will include the last steps for the idea, for example, if we take an e-commerce project, then delivery or payment pattern would be the last step for service.

V. Prototype to Support the Idea of Use Case Pattern Usage

The idea of implementation of the solution offered in this paper is to make a builder that includes pre-defined pieces/fragments of the already ready use case diagram parts, which can be utilised as parts for larger software system’s UML use case diagram. A prototype version of this solution provides a system that have ready UML use case patterns and acts as a constructor of the fragments of use case diagram for the system.

The primary requirements for the prototype that has been provided:

1. Users can select several pre-defined UML use case patterns;
2. Users can edit pre-defined patterns;
3. Users can add additional elements on a workspace;
4. Users can save UML use case diagrams;
5. Users can download UML use case diagrams in XML format.

The wireframe for the solution prototype is shown in Fig. 4. On the left side, the proposed set of use case patterns is offered as a tool bar to drag-and-drop them on the work area. It is offered to have an ability to draw additional elements of the use case diagram if needed. The tool bar with the use case diagram elements is offered on the top of the user interface form. Therefore, the proposed pattern combination method has a few rules, where pattern usage by user will have two choices: to select pattern and link it to a particular element. One might see this as a simple multi-step activity created for making many different hard tasks and fresh thoughts easy to understand.

The system side bar gives full help once the user starts it by showing a picture of the specified patterns. Additional elements can be considered parts of these patterns, from design templates to building plans. In the end, our aim is for users to choose a pattern which is compatible and shows all specialised demands for their ongoing project. One thing that should be noted is the full method used for selecting a pattern and linking it to particular workspace, so users can effectively use pre-designed patterns and show the relationships among them.

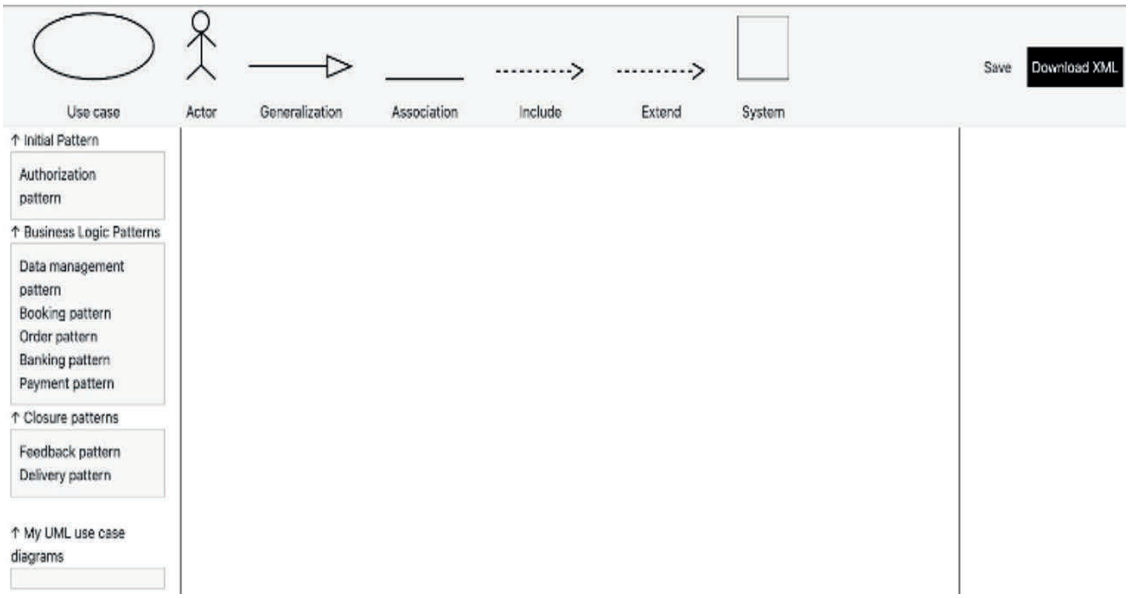

Fig. 4. Solution prototype wireframe.

VI. Practical Example and Validation

This section presents a picture of how the working process of a prototype can be applied in practice, used to back up the suggested research concept. To test the outcomes, we will make an instance illustration, which will include use case description for the shopping system. This will consist of commonly used features such as registration, placing orders, paying and getting items delivered. The use case diagram has three actors: Customer, Payment Gateway, Courier and a system itself that is identified by boundaries. The application case consists of the two ways of modelling the shopping system: creating a model through a standard way (see the left side of Fig. 5) and making it by applying pre-defined patterns of the developed prototype (see the right part of Fig. 5). The diagram created by the prototype combines patterns without any changes. In the case of the shopping problem domain, we used authorisation, order, delivery and payment patterns. The pattern of payment was linked to the "confirm order" use case, while other three were linked with their actors.

Emphasising the importance of evaluation process in the implementation is crucial because it acts as a method to check quality for an entire design procedure. Evaluation process provides a direct way to perform a broad analysis about how efficient and validly successful given patterns are in a specified test case. If we do not evaluate it properly, we cannot measure if the designed patterns really met the defined requirements and accomplished planned objectives.

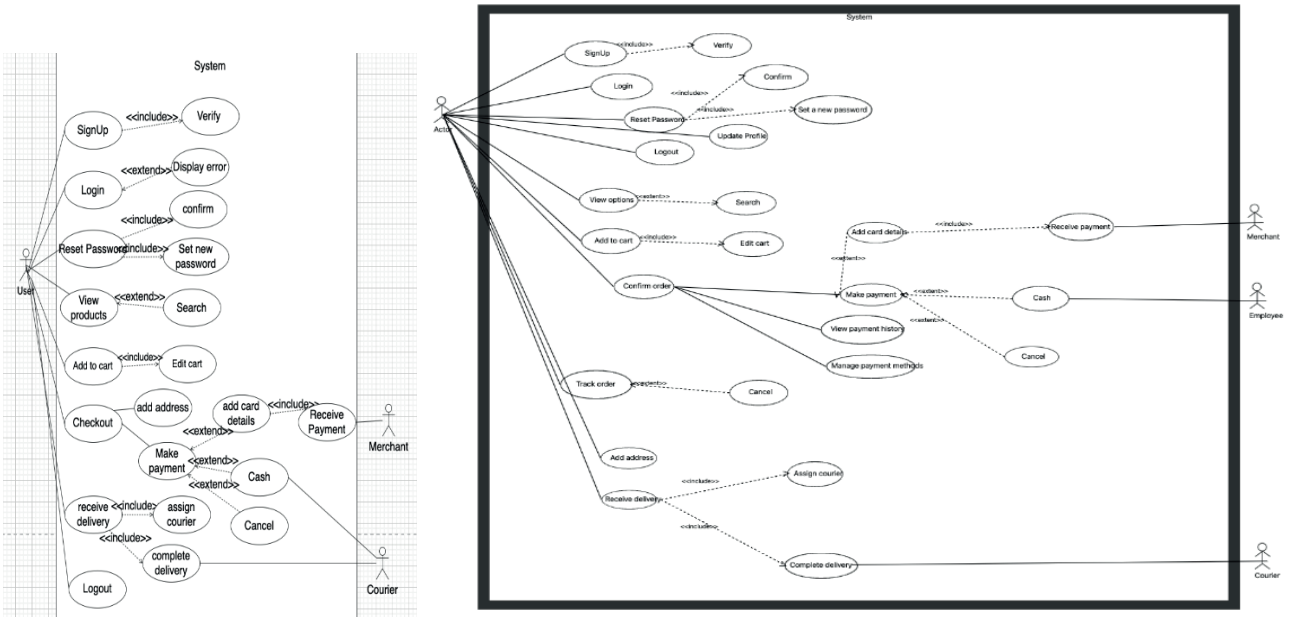

Fig. 5. Two use case diagrams created for shopping problem domain (a use case diagram on the left side is created manually, a use case diagram on the right side is created with patterns offered in the solution prototype).

Additionally, having a thorough evaluation makes it possible to achieve a significant level of the implementation. This can help in identifying the advantages and disadvantages of the suggested patterns, serving as an important hint for future designs. On the other hand, this process also offers a method to confirm how well-chosen patterns work in general and their adaptability in specific conditions of the test case.

To identify the weak and strong aspects of an idea about pre-defined patterns, we need to set up some specific elements. These are the core points that should be considered during the validation process:

1. Coverage. This is about the coverage of use cases in specified patterns. It shows how many required use cases are already included in the patterns, based on the requirement specification;
2. Structuring – stresses the accuracy of the diagram in putting use cases and displaying detailed action sequences;
3. Consumption of time. This shows how much time is used to create the model compared with the description;
4. Completeness. This talks about how well the likely user stories of system were taken care of. There might be missing parts in some use case definitions that make the structure of the system model incomplete.

In case of coverage for the given test scenario, it is highlighted that a standard modelling approach and model constructed using pre-defined patterns have 20 use cases. If we perform an efficiency analysis for a specific suggested design (see Table I), there are 21 use cases with the same purpose as stated in combined patterns, whereas locating 22 use cases in the standard way of doing things. "Display error" is the only one of those instances which could not be found within patterns.

TABLE I
EVALUATION RESULTS

Metric	Standard approach	Pattern combination method (use case)
Coverage	22	21
Structuring	-	2
Time consumption	1×	2.5×
Logical completeness	-	5

The completion level of this scenario might be different (see Table I). While they were not enough in other patterns, the permission pattern did not have a use case. After doing detailed research, we found out that the authorisation pattern includes updating user's profile because it is commonly seen in systems where users have profiles and can change their personal information. This feature may have gone unnoticed in the present description.

The expected goals that align are the test validation and performance analysis. A determined test case is made to ensure results are objective, repeatable and enable relevant result making for viability and effectiveness of an idea.

In order to carry out the successful validation procedure, an example scenario is made. This directly leads to using the model in a usual way and creating the same structure in prototype by using patterns that have been defined.

Additionally, according to [3], "The progression procedure can be recognized through four vital key points. Every single point clarifies what the core stages of a process are and it must include these to have successful results. With the consequences that come out, we can see how efficient patterns truly are – for instance with our example there's just one use case missing but all else present." However, it is worth noting that the combined patterns are not precise and in order. If we consider from the speed point of view, it was better and could be a useful stage to improve this feature. Additionally, for the completeness issue results are such that following a standard way can cause missing some use cases during the process.

VII. FURTHER PRACTICAL IMPLICATIONS OF THE USE CASE PATTERNS

The issue left in the solution prototype is the layout of the use case diagram. The authors previously already touched solving this problem and had valuable results on the layout of the use case diagram in [33], where the authors discussed the necessity of use case diagrams for the high-level understanding of system functionality and the automatic layout of UML use case diagrams. Based on the existing studies, the authors collected a set of guidelines which have been divided into two levels: micro and macro levels. In total, the authors defined more than 20 guidelines for use case diagram layout, which should be followed while creating use case diagrams to make them easy to understand. However, the automatic use case diagram layout algorithm, which would support the above-mentioned guidelines, is still under development.

Another thing we would like to discuss is creating user interface patterns for the use case patterns offered in this paper for the most common information system application cases. The experiments made on user interface generation for particular application domains within patterns like CRUD in [34] give a potential to refine the solution for use case development from the pre-defined patterns with the ability to generate pre-defined user interface wireframes for the particular use case fragments and further code generation from them [35]. According to this research, parallels can be drawn between use case and user interface templates. Researchers and user interface designers describe templates in a specific format, which may vary. Another study [36] gives examples of how to define user interface templates. Although several canonical forms [37], [38] have been established throughout the history of user interface templates, [39] they are not standardized and there is no single format for defining user interface templates.

For example, the user interface template appropriate for the same use case pattern "authorisation", which is demonstrated as an example in Section IV, is shown in Fig. 6.

Moreover, the principles of the model-driven software engineering [40] can be applied in both modelling the use case diagrams within the corresponding use case patterns as a source model and the particular user interface element generation. The solution for automatic code generation for the components of Web applications can be based on the modelling of the problem domain expressed in terms of business processes, then generation of the required use case diagram in accordance with

the use case patterns and the corresponding transformation rules, then automatic generation of the user interface wireframes in accordance with the pre-defined templates [34] and, finally, the transformation of these wireframes to the expected front-end components of Web applications [35].

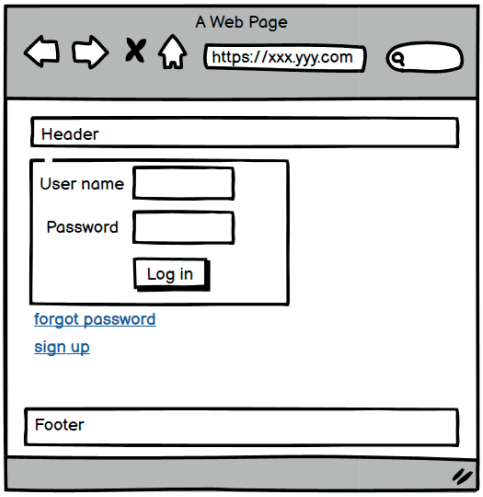

Fig. 6. User interface template for the use case pattern "authorization".

VIII. Conclusions and Future Research

This research is about the development of a set with pre-defined patterns for the UML use case diagrams that can be used in different areas of software project. From literature review, the authors found that projects had repetitive parts in modelling over time, and many related studies tried to find similarities and make patterns for reuse. Therefore, the authors proposed creating ready-made patterns by examining several examples given by the modelling tools.

The main results are the following.

1. The theoretical background, related studies, and existing tools were examined;
2. The patterns were proposed based on the analysed examples from applications;
3. Eight patterns were suggested and they were grouped into three classifications by functionality. The test case result evaluation showed that this concept was effective in terms of time usage, completeness, and effectiveness;
4. A prototype was developed that had default inserted patterns and a functionality to combine them;
5. The model that was made is able to be exported in XML format. It can also be imported into the draw.io.

Basically, the UML use case diagram is an important part of creating software. It can help with complex projects that have similar common use cases to make patterns for different systems and projects. One must remember that this diagram is not merely a set guide but a progressing strong instrument. It improves the design abilities of software creators, assisting them in carrying out projects more precisely and strongly as well. To sum up, the idea that has been proposed would give fresh value to information systems modelling field by allowing for the production of diagrams, which address most of functionality but not all. Moreover, all the shown outcomes demonstrate what this design can do by boosting system efficiency and giving several benefits. This study also acts as strong backing for more thoughts about software development projects that take into consideration UML design patterns.

References

[1] N. Ford and M. Richards, *Fundamentals of Software Architecture*. O'Reilly Media, Inc., 2020.

[2] UML Specification [Online]. Available: https://www.omg.org/spec/UML/2.5.1/About-UML. Accessed on: Apr. 11, 2024.

[3] A.Guliyeva. "Definition of patterns for UML use cases," M.S. thesis, Riga Technical University, 2023.

[4] OMG, "What is UML". [Online]. Available: https://www.uml.org/what-is-uml.htm. Accessed on: Apr. 11, 2024.

[5] I. Jacobson, G. Booch, and J. Rumbaugh, *Unified Modeling Language User Guide*. Addison Wesley, 1998.

[6] K. Rungta, *UML 2.0: Learn UML in 1 Day*, Independently published, 2019.

[7] H. Koç, A. M. Erdoğan, Y. Barjakly, and S. Peker, "UML diagrams in software engineering research: A systematic literature review," *Proceedings*, vol. 74, no. 1, Mar. 2021, Art. no. 13. https://doi.org/10.3390/proceedings2021074013

[8] The University of Texas at Austin, "Introduction to object oriented programming." [Online]. Available: https://www.cs.utexas.edu/users/mitra/csSpring2017/cs303/lectures/oop.html. Accessed on: Apr. 11, 2024.

[9] A. M. Qazi, A. Rauf, and N. M. Minhas, "A systematic review of use cases based software testing techniques," *International Journal of Software Engineering and its Applications*, vol. 10, no. 11, pp. 337–360, 2016.

[10] CPlusOOP, "Origin of use cases." [Online]. Available: https://www.cplusoop.com/uml/module3/use-case-model.php. Accessed on: Jul. 14, 2024.

[11] G. Schneider and J. P. Winters, *Applying Use Cases: A Practical Guide*. Addison-Wesley Professional, 2001

[12] B. Mondal, B. Das, and P. Benerjee, "Formal specification of UML use case diagram – A CASL based approach," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 2713–2717, 2014. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=dcd8b0e7adc51256bcfcc584cff3c2d807fac507

[13] A. Da Silva, S. Dusan, V. Sinisa, A. Ilija, L. Sasa, S. Vojislav, and M. Milos, "Patterns for better use cases specification," in *Proceedings of EuroPLOP'2015*, Jul. 2015, pp. 1–18. https://doi.org/10.1145/2855321.2855330

[14] D. Faitelson and S. Tyszberowicz, "UML diagram refinement (focusing on class-and use case diagrams)," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, Buenos Aires, Argentina, May 2017, pp. 735–745. https://doi.org/10.1109/ICSE.2017.73

[15] M. Arifin and D. Siahaan, "Structural and semantic similarity measurement of UML use case diagram," *Lontar Komputer Jurnal Ilmiah Teknologi Informasi*, vol. 11, no. 2, 2020, Art. no. 88.

[16] R. Fauzan, D. Siahaan, S. Rochimah, and E. Triandini, "Use case diagram similarity measurement: A new approach," in *2019 12th International Conference on Information & Communication Technology and System (ICTS)*, Surabaya, Indonesia, Jul. 2019, pp. 3–7. https://doi.org/10.1109/ICTS.2019.8850978

[17] A. M. R. Cruz, "A pattern language for use case modeling," in *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, Lisbon, Portugal, Jan. 2014. [Online]. Available: https://www.researchgate.net/publication/259737759_A_Pattern_Language_for_Use_Case_Modeling

[18] M. Ochodek, K. Koronowski, A. Matysiak, P. Miklosik, and S. Kopczyńska, "Sketching use-case scenarios based on use-case goals and patterns," in *Software Engineering: Challenges and Solutions*, L. Madeyski, M. Śmiałek, B. Hnatkowska, and Z. Huzar, Eds. Springer, 2016, pp. 17–30. https://doi.org/10.1007/978-3-319-43606-7_2

[19] R. Al-Msie'Deen, A. H. Blasi, M. Alsuwaiket, A. Alabadleh, S. Aljaafreh, W. Tarawneh, and S. Al-Showarah, "Detecting commonality and variability in use-case diagram variants," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 04, pp. 1113–1126, Feb. 2022. https://doi.org/10.48550/arXiv.2203.00312

[20] L. C. Briand, Y. Labiche, and A. Sauve, "Guiding the application of design patterns based on UML models," in *2006 22nd IEEE International Conference on Software Maintenance*, Philadelphia, PA, USA, Sep. 2006, pp. 234–243. https://doi.org/10.1109/ICSM.2006.30
[21] D. Ballis, A. Baruzzo, and M. Comini, "A rule-based method to match software patterns against UML models," *Electronic Notes in Theoretical Computer Science*, vol. 219, pp. 51–66, Nov. 2008. https://doi.org/10.1016/j.entcs.2008.10.034
[22] GeeksforGeeks, "Top 7 UML diagram tools that you can consider." [Online]. Available: https://www.geeksforgeeks.org/top-7-uml-diagram-tools-that-you-can-consider. Accessed on: Apr. 11, 2024.
[23] JGraph Ltd, "draw.io." [Online]. Available: https://www.drawio.com/. Accessed on: Apr. 11, 2024.
[24] Lucid Software Inc, "Lucidchart." [Online]. Available: https://www.lucidchart.com/pages/landing. Accessed on: Apr. 11, 2024.
[25] Visual Paradigm, "The four types of relationship in use case diagram." [Online]. Available: https://blog.visual-paradigm.com/the-four-types-of-relationship-in-use-case-diagram/. Accessed on: Apr. 11, 2024.
[26] Wondershare, "EdrawMax." [Online]. Available: https://www.edrawsoft.com/ Accessed on: Apr. 11, 2024.
[27] Perforce Software, Inc, "Gliffy." [Online]. Available: https://www.gliffy.com. Accessed on: Apr. 11, 2024.
[28] Nulab, Inc., "cacoo." [Online]. Available: https://nulab.com/cacoo. Accessed on: Apr. 11, 2024.
[29] EdarwMax, "Life line system use case." [Online]. Available: https://www.edrawmax.com/templates/1010094. Accessed on: Apr. 11, 2024.
[30] G. Me, G. Procaccianti, and P. Lago, "Challenges on the relationship between architectural patterns and quality attributes," in *2017 IEEE International Conference on Software Architecture (ICSA)*, Gothenburg, Sweden, Apr. 2017, pp. 141–144. https://doi.org/10.1109/ICSA.2017.19
[31] A. S. Khot, *Concurrent Patterns and Best Practices*. Packt, 2018.
[32] A. Shvets, *Dive into Design Patterns*. Refactoring Guru, 2021.
[33] O. Filipova and O. Nikiforova, "Definition of the criteria for layout of the UML use case diagrams," *Applied Computer Systems*, vol. 24, no. 1, pp. 75–81, May 2019. https://doi.org/10.2478/acss-2019-0010
[34] K. Babris and O. Nikiforova, "Towards automated UI Mockup generation from two-hemisphere problem domain models: A conceptual framework and approach," in *Proceedings of Iberian Conference on Information Systems and Technologies*, 2024 (in press).
[35] O. Nikiforova, K. Babris, and F. Mahmoudifar, "Automated generation of Web application front-end components from user interface Mockups," in *Proceedings of International Conference on Software Technologies*, vol. 1, Dijon, France, 2024, pp. 100–111. https://doi.org/10.5220/0012759500003753
[36] M. Welie, G. Veer, and A. Elins, "Patterns as tools for user interface design," in *Tools for Working with Guidelines*, J. Vanderdonckt and C. Farenc, Eds. Springer, London, 2000, pp. 313–324. https://doi.org/10.1007/978-1-4471-0279-3_30
[37] C. Kruschitz and M. Hitz, "Human-computer interaction design patterns: Structure, methods, and tools," *International Journal on Advances in Software*, vol. 3, no. 1–2, pp. 225–237, 2010. [Online]. Available: https://www.academia.edu/10320340/Human_Computer_Interaction_Design_Patterns_Structure_Methods_and_Tools
[38] S. Suleri, N. Kipi, L. C. Tran, and M. Jarke, "UI design pattern-driven rapid prototyping for agile development of mobile applications," in *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI'19)*, New York, NY, USA, Oct. 2019, Art. no. 52. https://doi.org/10.1145/3338286.3344399
[39] L. F. da Silva, P. A. Parreira Jr., and A. P. Freire, "Mobile user interaction design patterns: A systematic mapping study," *Information*, vol. 13, no. 5, May 2022, Art. no. 236. https://doi.org/10.3390/info13050236
[40] M. Brambilla, J. Cabot, and M. Wimmer, *A Model-Driven Software Engineering in Practice*, 1st ed. Morgan & Claypool, 2013.

Oksana Ņikiforova received the Doctoral degree in Information Technologies (system analysis, modelling and design) from Riga Technical University, Latvia, in 2001. She is presently a Professor at the Institute of Information Technology of Riga Technical University. She is an author of more than 100 publications on model-driven software development, object-oriented system analysis, modelling, design and development, and software development project management. Her recent publications are related to user experience analysis, work efficiency estimation and user behaviour model-driven insider's security threat identification. She has managed and participated in about 50 research and development projects. Her current research interests include model driven software development and agile software development project management methods and tools.
E-mail: oksana.nikiforova@rtu.lv
ORCID iD: https://orcid.org/0000-0001-7983-3088

Kristaps Babris received the Master degree in Computer Systems from Riga Technical University, Latvia, in 2018. He is presently a fourth-year PhD student at the Faculty of Computer Science, Information Technology and Energy of Riga Technical University. He is an author of several publications about model-driven engineering principles in the analysis of information systems usability and efficiency, and automatic generation of user interface prototype and front-end components for web applications. In parallel, he is a CTO at the company, which develops business intelligence solutions. He participates in research and development projects, where he works on innovative solution evolution. His current research interests include design and modelling.
E-mail: kristaps.babris@rtu.lv
ORCID iD: https://orcid.org/0000-0003-3855-6963

Aytaj Guliyeva received the Master degree in Computer Systems from Riga Technical University, Latvia, in 2023. She is presently a Software Developer at Accenture Baltics with several years of experience in the IT industry. She has specialised in IT project management and software development. Her current research interests include user interface design and development within advanced programming frameworks, where she combines creativity with technical expertise to push the boundaries of technology.
E-mail: guliyeva.aytaj6@gmail.com