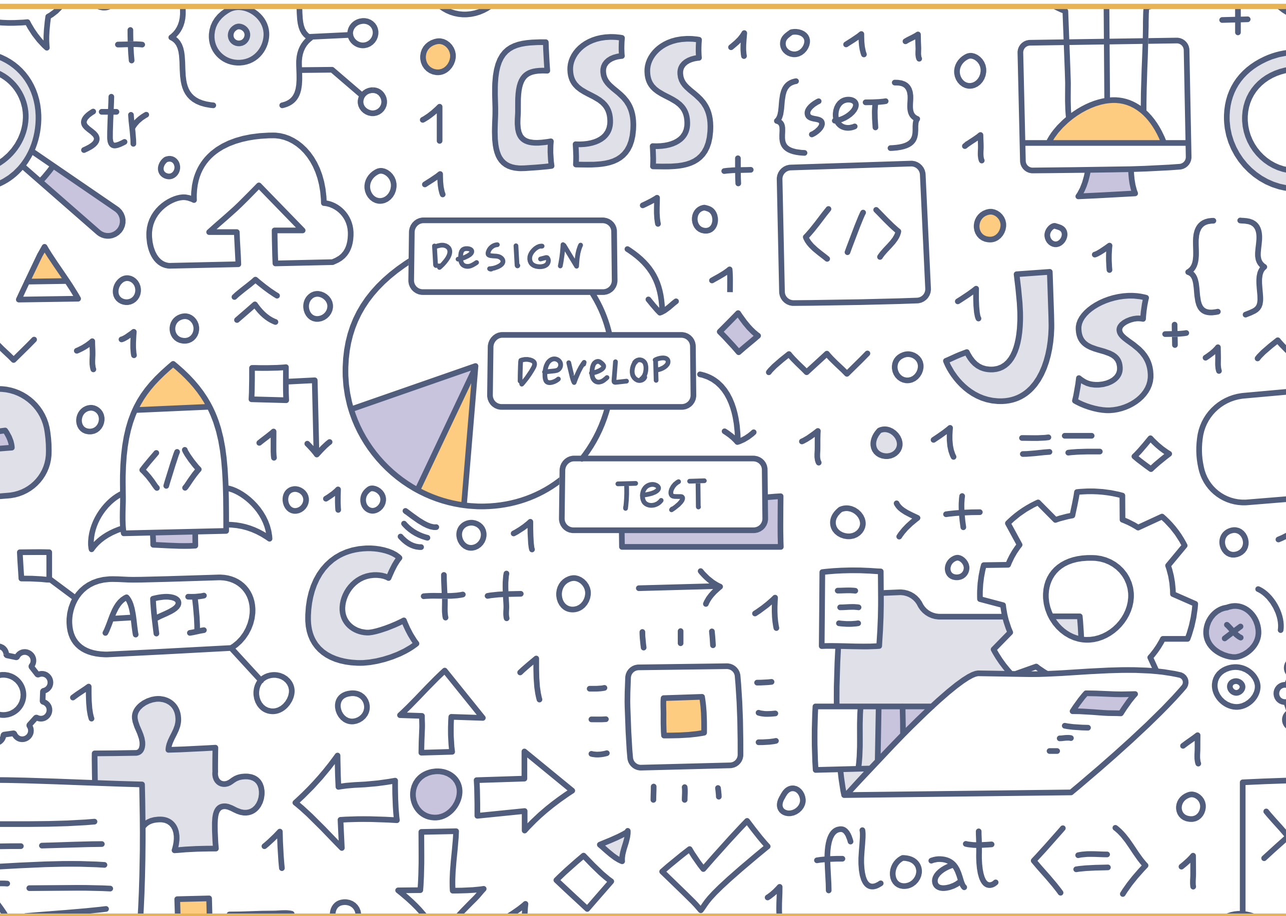
RIGA TECHNICAL UNIVERSITY

Kristaps Babris

# TWO-HEMISPHERE MODEL TRANSFORMATION-BASED GENERATION OF WEB APPLICATION USER INTERFACE PROTOTYPE

Summary of the Doctoral Thesis



RTU Press
Riga 2025

RIGA TECHNICAL UNIVERSITY

Faculty of Computer Science, Information Technology and Energy

**Kristaps Babris**

Doctoral Student of the Study Programme “Computer Science and Information Technology”

# TWO-HEMISPHERE MODEL TRANSFORMATION-BASED GENERATION OF WEB APPLICATION USER INTERFACE PROTOTYPE

**Summary of the Doctoral Thesis**

Scientific supervisor
Professor Dr. sc. ing.
OKSANA ŅIKIFOROVA

RTU Press
Riga 2025

Babris, K. Two-Hemisphere Model Transformation-Based Generation of Web Application User Interface Prototype. Summary of the Doctoral Thesis. – Riga: RTU Press, 2025. – 44 p.

Published in accordance with the decision of the Promotion Council “RTU P-07” of 7 October 2024, Minutes No. 24-5.

Cover picture from www.shutterstock.com.

https://doi.org/10.7250/9789934371295
ISBN 978-9934-37-129-5 (pdf)

# DOCTORAL THESIS PROPOSED TO RIGA TECHNICAL UNIVERSITY FOR PROMOTION TO THE SCIENTIFIC DEGREE OF DOCTOR OF SCIENCE

To be granted the scientific degree of Doctor of Science (Ph. D.), the present Doctoral Thesis has been submitted for defence at the open meeting of RTU Promotion Council on February 3, 2025, at the Faculty of Computer Science, Information Technology and Energy of Riga Technical University, 10 Zunda krastmala, Room 204.

OFFICIAL REVIEWERS

Professor Dr. habil. sc. ing. Jānis Grundspeņķis,
Riga Technical University

Professor Dr. sc. ing. Gatis Vītols,
Latvia University of Life Sciences and Technologies, Latvia

Professor Dr. Hab. Leszek Maciaszek,
Honorary Research Fellow, Macquarie University, Australia
Emeritus Professor, Wrocław University of Economics and Business, Poland

DECLARATION OF ACADEMIC INTEGRITY

I hereby declare that the Doctoral Thesis submitted for review to Riga Technical University for promotion to the scientific degree of Doctor of Science (Ph. D) is my own. I confirm that this Doctoral Thesis has not been submitted to any other university for promotion to a scientific degree.

Kristaps Babris ……………………………. (signature)
Date: ………………………

The Doctoral Thesis has been written in Latvian. It consists of an Introduction, 4 chapters, Conclusions, 76 figures, 45 tables, and one appendix; the total number of pages is 196, not including appendices (in total 229 pages). The Bibliography contains 253 titles.

# TABLE OF CONTENTS

**INTRODUCTION** ..... **5**
A problem to solve ..... 5
Research object, subject and hypothesis ..... 7
The goal and tasks of the doctoral thesis ..... 7
Research methods ..... 8
Scientific novelty ..... 10
Practical significance of the research ..... 10
Author’s publications ..... 10
Theses submitted for defence: ..... 12
Structure of the thesis ..... 12
**1. USER INTERFACE DEVELOPMENT** ..... **13**
**2. ELEMENTS OF MODEL-DRIVEN DESIGN IN THE PROPOSED SOLUTION** ..... **15**
**3. DEFINING THE TRANSFORMATION RULES** ..... **18**
3.1. Concept ..... 18
3.2. Navigation ..... 19
3.3. Presentation ..... 22
3.4. Transformation rule design principle ..... 23
**4. VALIDATION AND EVALUATION** ..... **25**
4.1. Description of the problem domain and expected outcome ..... 25
4.2. Conceptual framework and main components of the solution ..... 28
4.3. Target model source file and data format ..... 30
4.4. Mapping source and target models ..... 31
4.5. Comparing the expected user interface with the transformation result ..... 32
**RESULTS AND CONCLUSIONS** ..... **34**
**BIBLIOGRAPHY** ..... **38**

# INTRODUCTION

Software development as an engineering discipline is relatively young compared to civil engineering, medicine, mathematics, physics, etc., being 55 years old since the software crisis of 1969 (Aspray, Keil et al., 1999), and its establishment as an engineering discipline was elaborated in the final year of preparation of the Thesis. During this time, several studies have been carried out on software development methodologies, implementation solutions, technologies and architectural types. Nowadays, a set of core concepts for effective software development has been established, which are agile methodologies (Calvary, Coutaz et al., 2003; Al-Saqqa, Sawalha et al., 2020), automation (Narang & Mittal, 2022; Yigitbas, Jovanovikj et al., 2020), modular and flexible architecture (Akiki, Bandara et al., 2014; Mbuga, Korongo et al., 2022), use of frameworks, and effective testing and investment in development tools (Akiki, Bandara et al., 2014; Mbuga, Korongo et al., 2022). Introducing automation for repetitive tasks such as testing, building, and deployment significantly speeds up the development process (Nikiforova, Babris et al., 2021).

Today's software solutions are built assuming the following three perspectives:

1. Business logic components that implement the rules and processes of the application and form the basis of the system.
2. Data provides information that is essential for both analysis and operational activities, enabling informed decision-making and improving business processes.
3. The user interface provides the user experience and interaction with the system.

The user interface (UI) stands out as a critical component of the software, acting as a bridge between the user and the software, making the system's implementation options accessible and usable by the end users. The user interface is a key component of any application (Nguyen, Vu et al., 2018) and is crucial for the interaction of the application with the user. However, the user interface is not independent of its context of use, defined as the user, platform and domain (Calvary, Coutaz et al., 2003; Yigitbas, Jovanovikj et al., 2020). In fact, the business logic of an application may be the same, but the user interface may be implemented and adapted to each different platform (mobile devices, different operating systems, etc.). In addition, the pervasive digitalisation of different areas places increased demands on the speed of building information systems, where it is no longer a problem to offer software engineering solutions in any area, but the challenge is to offer them faster than the rivals. At the same time, modern user interfaces are becoming increasingly complex due to the need to support different heterogeneous application contexts, as it is no longer efficient to provide a single, one-size-fits-all user interface. However, the regular manual modification of such contexts or platforms leads to significant costs, variations and complicated management of the projects (Akiki, Bandara et al., 2014).

## A PROBLEM TO SOLVE

The problem that is addressed in this Thesis is the need to propose an approach for the development of user interface components that will enable the automated, and therefore rapid, extraction of user interface components from knowledge about the problem domain, which will, moreover, be independent of platform and implementation details, so that a sufficiently high abstraction of the problem domain knowledge is ensured. Model-Driven Development (MDD) can be a solution for these requirements, where the knowledge of the problem domain is represented as a model at the platform-independent level, and a formalism is provided to transform this model at the platform-specific level. Model-driven development started to evolve in 2001 (Lycett, Marcos et al., 2007) with the idea of transforming the software development process into a fully automated one, but despite the obvious advantages of the model-driven development (e.g. the overall view and initial representation of the system can be seen through modelling, corrections introduced to it to achieve full compliance with the knowledge of the problem domain, and only then the platform details can be added to the system model), the implementation of this development approach still has not reached the level of automation of all aspects of software development. Nowadays, model-driven development provides for the generation of static system artefacts (e.g. class definitions from entity relationship diagrams), while dynamic elements need to be worked on. The same applies to the user interface, where a qualitatively developed model can partially generate forms and their contents, but solutions for full-fledged user interface prototypes ready to be put into implementation are currently not yet developed and introduced into practice.

Recently, attempts to use generative artificial intelligence (AI) (Stige, Zamani et al., 2023; Silva (da), Martin et al., 2011) to generate user interface elements have also started to develop. While generative tools excel at some creative tasks, the nuanced and highly precise nature of user interface design poses challenges for them. The user interface variants generated by artificial intelligence (AI) are "fantasies" resulting in using of AI algorithms, which need a very precise definition of requirements and a considerable knowledge base with knowledge rules, which are, in essence, already sketches of the future user interface and require specific skills to work with generative AI tools (Alfaridzi & Yulianti, 2020; Riccio, Jahangirova et al., 2020). Defining the resulting user sketch description (Kompaniets, Lyz et al., 2020; Johnson, Gross et al., 2009) and verifying the resulting output requires excessive resources to build, test and debug (Alfaridzi & Yulianti, 2020; Riccio, Jahangirova et al., 2020). Human designers are currently still essential to ensure accuracy, functionality and user-centred aspects that are essential for effective user interface design (Newman & Landay, 2000; Li, Cao et al., 2023), and user interface sketching remains a challenge for designers to bridge the gap between conception and creation (Nikiforova, Zabiniako et al., 2021b).

In a way, the use of generative AI to generate user interfaces can also be referred to as model-based user interface design, as generative AI algorithms use knowledge about the problem domain as a source model and as a result propose variants of user interface sketches as a target model (Sharp, Rogers et al., 2019; Planas, Daniel et al., 2021). The creation and transformation of problem domain models based on metamodeling principles can enable the realisation of automatic generation of user interface content from a description of the problem domain if this description is created formally (Joo, 2019; Beek & McIver, 2021), for example, with a model that is complete and consistent. For the final software product to be of high quality, it is necessary to initially create models that contain complete and comprehensive information about the problem domain (Osis & Asnina, 2011).

## Research object, subject and hypothesis

The **research object** of the Thesis is the user interface of a Web application, especially in the context of a Web application where users expect a fast, intuitive and pleasant experience, in which the importance of the interface is even greater. The **subject of the Doctoral Thesis research** is the automated generation of a user interface prototype, where the user interface is constructed from the knowledge of the problem domain, represented as a model and contains complete and uncontroversial information about the logic of using the emerging software. It is intended that the solution proposed in the Thesis will result in user interface prototypes generated from the problem domain model that are both simple and reusable enough, and based on the principles of model-driven development (Beltramelli, 2018; Hussmann, Meixner et al., 2011), it is possible to generate new prototypes by introducing changes to the problem domain model. For such a solution to work properly, it is necessary to choose a notation for the source model that contains complete and uncontroversial knowledge about the logic of the problem domain. And knowing the content and notation of the expected target model, it is possible to define transformation rules that will be based in the metamodels of the source and target models. The two-hemisphere model (Nikiforova, Kozacenko et al., 2015) has been chosen as the source model for the solution development, as its completeness, continuity and usability have already been demonstrated in previous studies and publications (Nikiforova & Pavlova, 2011; Bajov, Nikiforova et al., 2013; Kozacenko, 2014; Nikiforova & Gusarov, 2020; Gusarov, 2020) for retrieving various software artefacts. Gusarovs (2020) has shown that:

1) the two-hemisphere model can serve as a model that contains sufficient information to generate a software system to support the problem domain;
2) business process modelling is one of the usual software engineering activities, so the need for the two-hemisphere model consisting of a business process model and a corresponding concept model does not require additional resources and changes in traditional development.

Accordingly, the **hypothesis** is that if the source code supporting business logic can be generated with the use of the two-hemisphere model (Gusarov, 2020), then it is also possible to generate the user interface prototype from the two-hemisphere model, which is essentially a representation of the result of business logic on the user side of the information system.

## The goal and tasks of the Doctoral Thesis

The **goal** of the Thesis is to develop an approach to the automatic generation of a user interface prototype from the two-hemisphere model with the defined problem logic scenarios. Furthermore, it is intended that the resulting user interface prototype is defined in the form of a model, which can be further transformed into one of the front-end component frameworks of a Web application.

To achieve the objective, the following tasks have been set:

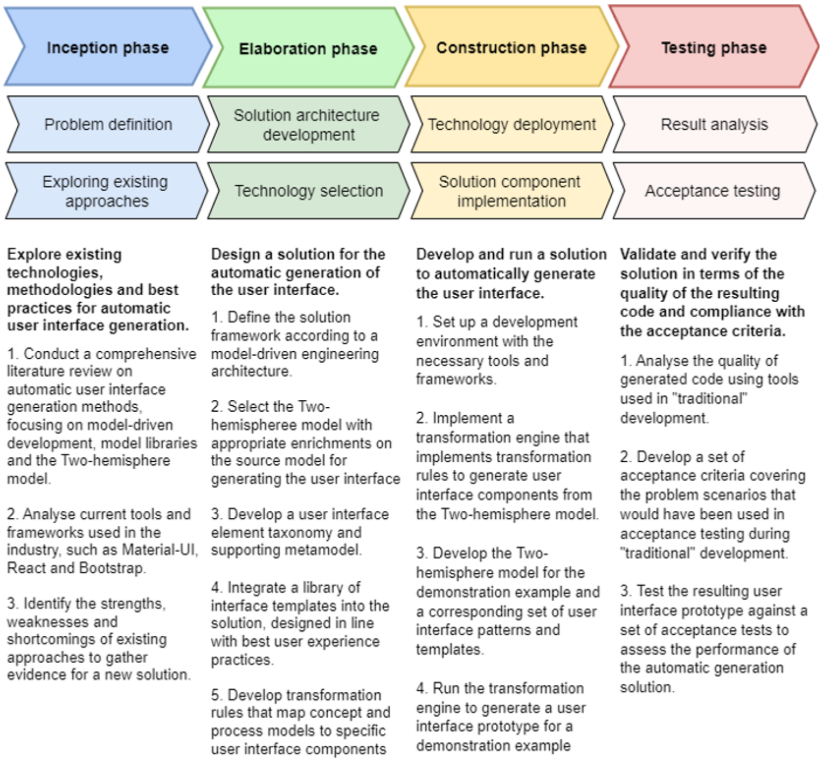
1. Investigate existing user interface development methods and techniques with a view to identifying potential basic elements and formalisms that can be used in the automatic generation of user interfaces.
2. Gather information on the diversity of user interface elements in order to define a taxonomy of user interface elements and templates that will serve as a basis for the development of the target metamodel.
3. Enrich the two-hemisphere model notation with elements necessary for transformation rules, as process metadata, and create its metamodel (as a source model).
4. Create a format for the content of the source/target model, with the intention of defining the transformation rules, where the source model is the two-hemisphere model of the problem domain, and the target model is expected to be the user interface prototype for a Web application supporting this problem domain.
5. Demonstrate the solution developed to generate the user interface prototype on a real example of a Web application for a problem domain and test its relevance to the expected outcome.
6. Evaluate the results and formulate the conclusions stemming from the study.

## RESEARCH METHODS

The research on scientific papers addressed to the problem domain and related work analysis are used as the research methods in the development of the Doctoral Thesis, which includes collection and analysis of existing scientific papers, books, monographs, and other sources in order to gain in-depth knowledge about the research object and existing solutions.

The research methodology applied in the Doctoral Thesis is defined according to the traditional phases of software development, which are inception, elaboration, construction and testing, and is shown in detail in Fig. 1 (Kruchten P., 2003). The inception phase uses theoretical research of scientific literature and analysis of related studies, which involves gathering and examining previously published materials such as scientific papers, books, monographs, and the like to acquire more comprehensive knowledge about the research object and available solutions. The elaboration phase proposes a solution based on the core concepts of model-driven software engineering: models, metamodels and model transformations. The source model for defining the transformations is the two-hemisphere model, and the target model is based on a taxonomy of user interface elements of a Web application created by the author. The user interface patterns library and the source code defined in them have been used during the execution of the transformation rules, which took place according to the "*Model-To-Text*" transformation conditions in the context defined by the object management group. Also, in essence, the generation of the corresponding user interface elements from the two-hemisphere model follows graph transformation theory, where both the two-hemisphere model and its transformation result, the user interface prototype, can both be viewed as graphs with corresponding elements (vertices) and transitions between them (edges).

The proposed solution is developed using methods and techniques of systems analysis, information analysis and synthesis, systems modelling, model transformation based on graph theory, and case study. From a practical point of view, the solution is developed using the prototyping method and includes activities such as analysing the problem and possible solution alternatives, conducting experiments and analysing their results. As a result, an engine of the transformation rules defined above has been implemented in the construction phase and validated on a small fragment of the problem domain.



**Explore existing technologies, methodologies and best practices for automatic user interface generation.**

1. Conduct a comprehensive literature review on automatic user interface generation methods, focusing on model-driven development, model libraries and the Two-hemisphere model.

2. Analyse current tools and frameworks used in the industry, such as Material-UI, React and Bootstrap.

3. Identify the strengths, weaknesses and shortcomings of existing approaches to gather evidence for a new solution.

**Design a solution for the automatic generation of the user interface.**

1. Define the solution framework according to a model-driven engineering architecture.

2. Select the Two-hemispheree model with appropriate enrichments on the source model for generating the user interface

3. Develop a user interface element taxonomy and supporting metamodel.

4. Integrate a library of interface templates into the solution, designed in line with best user experience practices.

5. Develop transformation rules that map concept and process models to specific user interface components

**Develop and run a solution to automatically generate the user interface.**

1. Set up a development environment with the necessary tools and frameworks.

2. Implement a transformation engine that implements transformation rules to generate user interface components from the Two-hemisphere model.

3. Develop the Two-hemisphere model for the demonstration example and a corresponding set of user interface patterns and templates.

4. Run the transformation engine to generate a user interface prototype for a demonstration example

**Validate and verify the solution in terms of the quality of the resulting code and compliance with the acceptance criteria.**

1. Analyse the quality of generated code using tools used in "traditional" development.

2. Develop a set of acceptance criteria covering the problem scenarios that would have been used in acceptance testing during "traditional" development.

3. Test the resulting user interface prototype against a set of acceptance tests to assess the performance of the automatic generation solution.

Fig. 1. Research detailing which was carried out as part of the Doctoral Thesis.

The validation of the solution has been carried out in two aspects. Firstly, the code quality analysis of the model transformation has been performed using *JSLint*, *JSHint* and *ESLint* to ensure that the result is consistent with programming practices and guidelines. Secondly, the Web prototype resulting from the model transformation has been tested using a standard (ISO/IEC/IEEE 29119) acceptance testing method. This validation method develops a set of acceptance criteria for a potential website against which a “manually” developed website would be validated in a “traditional” development process. By running this test set on a prototype resulting from a model transformation and verifying that the automatically extracted front-end components meet the acceptance criteria that would also be met by a “manual” development result, it is possible to verify that the proposed automation achieves the speed of development without sacrificing the quality of the result since the same criteria are also supported.

## Scientific novelty

1. An analysis of the two-hemisphere model has been performed and the possibilities of retrieving user interface prototypes from it have been evaluated. An analysis of the advantages and disadvantages of the two-hemisphere model in the context of generating user interface elements is carried out, and an improvement of the two-hemisphere model is proposed in order to generate target elements of the transformation rules, which are defined in the form of a user interface taxonomy.
2. A solution and a corresponding set of transformation rules have been developed to generate user interface prototypes from the two-hemisphere model. The solution developed in the Thesis has been validated in the *JavaScript* library *React.js* for generating prototypes in *Material Design* framework components.
3. Formats have been defined for storing and processing the source (two-hemisphere) and target (user interface prototype) models, and metamodels for both models have been specified. The metamodels can potentially be used in future research, where it will be necessary to use these model elements at the source or target level.
4. Recommendations are offered for the improvement and further development of the *BrainTool* tool developed at Riga Technical University.

## Practical significance of the research

The practical significance of the Thesis is the potential of the developed solution to generate user interface prototypes from the two-hemisphere models. The Thesis defines the necessary source and target model specification and transformation rules, which are further applicable to the implementation of the proposed solution support prototype. The solution proposed in the Thesis can be useful for software developers working with the development of different types of plug-ins/programs supporting the automation of the software development process, such as developers of integrated development environments, user interface development tools and system modelling tools. A demonstration of a solution developed using the *BrainTool* tool for building the two-hemisphere model, which has been complemented with user interface generation within the scope of this Thesis, demonstrated the potential of the tool for further development and possible commercialisation.

## Author's publications

The results of the Doctoral Thesis have been reflected in nine publications in international and recognised by the Latvian Council of Science journals and proceedings:

1. Babris, K., Ņikiforova, O., Sukovskis, U. Brief Overview of Modelling Methods, Life-Cycle and Application Domains of Cyber-Physical Systems. Applied Computer Systems, 2019, Vol. 24, No. 1, pp. 1–8, DOI: 10.2478/acss-2019-0001 (*Web of Science)*
   - Author's personal contribution: analysis of related work, development of criteria for comparison and summary.
2. Ņikiforova, O., Babris, K., Kristapsons, J. Survey on Risk Classification in Agile Software Development Projects in Latvia. Applied Computer Systems, 2020, Vol. 25, No. 2, pp. 105–116. DOI: 10.2478/acss-2020-0012 (*Web of Science)*

    - Author's personal contribution: analysis of related work, identification of risks, conclusions.
3. Ņikiforova, O., Babris, K., Madelāne, L. Expert Survey on Current Trends in Agile, Disciplined and Hybrid Practices for Software Development. Applied Computer Systems, Vol. 26 (1), 2021, pp. 38–43. DOI: 10.2478/acss-2021-0005 (*Web of Science*)
    - Author's personal contribution: analysis of related work, definition of requirements and features of the software development process, conclusions.
4. Ņikiforova, O., Zabiniako, V., Kornienko, J., Rizhko, R., Babris, K., Gasparoviča-Asīte, M. Efficiency Monitoring of Engineering System Designer Work Based on Multi-System User Behavior Analysis with AI/ML Algorithms. IEEE 62nd International Scientific Conference on Power and Electrical Engineering of Riga Technical University, 2021, pp. 1–6, DOI: 10.1109/RTUCON53541.2021.9711720 (Scopus)
    - Author's personal contribution: experiments, validating an initial version of the system model.
5. Ņikiforova, O., Zabiniako, V., Kornienko, J., Rizhko, R., Babris, K., Nikulsins, V., Garkalns, P., Gasparoviča-Asīte, M. Solution to On-line vs On-site Work Efficiency Analysis on the Example of Engineering System Designer Work. Applied Computer Systems, Vol. 26, No. 2, 2021, pp. 87–95, DOI: 10.2478/acss-2021-0011 (Scopus)
    - Author's personal contribution: experiments and validating the final version of the system model.
6. Nikiforova, O., Babris, K., Mahmoudifar, F. Automated Generation of Web Application Front-End Components from User Interface Mockups. Proceedings of International Conference on Software Technologies, SCITEPRESS Digital Library, 2024, pp. 100–111, DOI: 10.5220/0012759500003753 (Scopus)
    - Author's personal contribution: analysis of related work, development of a source and target metamodel.
7. Ņikiforova, O., Babris, K., Guliyeva, A. Definition of a Set of Use Case Patterns for Application Systems: A Prototype-Supported Development Approach. Applied Computer Systems, Vol. 29, No. 1, 2024, pp. 59–67, DOI: 10.2478/acss-2024-0008 (*Web of Science)*
    - Author's personal contribution: developing a practical example, experiments, collecting and structuring use case templates.
8. Babris, K., Nikiforova, O. Towards Automated UI Mockup Generation from Two-hemisphere Problem Domain Models: A Conceptual Framework and Approach. Proceedings of 19th Iberian Conference on Information Systems and Technologies, 2024, pp. 1–6 (in press), (Scopus)
    - Author's personal contribution: analysis of related work, development of source and target metamodels, development of solution concept.
9. Babris, K., Nikiforova, O. From Models to Interfaces: Leveraging the Two-hemisphere Model for Automated UI Generation. IEEE 65th International Scientific Conference on Information Technology and Management Science of Riga Technical University, Riga, Latvia, 2024, pp. 1–6, DOI: 10.1109/ITMS64072.2024.10741944 (Scopus)

   - Author's personal contribution: analysis of related work, development of a solution and a validation example.

The main results of the Doctoral Thesis were presented at three international scientific conferences:

1. CISTI'2024 – 19th Iberian Conference on Information Systems and Technologies, 2024, 25–28.06.2024 – Salamanca, Spain. Presentation "Towards Automated UI Mockup Generation from Two-hemisphere Problem Domain Models: A Conceptual Framework and Approach".
2. ICSOFT 2024 – 19th International Conference on Software Technologies, 08–10.07.2024 – Dijon, France. Presentation "Automated Generation of Web Application Front-end Components from User Interface Mockups".
3. ITMS'2024 – The 65th International Scientific Conference on Information Technology and Management Science of Riga Technical University, 03–04.10.2024 – Riga, Latvia. Presentation "From Models to Interfaces: Leveraging the Two-hemisphere Model for Automated UI Generation".

## THESES SUBMITTED FOR DEFENCE

1. It is possible to generate Web application user interface prototypes from the RTU-developed two-hemisphere model using the concepts of model-driven development, which are models and model transformations.
2. Automatic extraction of the Web application user interface prototype from business-level models speeds up the development process without sacrificing the quality of the result.

## STRUCTURE OF THE THESIS

The Thesis is structured as follows. Chapter 1 discusses the types and maturity levels of the user interface by analogy with the capability maturity levels and shifts the focus to the subject of the Thesis research – user interface prototype. The process of developing a user interface prototype at different levels of abstraction is also outlined. It provides a comprehensive summary of the tools and technologies that shape current user interface development and also highlights the need for simplification of the understanding and use of the interface. The basic principles of model-driven engineering, which are the source and target models and their metamodels, described in Chapter 2, have been chosen as the formalism for the generation of a user interface prototype. It explains how it is possible to transform a model into another model or source code by building abstract models. The metamodels used in the solution and described in Chapter 2 serve as the basis for the model transformation rules defined in Chapter 3. It explains the methodological way of creating these transformation rules to guarantee that they are precise, flexible and adaptable to different user interface design needs. Chapter 4 of the Thesis is devoted to the validation and evaluation of the developed solution, which is based on the development of the user interface for a certain problem domain and its comparison to the prototype developed within the application of the solution. The Thesis ends with conclusions on the results obtained and directions for future research.

# 1. USER INTERFACE DEVELOPMENT

This Chapter discusses the user interface development process, including its elements and key details. The user interface is the environment in which the user interacts with the system, receiving feedback in an understandable way. In the context of this Thesis, the user interface is meant as the user interface of a Web application. It combines visual design, interaction design and information architecture and is the front-end component of the application (Nguyen, Vu et al., 2018; Alfaridzi & Yulianti, 2020).

The development of a user interface involves several stages, from sketch to high-fidelity prototype, involving a wide range of specialists. This process is labour-intensive and involves many manual and time-consuming activities to reach the final product (Bajammal, Davood et al., 2018). In the prototyping process, concepts are transformed from drafts into usable prototypes. It is a repetitive and time-consuming process that requires several iterations until the required level of maturity is reached (Suleri, Pandian et al., 2019).

Different levels of fidelity are used in user interface design, reflecting different stages and levels of detail in the design process (Stompff & Smulders, 2015). From sketches, which are low-fidelity drawings, to wireframes, which provide a structural outline of the interface without a detailed visual style, the design process involves several stages (Rivero, Rossi et al., 2011). Mockups offer high-fidelity static visual representations with detailed styles and colours, while prototypes provide an interactive simulation of the user interface, offering a realistic user experience (Rudd, Stern et al., 1996; Virzi, Sokolov et al., 1996). The final user interface is a fully developed and ready-to-implement version with the highest level of fidelity (Rudd, Stern et al., 1996). Each of these levels helps designers to develop, refine and test design concepts at different stages, from the initial idea to the final implementation of the user interface. Sketches and wireframes provide a basic visual representation and roadmap of functionality, while mockups and prototypes offer a more detailed and realistic representation, helping to visualise and evaluate the final design before implementation (Riaz, Arshad et al., 2022; Silva (da), Martin et al., 2011; Newmanm & Landay, 2000). Prototyping can be done using exploratory, experimental or evolutionary approaches, each of which differs in its fidelity and objectives and offers different levels of detail and orientation (Nacheva, 2017).

There are three main approaches to user interface design: manual, semi-automated and automated, each with its own advantages and disadvantages (Molina, Meliá et al., 2002). Manual development, although very flexible and adaptable, requires considerable time and effort and has a high probability of errors (Molina, Meliá et al., 2002). Semi-automated development uses tools based on design patterns to ease the process and reduce development time, but this approach can cause difficulties in maintenance and updating when design changes are needed, or new technologies emerge (Pelechano, Pastor et al., 2002). Automated development provides fast results and can significantly reduce project costs through code generation, but this approach is not suitable for all systems, especially those that require specific optimisation or do not have pattern approaches (Sunitha & Samuel, 2019; Pastor, Abrahao et al., 2001). In general, each approach is suitable for different situations and the choice between them depends on the project requirements, the experience of the designer and the specifics of the problem. The focus of this Thesis is the automatic design of user interfaces.

User interface prototyping solutions include a variety of tools and approaches that help automate the design process to reduce its impact on development (Nikiforova, Babris et al., 2020). While there are many tools that support different aspects of software prototyping, there is a lack of a unified framework that covers the entire process from start to finish (Suleri, Pandian et al., 2019). Well-defined, mature and scalable user interface development processes are also needed that are aligned with organisational goals (Gilbert, Fischer et al., 2021).

Capability Maturity Model Integration (CMMI), Model Maturity Index (MMI), and user interface maturity levels provide ways to measure and improve processes, models and interfaces. CMMI levels indicate the maturity of processes in an organisation, from an initial to an optimised level (Hinderks, Mayo et al., 2022). MMI levels show the evolution of models in a project, while user interface maturity levels reflect the progression of interface design from sketches to refined and interactive interfaces. While each of these elements covers its own area, together, they all move towards greater automation and optimisation, which helps improve the quality and usability of the final product.

Analysis of existing solutions in user interface design shows that existing research only addresses some aspects of the user interface and allows partial generation of user interface prototypes (Nikiforova, Babris et al., 2024a). Although user interface generation has evolved significantly, the current state of automation tools has limitations in dealing with the complex needs of newer interfaces, such as adaptive design, multi-platform suitability and evolving content. Often, designers have to intervene manually to deal with emergency situations and to confirm that the interfaces created meet certain user requirements and application scenarios (Diehl, Martins et al., 2022). It is still difficult to make sure that the user interfaces automatically created are of good quality and can be used correctly. Although many tools have built-in validation checks and usability rules, robust testing and evaluation methods are still needed to learn about problems related to ease of use, accessibility barriers as well as design differences. User interface generation methods using AI often do not offer transparency or interpretability (Alfaridzi & Yulianti, 2020; Riccio, Jahangirova et al., 2020). Trust in the approach and the ability to understand the results of generation algorithms can be a barrier for user interface designers. As AI-driven user interface generation becomes more common (Stige, Zamani et al., 2023), there are also growing concerns about ethics, such as lack of objectivity in algorithms, accuracy of results, privacy risks and possible replacement of human designers. Designers and developers need to think carefully about the implications of automated user interface design and find ways to prevent any damage. Once these hurdles are overcome, automated user interface generation can have a significant impact on the way software is developed, making it easier for designers to create user interfaces that are more efficient and human-centred.

# 2. ELEMENTS OF MODEL-DRIVEN DESIGN IN THE PROPOSED SOLUTION

The analysis of related studies that have proposed different ways in which user interface design can be automated has concluded that models and metamodels are sufficient formalisms for user interface design, as a set of user interface elements can be described as a model (Escalona, García-Borgoñón et al., 2021), and hence apply the principles of model-driven design, where the basic concepts are models, metamodels and transformation rules (see Fig. 2.1).

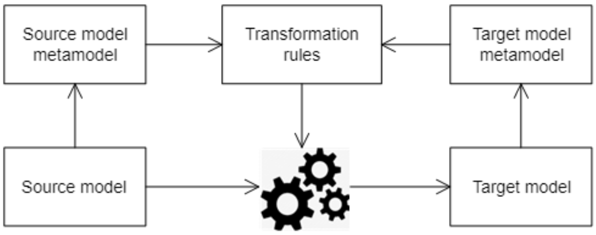


Fig. 2.1. Conceptual scheme of model-driven development (adapted from (Kleppe, Warmer et al., 2003)).

Model-driven development is based on the automatic generation of software code from domain models, where the core activities are domain analysis and modelling, meta-modelling, model-driven code generation, pattern languages, and the development of domain-based frameworks (Völter & Bettin, 2004). The following principles of model-driven development are applicable in the context of user interface design:

1. A model, or models, describing user interface scenarios and containing complete and continuous information about the knowledge and context of the domain – a source model.
2. All necessary information about the user interface forms, their content and the transitions between them is collected and represented in the target model.
3. Source and target models allow the definition of source and target metamodels, expressed using UML or a domain-specific language.
4. Transformation rules are defined that allow the target model to be obtained by applying predefined transformation rules to the source model (Babris, Nikiforova et al., 2019).

The result of the transformation can be used as it is obtained after applying the transformation rules or manually updated. To be usable, the transformation result must be runnable code or importable into an integrated development environment that can process the transformation result. This chapter describes the elements of a user interface to allow the construction of a target model and a target metamodel for defining transformation rules and describes the two-hemisphere model and its adaptation for defining transformation rules so that the resulting user interface can be prototyped.

Figure 2.2 shows the user interface metamodel for organising a *React.js*/*Redux.js* Web application for automatic user interface creation. The metamodel focuses on how the user interface containers and components are related and dependent on each other. This metamodel shows what elements are expected to be created as a result of a solution to automatically build a *React.js*/*Redux.js* Web application, separating functionality into Containers and Components, as well as the inclusion of *Redux.js* for data management and *XState* for process organisation. This approach offers a flexible and scalable framework for the automatic generation of user interfaces.

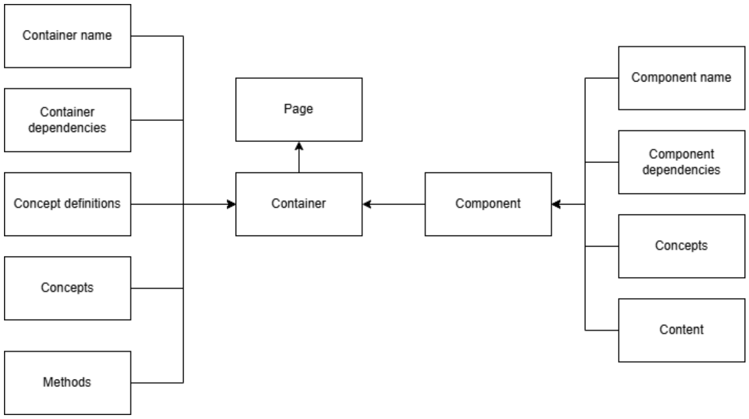


Fig. 2.2. User interface metamodel for the solution to be developed.

The metamodel divides a web app into Pages, Containers and Components. Pages consist of Containers that handle the current state of a process and define concepts and methods. Components represent individual user interface elements that receive data and methods from their containers via Properties. A Container is defined by a name, a list of dependencies, and a concept definition section, which also contains variables and methods. In the same way, Components are defined with names, dependencies, concepts and, most importantly, content, which contains the appearance and the behaviour. The connections between Pages, Containers and Components ensure that everything is set up in a modular way to facilitate the maintenance of the system. This metamodel helps to transform business process models into functional user interfaces by mapping entities to *React.js* components. This structured framework helps to easily create dynamic user interfaces with states, keeping a clear distribution of responsibilities and increasing flexibility.

The two-hemisphere model-driven approach (Nikiforova, 2009) is one of the model-driven software development approaches designed and developed at Riga Technical University and first published in 2004 (Nikiforova & Kirikova, 2004). By contrast to other model-driven approaches developed at the time (Nikiforova, Kozacenko et al., 2015), the two-hemisphere model-driven approach proposed to base software development on two interrelated models, namely a process model that described the functioning of the domain, and a concept model that represented the data structures present in the domain. It was the link between these two models that enabled the automatic transformation of domain analysis information into software system design models, which were in the form of various UML diagrams. This transformation thus offered a broader set of design artefacts for further code generation directly from the domain. The two-hemisphere model-driven approach was validated in several domains (Nikiforova, Sukovski et al., 2015; Nikiforova, el Marzouki et al., 2017; Nikiforova & Gusarov, 2020; Nikiforova, Iacono et al., 2020) and applied in several PhD theses as a formalism to justify transformations (Pavlova, 2008; Gusarov, 2020; Marzouki (el), 2021).

The metamodel of the two-hemisphere model is shown in Fig. 2.3. It can be seen that the two-hemisphere model essentially includes a conceptual diagram and one or more business process diagrams. The linking of these two models is represented by relating the data structure from the conceptual diagram to the data flow element in the process diagram (Nikiforova, 2002). Each data flow is explicitly linked to one concept, although each concept may be linked to several process flows. This relationship forms the basis for assigning responsibility to object classes during subsequent transformations.

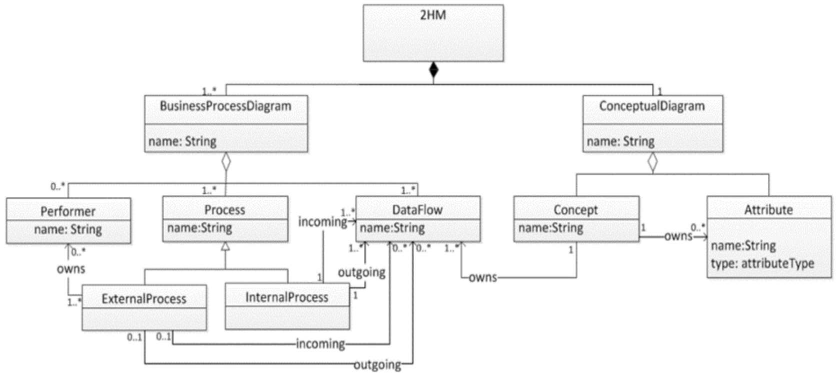


Fig. 2.3. The two-hemisphere metamodel (derived from (*Kozačenko*, 2014)).

The work is based on a collection of user interface elements, which allows to identify a set of elements necessary for the user prototype of the interface as a target set. Moreover, the existence of previous research on the development and application of a user interface metamodel for defining transformations at different levels of abstraction gives additional confidence in the correctness of the problem-solving path chosen in this Thesis. The existing metamodels served as a basis for the creation of a unified user interface metamodel, which can be used not only in the solution developed within this Thesis but can be further used by other researchers (Babris & Nikiforova, 2024a).

As well as the practical application of the two-hemisphere model, the metamodel developed in previous studies already contains a complete set of elements that can potentially be used to define transformations for the generation of user interface elements. In order to be able to apply the formalisms of the two-hemisphere model to the automatic generation of a user interface prototype, the next Chapter describes the development of transformation rules in the expected elements of the target model according to its metamodel.

# 3. DEFINING THE TRANSFORMATION RULES

Model-driven development for application software engineering challenges dates back to the 2000s with the very attractive idea of creating a platform-independent model for software development; its transformation from platform-independent to a platform-specific model would further encourage the use of automatic code generation (Suleri, Pandian et al., 2019; Stahl, Völter et al., 2006). The idea was promising but, for various reasons, utopian (Hailpern & Tarr, 2006; Thomas, 2004). However, the formalism it proposes (Trehan, Chapman et al., 2015) is quite useful in tasks where a complete and consistent model can be transformed, using metamodeling principles, into its knowledge presented in a different format (Nikiforova & Gusarovs, 2020; Domingo, Echeverría et al., 2020).

In the previous chapter, a metamodel of the target model was developed, according to which the components expected in the user interface prototype are divided into three groups (Leuthold, 2010; Koch & Mandel, 1999):

1. Concept elements.
2. Navigation elements.
3. Presentation elements.

The source model is the two-hemisphere that has been previously used for the automatic generation of design and software components.

This chapter defines a set of transformation rules for transforming a source model into a target model according to the groups of elements defined in the target metamodel. In addition, for missing elements in the source model, the possibilities of using other formalisms to cover the complete extraction of the target model elements are explored.

## 3.1. Concept

The activities of the conceptual design step are defining classes, specifying attributes and operations, defining hierarchical structures and defining subsystems. To achieve this, well-known object-oriented modelling techniques are used.

The classes and associations defined in this activity are also used during navigation design to derive the structure links of the web application. Relations will be used to obtain links. Classes and associations can be organised in groups. Classes are described using attributes and actions and represented graphically (Koch & Mandel, 1999). The result of the conceptual development step is summarised in a conceptual model consisting of classes and associations between classes that model the problem domain. In this Thesis, the concept model generated from the two-hemisphere model is used as a basis, which allows finding relationships between different data models.

This information allows the creation of something like an ER diagram from which user interface elements can be generated. Mapping concept attributes to user interface elements ensures that the associated information is displayed and can be interacted with correctly in the UI. Not all information from the concept model needs to be visible to the user; in fact, it is necessary to specify which fields are available to the user and which should be hidden. In addition, this addition of field attributes needs to be manageable; for example, during editing or form input, a field may be hidden, but in read mode, the field and its contents may be visible.

User interface templates that already have the knowledge of how the data is to be displayed can help to identify data input and output. Without using user interface patterns, a user interface prototype can be retrieved, as shown in Fig. 3.1.

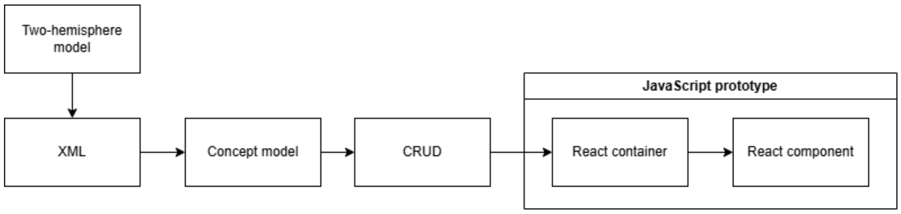


Fig. 3.1. Extraction of concept view from two-hemisphere model.

After some experiments with the concept, it was found that in the two-hemisphere model (there are only eight types in the *BrainTool*), there is a limited number of concept field types, which means that, at the moment, it is not possible to use a complete mapping of user interface elements. It can be concluded that the two-hemisphere model needs to be supplemented with other field types. On the positive side, it is quite easy to point to related entries in the concept. In this case, user interface patterns may be useful, which would already specify the layout and form elements and would only need to be linked to the field types and names in the concept.

Similar to another study (Mahatody, Ilie et al., 2021), the types of concept fields should be added, e.g.:

1) *Text*, to set the long text input option;
2) *Enum*, to help define short drop-down lists;
3) *Date* un *Datetime*, to help create date and time fields (e.g. from a database).

Perhaps the use of user interface patterns with a predefined concept related to the user interface elements would solve this problem. For example, hiding and displaying fields in read and write mode could be solved by applying a defined user interface pattern in each case. The Thesis goes on to describe the information related to the navigation.

## 3.2. Navigation

For the navigation, a state chart (Sunitha & Samuel, 2019) is used, which is transformationally derived from the two-hemisphere model. This is similar to other studies, such as extracting state charts from requirements specifications (Pimentel, Castro et al., 2014; Briand, Labiche et al., 2005). User interface navigation can be represented as a state chart that the user moves through using navigation.

It is also possible to generate state charts from UML Sequence Diagrams, for example, using graph transformations (Grønmo & Møller-Pedersen, 2011). Other authors have proposed to generate user interfaces from state charts (Horrocks, 1999; Wellner, 1989; Harel, 1987), using states at different resolutions of the user interface – from the operation of basic user interface elements such as buttons to overall user interface navigation schemes. Within this Thesis, state charts are used as the basis for a common navigation model that the end user can navigate without having to go into each user interface element (Fig. 3.2).

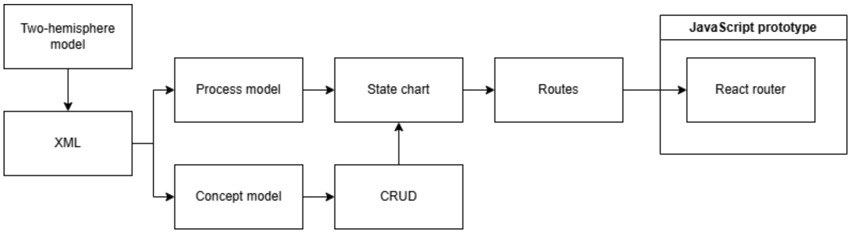


Fig. 3.2. Extraction of navigation view from the two-hemisphere model.

Based on the conceptual model, a navigation model is created based on the navigation design, which is a view of the conceptual model. This navigation model is produced in two stages (Koch & Mandel, 1999):

1) which objects are potentially reachable using navigation;
2) how these objects are reached.

The navigation model requires the use of knowledge from other objects to perform object grouping (Koch & Mandel, 1999). Model-driven approaches have the goal of automatically generating code from models. Thus, in each model-driven methodology, work is done with models and transformations. These models have to correspond to the corresponding metamodels. The transformations, in turn, are described using transformation languages (Gharaat, Sharbaf et al., 2021).

To perform the transformation to a state chart, it is necessary to determine what states the system can be in. From the given example of the processes of the two-hemisphere model, it can be concluded that the system can be in 4 states. Each state (except the "*Add Item to Cart*" state) has its own visual representation, which means that the notation of the two-hemisphere model needs to be supplemented with how the process is to be represented – either it is a state with its own view or it is an intermediate state, in fact, to be interpreted as an event. In addition, "*Add Item to Cart*" from the user interface is essentially a process or event, so we need to skip it and analyse the next process, in this case, "*Show Cart*". If the intermediate state can further progress to more than one process, then this should be considered when designing the two-hemisphere process model so that the notation does not need to be unnecessarily extended. The diagram below shows the process path of the two-hemisphere model using the proposed online shop example. This state chart has been generated using *BrainTool* as the two-hemisphere model building tool and the *XState* library, mapping the two-hemisphere model processes using the above approach. Transformation rules for generating navigation elements.

Figure 3.3 shows a general flowchart for extracting a position diagram from the two-hemisphere model. The first step is to extract an XML file from the *BrainTool*, where the two-hemisphere model is developed. Next, the data is extracted from the XML file and normalised to extract the process tree (which states are available and where to go next), the concepts which are passed on to the state chart as context and the methods, if any, for the developed the two-hemisphere model. The methods are added to the state chart in the context accordingly. The attributes developed for each concept are created for the context of the state chart, and values are added according to the data types of the attributes.

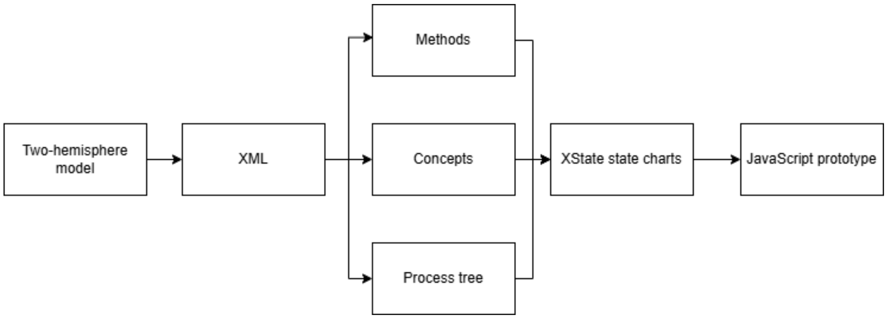


Fig. 3.3. Extraction of state charts from the two-hemisphere model.

The process model of the two-hemisphere model is very close in nature to a state chart, as it contains states (processes) and transitions between them (connections), but no information to indicate whether the current state is a page that needs to be presented to the user or a state to be ignored from a user interface point of view. The author combines the two-hemisphere model with *XState* state charts and *React Router* for declarative routing in *React.js* applications. In fact, other libraries and frameworks can be used – this does not change the principle. This approach structures user interactions, as *XState* defines user interactions with clear states and transitions. As well as declarative routing, *React Router* simplifies user interface navigation based on URL changes and provides further declarative options for use in development. This method can handle both the functional (invisible to the user) and the presentational parts (states visible to the user), but in this case it is necessary to extend the two-hemisphere model with the possibility to determine whether the current process should be shown to the user or not. For large applications, state management can become complex and we may have to consider how to make these states, routes and transitions more modular.

Because the two-hemisphere model processes cannot specify whether the current process is a state with its own presentation template or an intermediate state (transition), it is necessary to add a type to the two-hemisphere model process that, from the user interface side, determines whether the process has its own template that needs to be displayed, or whether it is a process that the user does not need to see. This option is discussed in the next section, which looks at retrieving a presentation.

The two-hemisphere model may not be extended with this type of attribute if the next section can determine whether the current process should be visible to the user or not. Such an approach could facilitate the development and maintenance of the two-hemisphere model.

## 3.3. Presentation

Presentation design involves modelling the abstract user interface, showing how the navigation structure is demonstrated to the user (Koch & Mandel, 1999). Presentation design determines the way navigation nodes will appear, select user interface objects to activate navigation, and determine which interface transformations will take place. The same navigation structure can provide different presentations depending on the constraints of the target platform and the technology used (Koch & Mandel, 1999).

Using user interface patterns defined by state name, possible future states and data model (concept), it is possible to demonstrate a responsive user interface that displays equally well in Web and mobile applications. The size and position of the elements to be demonstrated are determined by the *Material-UI* and user interface patterns. Pairs of labels and text fields, grid, alignment of individual labels or buttons, etc. can be demonstrated with predefined user interface patterns. Figure 3.4 shows the overall scheme for extracting interface views from the two-hemisphere model combining both interface content elements (concepts) and navigation elements.

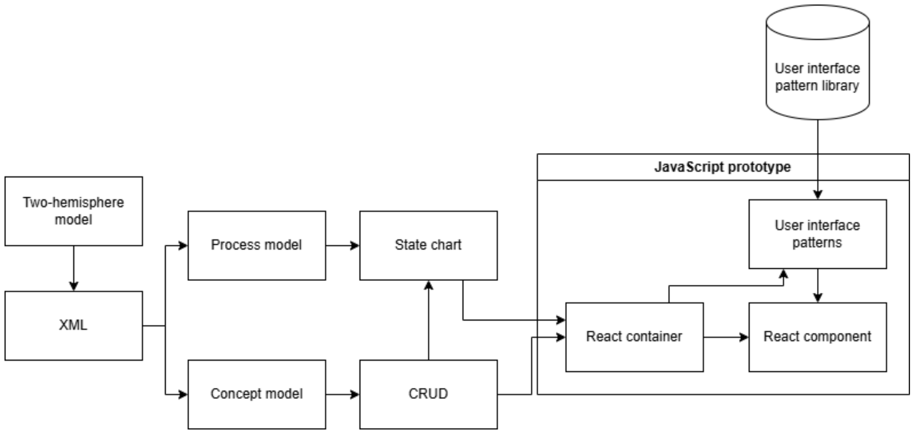


Fig. 3.4. Extraction of views from the two-hemisphere model.

The first step is to extract an XML file from the *BrainTool*, where the two-hemisphere model is developed. Next, the data is extracted from the XML file, and the process and concept models are extracted. From the concept model, a CRUD mapping is created, and in parallel, by retrieving the state chart, clear names for Containers and Components can be created. The *React.js* Container is responsible for mapping data and processes for the *React.js* component, which is actually a static view that only displays data, but all events are handled by the *React.js* Container. Using the existing state chart and concept model with CRUD mapping, a usable user interface prototype can be created, however, there is potential to further enhance these views by adding new elements to enable the representation of more complex and element-rich user interfaces. A new element that has emerged in this scheme is the need for user interface patterns. There are several approaches to creating interface patterns.

In order to apply user interface patterns to the processes of the project to be developed, it is necessary to identify these patterns and map them to the processes. In this Thesis, the author reviews the ways of semi-automatically or automatically applying development patterns found in the literature. It is assumed that defining and selecting development patterns is similar to defining user interface patterns.

In this Thesis, a text classification approach is chosen as a selection method for low-cost preprocessing of user interface patterns, making it faster, simpler and cheaper than other approaches. The proposed method was evaluated using three groups of design patterns and sufficient descriptions of real design problems (Hasheminejad & Jalili, 2012).

Just as the use of software patterns in code generation improves the quality of the generated code (Sunitha & Samuel, 2019), the use of user interface patterns improves the quality of usability, with navigation and presentation that people are familiar with.

The presentation design is the task of defining the presentation of the navigation objects retrieved in the previous steps in the form of a static and a dynamic presentation model (Koch & Mandel, 1999). The static presentation model associates at least one presentation object with each navigation object and the dynamic presentation model describes the behaviour of these presentation objects (Koch & Mandel, 1999). In the user interface generation process, CRUD actions can also be extracted using defined patterns (Nasiri, Rhazali et al., 2023) so that each process and concept is transformed into a user interface element (or set of elements) (Antović, Vlajić et al., 2012).

## 3.4. Transformation rule design principle

Transformation rules need to be created to match process names with patterns from the user interface pattern library. The aim of transformation rules is to automatically suggest user interface patterns from the pattern library, relying on process names that indicate both the name of the pattern and its function.

First, it is necessary to identify the keywords (Haz, Funabiki et al., 2024). The process name of the two-hemisphere model is used to retrieve the user intent. For example, the process name "*Show Cart*" contains both "*Show*" and "*Cart*", which indicates that the process is related to the shopping cart and display function. Second, the identified keywords must be mapped to the user interface patterns. The identified keywords for the process names of the two-hemisphere model shall be mapped to the pattern library of the user interface patterns. This library associates keywords or functionality with certain user interface patterns. For example, "*Create*" can be associated with a form pattern, and "*Item*" with a specific data entry form subpattern in the user interface pattern library. Thirdly, the names of context or concept elements can optionally also be considered to more accurately map user interface patterns. For example, "*Search Items*" may offer a "*Search Bar*" pattern, but if the process includes filtering by certain criteria, then a more precise "*Filter Panel*" template would be a better fit. The effectiveness of these transformation rules depends largely on the comprehensiveness and clarity of both the process names and the library of user interface patterns. In order to formally structure the necessary steps to be performed, it is proposed to define the user interface pattern process as a set of tuples.

There may be several situations where you need to find the most accurate pattern. The pattern library must be well-defined with clear descriptions and examples for each pattern, as the text classification method will select patterns based on keywords *(*Hasan, Sanyal et al., 2017). However, the transformation rules can be adapted depending on the application's domain and functions. Further research can consider machine learning techniques to improve the accuracy of pattern selection by analysing previous mappings and user interactions.

Keyword identification and matching is the key idea to turn process names into user interface patterns. Keyword matching enables to map keywords from a process name to predefined patterns in the library. If there are several possible matches based on the keywords, the user interface pattern library can offer priority levels for the patterns. Higher priority patterns are usually more specific and have priority. This approach helps to offer a primary user interface pattern that corresponds to the functionality of the process. In the "*Create Customer Account*" example, the "*User Registration Form*" is recommended as the initial option. Efficiency depends largely on the accuracy and detail of both process names and keywords associated with user interface patterns in the user interface pattern library. This approach can automate user interface design based on process definitions, which, in turn, can speed up user interface development. This approach also promotes consistency of user interface patterns throughout the application. However, the disadvantages are that complex processes may require manual intervention to select the most appropriate pattern, and the effectiveness of this approach depends on the quality of the process names and the library of user interface patterns.

To start the transformation process, the two-hemisphere model is needed at the beginning. Most often, this is created in *BrainTool* and consists of a process model and a concept model. *BrainTool* offers to export this model to an XML file format. Processing this XML file produces both a process model and a concept model in a form that can be processed outside the *BrainTool* tool. The concept model is further mapped with CRUD to retrieve the entities of the data model and decide what parameters should be passed. The extracted objects and variables are further mapped to the demonstration data so that the prototypes have completed views and can see roughly what the finished user interface will look like. Variables with set values are further defined in the *React.js* container. In turn, a state chart is extracted from the process model, which is further mapped to *React Router* routes. Routes are defined separately, along with *React.js* container calls, to specify which container is displayed to the end user on which route call. Once both variables and routes are prepared, a *React.js* container component is created, which is selected based on the process model item name using a text classification method – retrieved from the user interface pattern library. This produces a *JavaScript* prototype of the two-hemisphere model. Combining model transformation with the latest front-end technologies, such as *React.js*, speeds up development time and improves overall software quality. The two-hemisphere model provides a useful structure for developers to map processes and concepts to front-end interfaces. This, in turn, can help software projects develop faster, meeting user requirements and business goals – bridging the gap between technical features and end-user experience.

# 4. VALIDATION AND EVALUATION

The first step of the solution developed in the Thesis involves understanding the specific domain for which the user interface prototype is being developed and developing the source model (two-hemisphere model). The source model is then fed to a transformation rule engine, resulting in a user interface prototype as a set of screen source files that implement the functionality of the domain and the user interaction in the domain. Automatic generation of user interface prototypes uses predefined user interface patterns as the basis for design. User interface patterns provide design solutions for common user interface elements and functions, promoting an intuitive and user-friendly interface (Nikiforova, Babris et al., 2024b). Using pre-defined patterns can speed up the prototyping process compared to building a user interface from scratch. User interface patterns ensure consistency between prototypes and in user interactions throughout the application. The consistency of the transformation result with the expected result demonstrates the workability of the solution.

In order to validate the developed solution in terms of the quality of the result obtained, it is necessary to analyse the transformation result in two aspects:

1. Code quality, which refers to the overall efficiency and maintainability of the code. This refers not only to the correct functioning of the code but also to how well it is written, structured and documented (Nikiforova, Zabiniako et al., 2021a). The code must be easy to understand for any developer who is familiar with the programming language used. This includes proper indentation, meaningful variable names and comments explaining complex logic, if necessary.
2. The web application was matched to the business process of the domain and to the expected use case scenarios, which were related to the functioning of the code. The functionality of the application can be verified by an acceptance testing method (ISO/IEC/IEEE 29119, 2013), which can validate that the resulting prototype works correctly and fits the business processes identified in the domain.

This Chapter demonstrates the application of the developed solution on a small abstract validation example, which proves the solution's workability, and describes the acceptance testing of the transformation result, which validates the application of the solution.

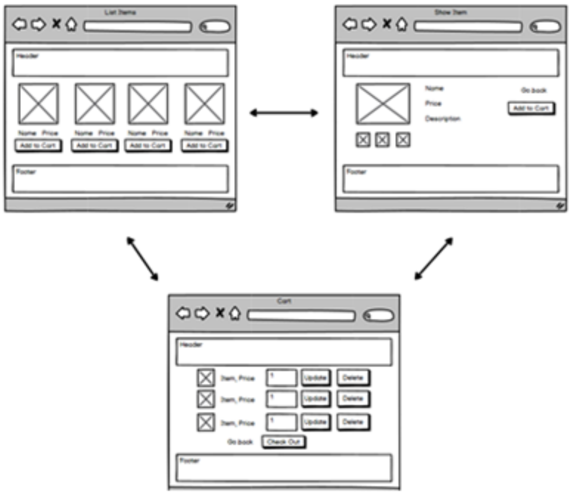
## 4.1. Description of the problem domain and expected outcome

An example was chosen to demonstrate the proposed approach. In this case, the example is a product sold by an online shop – a modular system that can be easily installed on the chosen platforms without any specific customisation processes. The product can be purchased either on a monthly basis or on a yearly subscription basis, with the option to automatically renew the subscription if selected. In this process, users can register, log in, view the products, add them to the cart and, finally, create an order. The proposed process outlines a typical (Fernández, Liu et al., 2021) user interaction with an online shop. The process can start with registration, application and viewing the product list. After browsing the products, you can add the product to your shopping cart. The shopping cart can be managed – items can be removed, or quantities can be changed. You can then place an order. The process is described in more detail in (Babris & Nikiforova, 2024b):

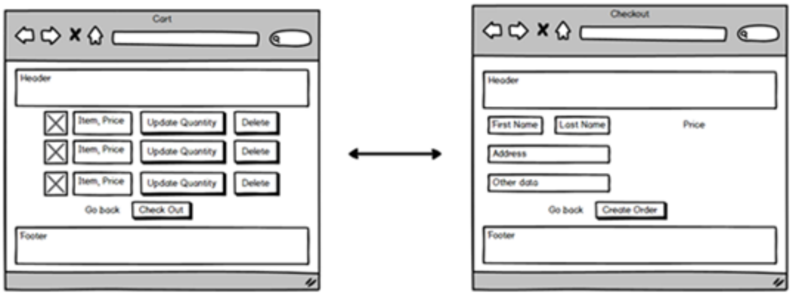
1. User registration – when visiting the online shop website, the user activates the "*Register*" button and is redirected to a registration form where the user enters his/her personal data, such as name, email address, and password. After filling in the required information, the user submits the form. The system creates a new user's account.
2. User's login – if the user already has an account, they can log in by clicking on the "*Log in*" button. The user enters their email address and password in the login form. After submission, the system checks the credentials and, if correct, grants access to the user's account.
3. Browsing – the user can browse the catalogue of products in the online shop. The user can use the search functionality to find specific items that interest them. Each item is displayed with its name, price, description and the option to add it to the cart.
4. Adding items to the cart – when the user finds an item he/she wants to buy, the user clicks on the "*Add to Cart*" button. The selected item is added to the user's shopping cart, which shows a summary of all items added so far. The user can adjust the quantity of each item in the cart or remove items altogether.
5. Cart management – in the cart view, the user can see all added items, as well as their quantities and prices. The user can update the quantity by adjusting the quantity field or remove items by clicking on the "*Remove*" button.
6. Checkout process – after completing the selection, the user proceeds to the checkout view by clicking on the "*Checkout*" button. The user is presented with a form in the checkout view where he confirms his/her delivery address. Once the required information has been submitted, the user submits the payment form, and the system processes the order.
7. After placing an order, the user is redirected to his/her profile.

Users navigate from the product catalogue to see detailed information about the specific products they want to view. This allows users to easily explore product information such as descriptions, images and specifications. After checking the product details and making a purchase decision, users can seamlessly navigate to the shopping cart area to add the selected products. Alternatively, they can add an item to their cart immediately from the product catalogue. Figure 4.1 (a) shows a diagram of a catalogue, an item and a cart, how to navigate through these elements.

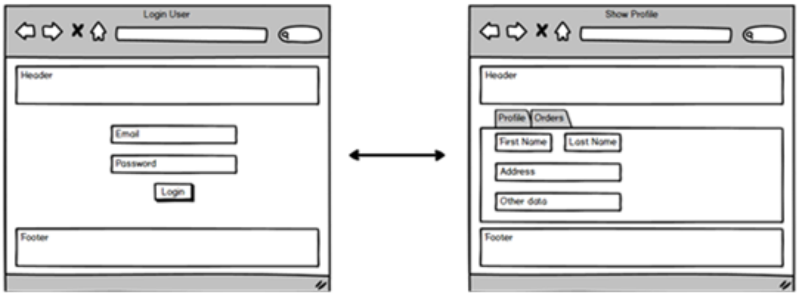
From the shopping cart, the user can either go to the payment section, where the user has to enter the details for invoicing, select the payment methods, etc., or back to the product catalogue (Fig. 4.1 (b) shows the shopping cart and payment scheme). When the order is completed, the user is redirected to the user profile (Fig. 4.1 (c)). The user is also redirected to the user profile if the user has logged in to the system. On logging out of the system, the user is redirected to the login view.



(a) Catalogue, item and cart interaction schema



(b) Cart and checkout interaction schema

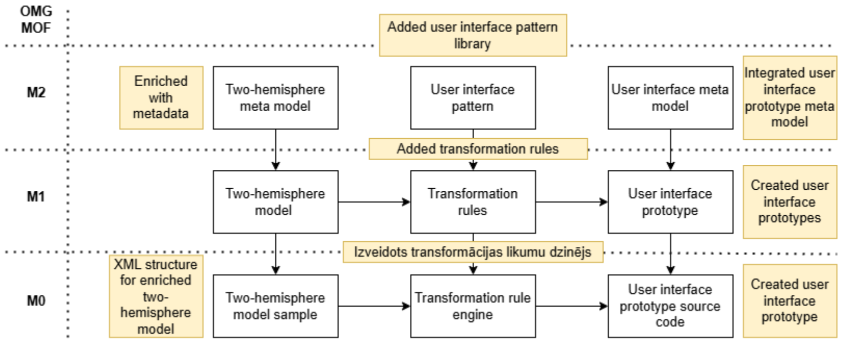


(c) User login and profile interaction schema

Fig. 4.1. Screen interaction sketches.

## 4.2. Conceptual framework and main components of the solution

The solution for generating the user interface prototype developed in the Thesis is shown in Fig. 4.2, showing the solution components according to the Meta Object Facility (MOF) framework of the Object Management Group (https://www.omg.org/mof/).



OMG MOF
M2
M1
M0
Added user interface pattern library
Enriched with metadata
Two-hemisphere meta model
User interface pattern
User interface meta model
Integrated user interface prototype meta model
Added transformation rules
Two-hemisphere model
Transformation rules
User interface prototype
Created user interface prototypes
Izveidots transformācijas likumu dzinējs
XML structure for enriched two-hemisphere model
Two-hemisphere model sample
Transformation rule engine
User interface prototype source code
Created user interface prototype

Fig. 4.2. Conceptual diagram of the solution for generating a user interface prototype from the two-hemisphere model.

The white blocks represent the components of the solution. The comments on the yellow background describe additions, modifications or new developments that have been made to the artefacts within the Thesis. The central component of the solution is the transformation rules, which define how user interface elements are built from the elements of the two-hemisphere model, following the corresponding user interface patterns. The solution implements them in the PHP programming language. The transformation rules use user interface patterns that can be selected depending on the process metadata. The transformation rules engine (software) runs transformation rules on the two-hemisphere model XML file and produces the source code for the front-end components of the Web application in *JavaScript* programming language. The two-hemisphere model metamodel and the user interface metamodel describe the content and structure of the source and target models for defining transformation rules. The two-hemisphere model has been extended with process metadata compared to its original version. The user interface meta-model is built by integrating existing user interface metamodels and restructuring them with respect to the mapping of the front-end application system components and the expected set of Web application user interface elements (described as a UML class diagram). The user interface prototype is essentially a working Web application in the form of interacting screen forms running in a Web browser. And the result of the transformation is the source code of the front-end components of the Web application that can be run in an integrated development environment with the support of a framework chosen for implementation (the Thesis uses *React.js*, *Redux.js*, and *XState* libraries for the demonstration example).

To demonstrate and validate the solution, Fig. 4.3 shows the two-hemisphere model for a selected domain, with the *BrainTool* tool specifying the relationships between the relevant process flows and concepts. The domain is a typical online shop, which can be used to select and subscribe to a few products, and does not consist of category hierarchies or other complex classification systems. The main requirement is that a potential or existing user of the company should be able to pay for a subscription to the system for a period of time of his/her choice.

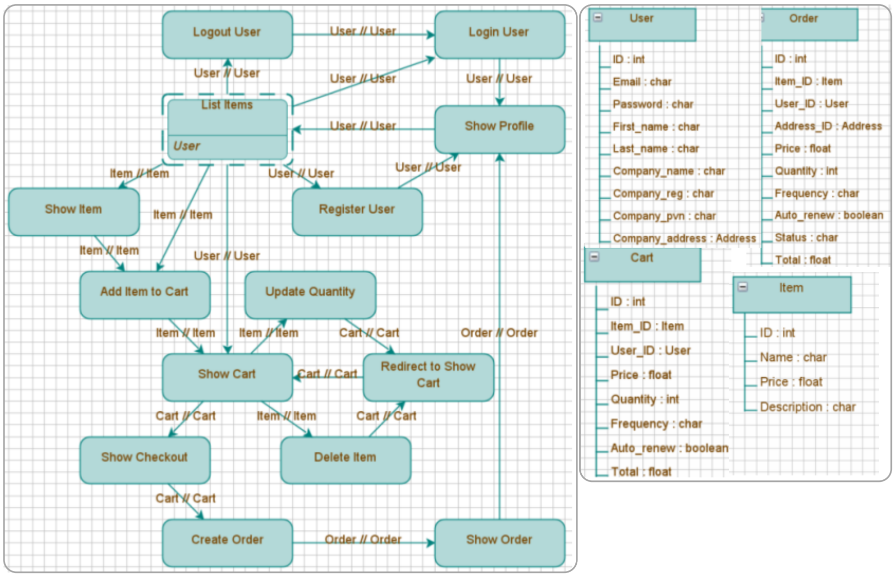


Fig. 4.3. Processes (left) and concepts (right) of the two-hemisphere model of the problem domain.

As the two-hemisphere model is built in *BrainTool*, it is possible to transform the information about the concepts, processes and their relationships into a processable form, i.e. export the corresponding XML (Extensible Markup Language) file.

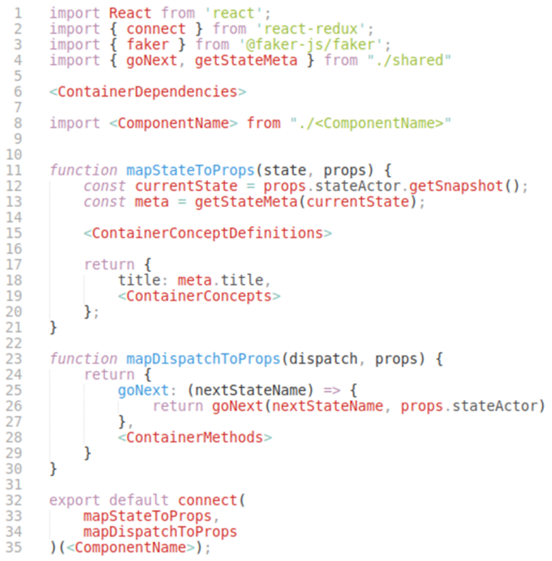
XML is a markup language that defines a set of rules for encoding documents in a format that is both human and machine-readable. It is typically used to store structured data in a hierarchical format. The source model of the two-hemisphere model is usually an XML file representing processes and concepts. In this two-hemisphere model, the structure of the source model XML file defines the elements and attributes that can be used to generate user interface components, as well as their properties, relationships and behaviour. In the two-hemisphere model, XML elements could describe entities, functions, links and actions, which would be further used in the proposed solution. It is also possible to generate sequence diagrams (Nikiforova, Kozacenko et al., 2013) and UML class diagrams (Nikiforova & Pavlova, 2011) from the two-hemisphere model, which shows that the binary model is functional enough to generate user interface elements from it.

## 4.3. Target model source file and data format

The output of the solution is presented as *React.js JavaScript* source code. Each framework or library can develop its own model transformation rules to get a complete user interface.

Given that one of the most detailed approaches to user interface modelling is Interaction Flow Modelling Language (IFML), the proposed solution borrows elements such as *Container* and *Component* to separate functional and contextual elements from presentation elements. Applying this architecture can result in *React.js* and *Redux.js* code.

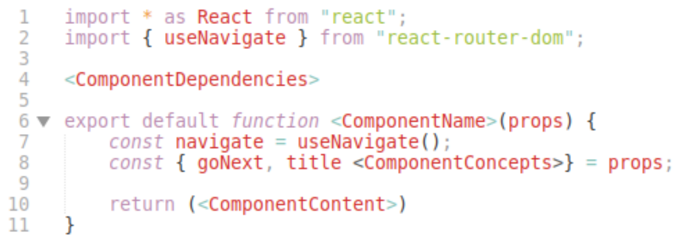
*React.js* and *Redux.js* Web applications have many parts that help to organise and control the user interface and state management. For example, <*ComponentName*> is the technical name of a *React.js* component that can be reused in different places in the project. This name is responsible for displaying a specific part of the user interface on the screen while controlling how that area works. The element named <*ContainerDependencies*> contains pointers to the components or containers to be used, it can also contain pointers to the use of certain libraries. The <*ContainerConceptDefinitions*> element contains the main data structures or entities handled by the Web application. Using a demonstration data generator, these data structures can be populated in a way that is easy for the user to understand and that creates a prototype that is as close as possible to a real user interface. The <*ContainerConcepts*> element is a list of predefined <*ContainerConceptDefinitions*> elements that are passed to the component from the container definition. In fact, <*ContainerConcepts*> is a simplification of <*ContainerConceptDefinitions*>. The last element <*ContainerMethods*> may contain different methods or functions belonging to this container component. Typically, these methods handle things like user actions and data processing, as well as navigation (Fig. 4.4).



```
import React from 'react';
import { connect } from 'react-redux';
import { faker } from '@faker-js/faker';
import { goNext, getStateMeta } from "./shared"

<ContainerDependencies>

import <ComponentName> from "./<ComponentName>"


function mapStateToProps(state, props) {
    const currentState = props.stateActor.getSnapshot();
    const meta = getStateMeta(currentState);

    <ContainerConceptDefinitions>

    return {
        title: meta.title,
        <ContainerConcepts>
    };
}

function mapDispatchToProps(dispatch, props) {
    return {
        goNext: (nextStateName) => {
            return goNext(nextStateName, props.stateActor)
        },
        <ContainerMethods>
    }
}

export default connect(
    mapStateToProps,
    mapDispatchToProps
)(<ComponentName>);
```

Fig. 4.4. Basic container template.

Also, in the case of a component, <*ComponentName*> is the technical name of the component and the element named <*ContainerDependencies*>, like <*ContainerDependencies*>, contains pointers to the components or containers to be used, as well as to pattern libraries or other necessary dependencies. Elements, <*ComponentConcepts*> as well as <*ContainerConcepts*>, are a list of predefined concept elements that are passed to the component from the container definition. A component can be reused in this way in other parts of the project (Fig. 4.5).



```
import * as React from "react";
import { useNavigate } from "react-router-dom";

<ComponentDependencies>

export default function <ComponentName>(props) {
    const navigate = useNavigate();
    const { goNext, title <ComponentConcepts>} = props;

    return (<ComponentContent>)
}
```

Fig. 4.5. Basic component template.

The last element <*ComponentContent*> is the most important element because it contains the set of elements to be displayed to the user, defined in the user interface template.

Because prototypes are generated, without data, there is no way for the end user to fully understand what the results will look like in the user interface. To address this problem, it is proposed to use demonstration data, which do not carry any information but are only used to show the user as realistic a prototype as possible.

## 4.4. Mapping source and target models

Mapping the source and target model to *React Router*, containers and components means transforming the representation of the architectural model into a layout of routes and components in the *React.js* Web application.

*React Router* is a tool that helps to connect each container and component to a specific route in a *React.js* Web application. This requires defining routes that correspond to the different processes of the two-hemisphere model.

By performing these steps, developers can map the process flow of the two-hemisphere model to the *React Router*, containers and components. This creates a single, functional Web app prototype that the user can interact with. It is also necessary to map the concepts of the two-hemisphere model to *React.js* containers and components so that it is possible to know what data models are needed in each container in order to be able to define them. As can be noticed, if the variable type is an array, then the two-hemisphere model concept has to be used in plural and is converted to an array of objects at transformation. However, if the type of the expected variable is an object, then the variable is called singular and left as the name of the two-hemisphere model concept. Variable types are defined at the user interface pattern, this is done manually when creating the pattern.

## 4.5. Comparing the expected user interface with the transformation result

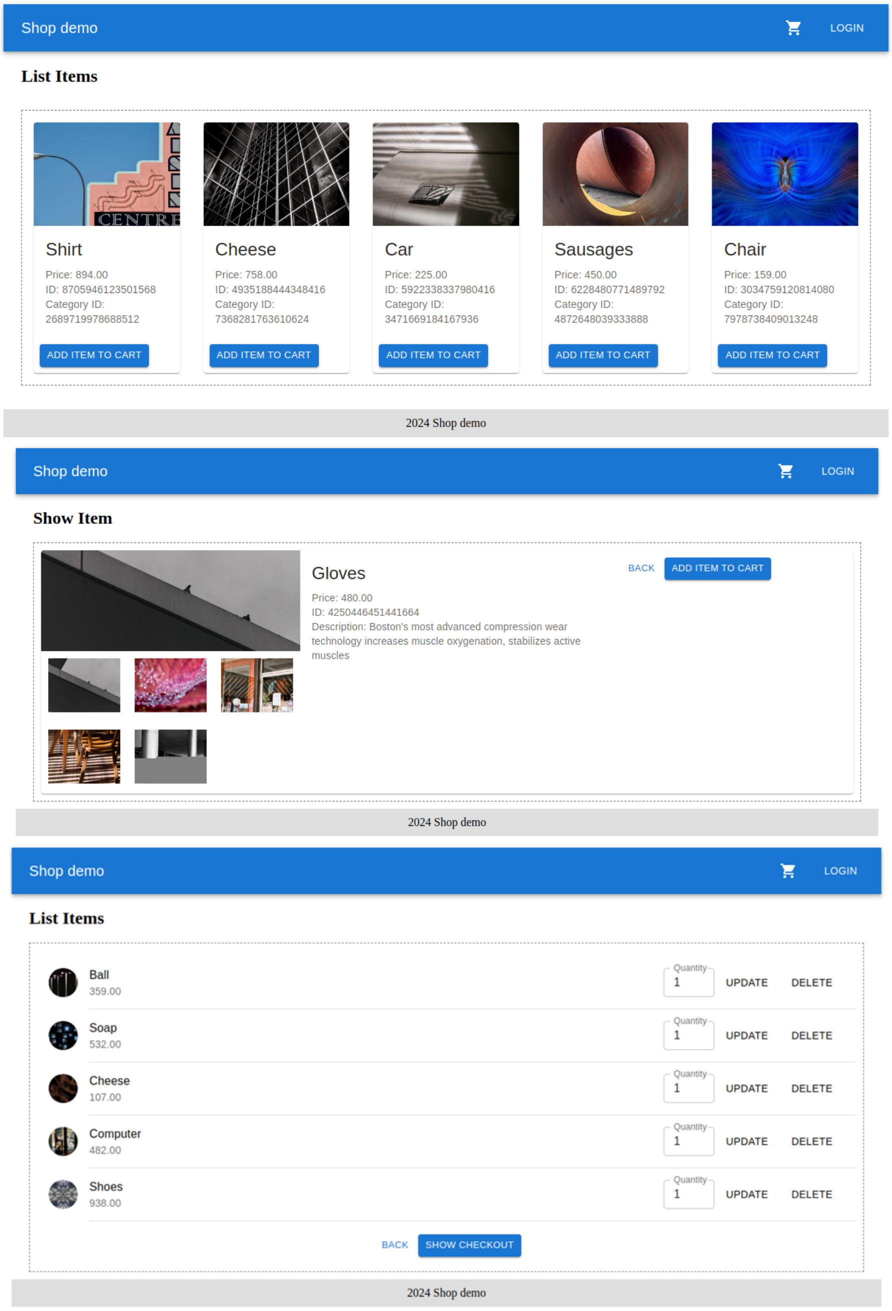
If the above procedure is successful, then the address can be opened and results seen. The “List Items” should appear as the initial view.

The “List Items” section (top screenshot of Fig. 4.6) is a basic section designed to display available items and help users browse through them. This section, which is automatically created according to user interface patterns and design guidelines, guarantees consistency and user-friendliness throughout the shopping interface. This section displays product thumbnails arranged in a grid or list view, where each thumbnail represents one item that can be added to the cart. Users can navigate through the catalogue and view an item or add it to the cart immediately. The “Show Item” view (central screenshot in Fig. 4.6), often referred to as the product information page or product page, is an essential part of an online shop. It serves as a special place where users can study detailed information about a specific product before making a purchase decision. The product information page displays a picture of the product together with a thumbnail gallery, a description of the product, the price and the option to add it to the cart. In the “Show Cart” section (bottom screenshot of Fig. 4.6), each item in the cart is accompanied by its thumbnail image, name, unit price, and selected quantity. Next to each item are interactive elements that allow customers to update the quantity or remove the item completely from the cart. When customers adjust quantities or delete items, the cart is dynamically updated to reflect these changes, ensuring that total costs are accurately calculated in real time.

This work proposes to validate the generation of user interface prototypes against manual results, in the context of an online shop. The validation is based on the fact that an expected result is developed for the problem domain and compared with the transformation result using the algorithm described in the paper. The Thesis defines acceptance criteria that have been identified by analysing the functionality in the process model. For an automatically generated user interface prototype, the validation focuses on confirming that the prototypes developed correctly represent the defined functional requirements and user interactions. The quality of the generated code is also examined by checking it with tools such as *JSLint*, *JSHint* and *ESLint*. The Thesis also discusses other important factors such as usability, performance and security.

Model-driven approaches allow software developers to focus on high-level design issues. However, there is still debate, for example, about how effective automated code generation techniques will be in complex projects, or how these techniques will work in different programming languages (Uyanık & Sayar, 2024).

The validation and acceptance testing of the solution allows us to claim that the two-hemisphere model offers a valuable framework for structuring user interfaces. The transformation process, driven by user interface patterns and metamodels, translates this model into concrete user interface components and interactive prototypes. Using this method, user interface prototypes can be designed to fit well with the basic architecture of the system. The resulting automatically generated user interface prototype is consistent with the expected functionality, which has been verified by acceptance testing in the Thesis.


Shop demo

LOGIN

List Items

Shirt
Price: 894.00
ID: 8705946123501568
Category ID:
2689719978688512

ADD ITEM TO CART

Cheese
Price: 758.00
ID: 4935188444348416
Category ID:
7368281763610624

ADD ITEM TO CART

Car
Price: 225.00
ID: 5922338337980416
Category ID:
3471669184167936

ADD ITEM TO CART

Sausages
Price: 450.00
ID: 6228480771489792
Category ID:
4872648039333888

ADD ITEM TO CART

Chair
Price: 159.00
ID: 3034759120814080
Category ID:
7978738409013248

ADD ITEM TO CART

2024 Shop demo

Shop demo

LOGIN

Show Item

Gloves
Price: 480.00
ID: 4250446451441664
Description: Boston's most advanced compression wear technology increases muscle oxygenation, stabilizes active muscles

BACK ADD ITEM TO CART

2024 Shop demo

Shop demo

LOGIN

List Items

| | | | |
|---|---|---|---|
| Ball 359.00 | Quantity 1 | UPDATE | DELETE |
| Soap 532.00 | Quantity 1 | UPDATE | DELETE |
| Cheese 107.00 | Quantity 1 | UPDATE | DELETE |
| Computer 482.00 | Quantity 1 | UPDATE | DELETE |
| Shoes 938.00 | Quantity 1 | UPDATE | DELETE |

BACK SHOW CHECKOUT

2024 Shop demo

Fig. 4.6. Generated user interface with catalogue, item and shopping cart pattern.

# RESULTS AND CONCLUSIONS

In software engineering, the user interface became an essential part of it after the software engineering crisis in 1969. This situation revealed serious difficulties in software development, such as project budget overruns and difficulties in delivering projects on time or at all. The problem is that as programs become more complex and difficult to handle, it becomes vital for developers not only to build them but also to provide users with ways to interact effectively with the projects they develop. A well-designed user interface has thus become one of the main answers to these problems, improving usability – making complex software systems easier to use by applying intuitive design principles. User interface design plays an important role in improving user satisfaction, reducing training and support costs by making user interfaces easier to understand and use. Efficient user interface design with simple development processes minimises user errors and ensures that tasks can be performed quickly and accurately. By fostering better communication between users and developers, user interface design makes it easier to create software that more accurately meets users' requirements. In addition, modern user interface design techniques allow to adapt to different needs through iterative, user-centred development and design, such as user interface patterns. In fact, focusing on user interface design addresses many of the problems associated with the software engineering crisis.

Another important aspect of software engineering is the rapid development of software components without sacrificing quality. Hence, this Thesis is devoted to the development of a solution based in the basic concepts of model-driven engineering, which enables the automatic generation of source code for the front-end components of a Web application, which, on the one hand, speeds up the development of the user interface component and on the other hand ensures the quality of the transformation result.

The tasks defined for the present Thesis are fully fulfilled:

1. Methods and techniques for user interface development are explored with a view to identifying potential basic elements and formalisms that can be used in the automatic generation of user interfaces.
2. Information on the diversity of user interface elements is gathered in order to define a taxonomy of user interface elements and templates that will serve as a basis for the development of the target meta-model.
3. The two-hemisphere model notation is enriched with elements necessary for transformation rules – process metadata.
4. The content of the source and target models is created in order to define the transformation rules, where the source model is the two-hemisphere model of the problem domain and the target model is expected to be a user interface prototype for a Web application supporting this problem domain.
5. A solution for generating user interface prototypes is developed, demonstrated on an example of a real problem-supporting Web application, and tested against the expected outcome.
6. The solution is evaluated, and conclusions are made.

The **main result** of this Thesis is a proposed solution that enables the generation of a Web application user interface prototype from the two-hemisphere model using the basic concepts of model-driven development, which are metamodeling, models and their transformations.

The Thesis examines the importance of user interface design in modern applications. It covers various rules and methods of user interface design, emphasising the requirement for interfaces that are easy to understand and use by users. Current styles and technologies in user interface design are also explored, highlighting their impact on users' experience and software usability.

The potential of model-driven development to improve user interface design is discussed. The basic idea of a model-driven approach is to create abstract models that can be automatically transformed into executable code. The aim is to increase efficiency, reduce errors and guarantee uniformity across different software platforms. The Thesis provides a thorough explanation of the tools and frameworks that support model-driven development.

The Thesis examines in detail the nature of the model-driven approach. Transformation rules are defined that transform high-level models into real implementations. It also presents how to create these rules and make sure that they are correct, flexible and can be adapted to different user interface needs. Some examples of transformation rules are presented, showing how they are used in practical situations. In addition, the difficulties and the main techniques for implementing and processing these rules to obtain the best results are also discussed.

The last chapter is devoted to the validation of the proposed model-driven user interface development solution. It explains the process of application and evaluation of the proposed solution. Test results are presented to show how accurate the proposed solution is with respect to what is expected. The Thesis concludes with summarising all the findings, highlighting the advantages and possible limitations of the solution. It offers recommendations for future research and progress in the field of model-driven user interface design.

The Thesis is a detailed study on the improvement of user interface design using a model-driven approach. The study proposes a solution that combines abstract modelling with real implementation using exact transformation rules, with the aim of improving the design, uniformity and ease of use of software user interfaces.

Overall, the following can be considered as the results of the Thesis:

1. Information on the element sets present in front-end programming frameworks for Web applications is summarised, which served as a basis for the creation of a taxonomy of user interface elements for the target elements of the transformation rules.
2. User interface metamodel parameters are defined and a metamodel is created to perform transformations from the two-hemisphere model by augmenting the process with user interface patterns.
3. The addition of the two-hemisphere model notation in the process model metadata to label processes that need to be exposed to the user interface development process and that are not relevant for prototyping and the introduction of additional attribute data types for concept model elements is proposed.

4. Transformation rules are defined, which, when used in conjunction with a user interface pattern library and a pattern selection method, can be run to automatically generate user interface prototypes.
5. A solution is proposed for the automatic generation of a user interface consisting of components according to the basic principles of model-driven engineering.
6. The practical applicability of the proposed solution is demonstrated by using it to build the front-end components of a software system.
7. The possibilities of developing and enriching the *BrainTool* for the two-hemisphere model support with features such as user interface patterns, process choice prediction linked to user interface pattern library element links, etc., are outlined.

The work carried out and the results obtained justify the assertion that the theses defined in the introduction to the Thesis have been confirmed:

1. From the RTU-developed two-hemisphere model, it is possible to generate Web application user interface prototypes using the basic concepts of model-driven development, which are models and model transformations – as evidenced by the developed solution and its demonstration on a validation example.
2. The automatic extraction of a Web application user interface prototype from business-level models speeds up the Web application development process without sacrificing the quality of the result – as demonstrated by the analysis of the transformation result both in terms of the quality of the extracted code and the compliance of the extracted Web application with the acceptance criteria of the problem domain.

Based on the research carried out and the results obtained, the following conclusions are drawn:

1. It is now common in model-driven engineering to use UML to define artefacts, but this is more for describing off-the-shelf solutions and can be difficult for problem domain experts to understand. However, the study suggests that the two-hemisphere model is more suitable for generating user interface prototypes.
2. The two-hemisphere models must be of high quality and strictly defined, and the user interface patterns and their library must be of high quality. The code of the patterns themselves must be written correctly so that user interface prototypes can be generated.
3. The notation of the two-hemisphere model almost completely handles the generation of the user interface, but the notation needs to be extended – in the concept model with additional field type values and in the process model addressing the issue, whether a particular process is visible to the user or not.
4. *BrainTool* can be extended with user interface patterns and process prediction. For example, when a new process is created, the following processes would be automatically created if this information is stored in the user interface pattern library.

**Practical importance.** The developed solution is proposed to be used for the development of user interfaces for Web applications by automatically generating front-end component source code for prototypes. This approach aims to address common problems in software development, such as project budget overruns and project deadlines, by using model-driven engineering principles to improve development efficiency and ensure development quality.

By defining a target metamodel and transformation rules, the solution enables the automatic generation of user interface prototypes from high-level abstract models. This reduces manual programming effort, speeds up the development process and reduces human errors.

The use of standardised patterns and transformation rules ensures that the generated user interfaces are consistent in design and conform to best practices, resulting in higher quality and more maintainable code.

The proposed solution aims to streamline the user interface prototyping process, making it faster, more efficient and capable of producing high quality, user-centric interfaces. This approach not only addresses some long-standing software engineering challenges but also lays the foundation for future advances in model-based user interface development.

By modifying the existing two-hemisphere model notation and adapting it for transformation into the source code of user interface prototypes, the prototype generation algorithm developed in the Thesis can be implemented in the industry after its implementation in the corresponding support tool because the algorithm enables to extract the source code of user interface prototypes from a model that is understandable to experts in the problem domain.

The conclusions and results obtained in the course of the study may also be of interest to a wide range of professionals, both in industry and in further research in the field of model-driven software. The solution developed in the framework of the Thesis enables the software code engineering problem to be solved in one direction from the problem model to the code. The architecture of the proposed solution also allows the investigation of re-engineering of the code, which is to ensure that changes made in the code can be maintained with respect to the content of the model. In the framework of the Thesis, the problem of code reengineering is beyond the scope of the study.

Another future research direction would be to complement the existing tool, *BrainTool*, with the approach discussed in the Thesis or to create a new editor to implement the Thesis approach in a user-friendly environment. The *BrainTool* could be extended to allow the user, when creating the processes of the two-hemisphere model, to automatically suggest what the next processes should be and, consequently, the user interface patterns. In addition, the list of user interface patterns could be extended with definitions and different user interface frameworks to allow the generation of prototypes across a wider range of technologies.

A more important research would be to use the possibilities offered by artificial intelligence to automatically create and link together the necessary patterns from the user interface library descriptions, thus reducing the manual effort to develop user interface patterns.

Given the current widespread development of different user interface frameworks such as *Material Design*, *Ant Design*, etc., it would be valuable to investigate how the semantics and other approaches of these frameworks can be used to automatically create the desired user interface elements just by specifying which user interface framework to use.

# BIBLIOGRAPHY

Akiki P., Bandara A., Yu Y. (2014) Adaptive Model-Driven User Interface Development Systems. ACM Comput. Surv. 47, 1, Article 9 (July 2014), 33 pages. DOI: https://doi.org/10.1145/2597999.

Alfaridzi M. D., Yulianti L. P. (2020) UI-UX Design and Analysis of Local Medicine and Medication Mobile-based Apps using Task-Centered Design Process, 2020 International Conference on Information Technology Systems and Innovation (ICITSI), 2020, pp. 443–450, DOI: https://doi.org/10.1109/ICITSI50517.2020.9264947.

Al-Saqqa S., Sawalha S., Abdel-Nabi H. (2020) Agile Software Development: Methodologies and Trends. International Journal of Interactive Mobile Technologies (iJIM), 14, 246, DOI: https://doi.org/10.3991/ijim.v14i11.13269.

Antović I., Vlajić S., Milić M., Savic D. (2012) Model and software tool for automatic generation of user interface based on use case and data model, IET Software, 6, DOI: https://doi.org/10.1049/iet-sen.2011.0060.

Aspray W., Keil R., Parnas D. (1999) History of Software Engineering.

Babris K., Nikiforova O. (2024a) Towards Automated UI Mockup Generation from Two-Hemisphere Problem Domain Models: A Conceptual Framework and Approach, Proceedings of 19th Iberian Conference on Information Systems and Technologies (CISTI 2024, June 25–28), 2024 (accepted for publication) (SCOPUS).

Babris K., Nikiforova O. (2024b) From Models to Interfaces: Leveraging the Two-Hemisphere Model for Automated UI Generation, 2024 IEEE 65th International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS), Riga, Latvia, 2024, pp. 1–6, submitted for publication (SCOPUS).

Babris K., Nikiforova O., Sukovskis U. (2019) Brief Overview of Modelling Methods, Life-Cycle and Application Domains of Cyber-Physical Systems. Applied Computer Systems, 2019, Vol. 24, Issue 1, pp. 1–8, DOI: https://doi.org/10.2478/acss-2019-0001 (Web of Science).

Bajammal M., Davood M., Ali M. (2018) Generating reusable web components from mockups. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18). Association for Computing Machinery, New York, NY, USA, 601–611, DOI: https://doi.org/10.1145/3238147.3238194.

Bajovs A., Nikiforova O., Sējāns J. (2013) Code Generation from UML Model: State of the Art and Practical Implications. Scientific Journal of Applied Computer Systems, 14, 7–18, DOI: https://doi.org/10.2478/acss-2013-0002.

Beek (ter) M. H., McIver A. (2021) Formal methods: practical applications and foundations. Form Methods Syst Des 58, 1–4, DOI: https://doi.org/10.1007/s10703-021-00380-6.

Beltramelli T. (2018) Pix2code: Generating Code from a Graphical User Interface Screenshot. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '18). Association for Computing Machinery, New York, NY, USA, Article 3, 1–6, DOI: https://doi.org/10.1145/3220134.3220135.

Briand L. C., Labiche Y., Lin Q. (2005) Improving statechart testing criteria using data flow information, 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), Chicago, IL, USA, 2005, pp. 10–104, DOI: https://doi.org/10.1109/ISSRE.2005.24.

Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J. (2003) A Unifying Reference Framework for multi-target user interfaces, Interacting with Computers, Volume 15, Issue 3, June 2003, pp. 289–308, DOI: https://doi.org/10.1016/S0953-5438(03)00010-9.

Diehl C., Martins A., Almeida A., Silva T., Ribeiro Ó., Santinha G., Rocha N., Silva AG. (2022) Defining Recommendations to Guide User Interface Design: Multimethod Approach. JMIR Hum Factors. 2022 Sep 30, 9(3), e37894, DOI: http://doi.org/10.2196/37894.

Domingo A., Echeverría J., Pastor O., Cetina C. (2020) Evaluating the Benefits of Model-Driven Development: Empirical Evaluation Paper. DOI: https://doi.org/10.1007/978-3-030-49435-3_22.

Escalona M. J., García-Borgoñón, L., Koch, N. (2021) Don't Throw your Software Prototypes Away. Reuse them! In Insfran, E., González, F., Abrahão, S., Fernández, M., Barry, C., Linger, H., Lang, M., Schneider, C. (Eds.), Information Systems Development: Crossing Boundaries between Development and Operations (DevOps) in Information Systems (ISD2021 Proceedings). Valencia, Spain: Universitat Politècnica de València.

Fernández E., Liu Y., Pan R. (2001) Patterns for Internet shops. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8ad2c6f226600828dab4e5317ff713603e003383.

Gharaat M., Sharbaf M., Zamani B. et al. (2021) ALBA: a model-driven framework for the automatic generation of android location-based apps. Autom Softw Eng 28, 2, DOI: https://doi.org/10.1007/s10515-020-00278-3.

Gilbert D., Fischer H., Röder D. (2021) UX at the Right Level: Appropriately Plan the UX Expertise Using the PUXMM – A UX Maturity Model for Projects. i-com, Vol. 20, Issue 1, pp. 105–113, DOI: https://doi.org/10.1515/icom-2020-0029.

Grønmo R., Møller-Pedersen B. (2011) From UML 2 Sequence Diagrams to State Machines by Graph Transformation, Journal of Object Technology, Vol. 10, pp. 8:1–22, DOI: http://doi.org/10.5381/jot.2011.10.1.a8.

Gusarovs K. (2020) Metodes izstrāde koda ģenerēšanai no divpusložu modeļa, promocijas darbs, Rīgas Tehniskā universitāte, 2020, DOI: https://doi.org/10.7250/9789934225772.

Hailpern B., Tarr P. (2006) Model-driven development: The good, the bad, and the ugly, in IBM Systems Journal, vol. 45, no. 3, pp. 451–461, DOI: https://doi.org/10.1147/sj.453.0451.

Harel D. (1987) Statecharts: a visual formalism for complex systems, Science of Computer Programming, Volume 8, Issue 3, 1987, pp. 231–274, ISSN 0167-6423, DOI: https://doi.org/10.1016/0167-6423(87)90035-9.

Hasan H. M., Sanyal F., Chaki D., Ali Md. (2017) An empirical study of important keyword extraction techniques from documents. DOI: https://doi.org/10.1109/ICISIM.2017.8122154.

Hasheminejad S. M. H., Jalili S. (2012) Design patterns selection: An automatic two-phase method. Journal of Systems and Software, 85, 408–424, DOI: https://doi.org/10.1016/j.jss.2011.08.031.

Haz A. L., Nobuo F., Fajrianti E. D., Sukaridhoto S. (2024) A Study of Summarization and Keyword Extraction Function in Meeting Note Generation System from Voice Records. In Proceedings of the 2023 12th International Conference on Networks, Communication and Computing (ICNCC '23). Association for Computing Machinery, New York, NY, USA, 106–112, DOI: https://doi.org/10.1145/3638837.3638853.

Hinderks A., Mayo F. J. D., Thomaschewski J., Escalona M. J. (2022) Approaches to manage the user experience process in Agile software development: A systematic literature review, Information and Software Technology, Volume 150, 2022, 106957, ISSN 0950-5849, DOI: https://doi.org/10.1016/j.infsof.2022.106957.

Horrocks I. (1999) Constructing the User Interface with Statecharts (1st. ed.). Addison-Wesley Longman Publishing Co., Inc., USA, ISBN:978-0-201-34278-9.

Hussmann H., Meixner G., Zühlke D. (2011) Model-Driven Development of Advanced User Interfaces, DOI: https://doi.org/10.1007/978-3-642-14562-9.

ISO/IEC/IEEE 29119 Software Testing Standards, International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE), 2013.

Johnson G., Gross M., Hong J., Do E. (2009) Computational Support for Sketching in Design: A Review. Foundations and Trends in Human-Computer Interaction, 2, 1–93, DOI: http://doi.org/10.1561/1100000013.

Joo H. (2017) A Study on Understanding of UI and UX, and Understanding of Design According to User Interface Change, in International Journal of Applied Engineering Research ISSN 0973-4562, vol. 12, no. 20, pp. 9931–9935, 2017.

Kleppe A. G., Warmer J., Bast W. (2003) MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publ., USA. ISBN: 978-0-321-19442-8.

Koch N., Mandel L. (1999) Using UML to design hypermedia applications. Technical Report 9901, Institut für Informatik, Ludwig-Maximilians-Universität, München, March 1999.

Kompaniets V., Lyz A., Kazanskaya A. (2020) An Empirical Study of Goal Setting in UX/UI-design, 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT), 2020, pp. 1–5, DOI: https://doi.org/10.1109/AICT50176.2020.9368570.

Kozačenko L. (2014) Divpusložu modeļa lietošanas analīze UML diagrammu ģenerēšanā, Maģistra c, Rīgas Tehniskā universitāte, 2014.

Kruchten P. (2003) The Rational Unified Process: An Introduction (3rd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA, ISBN: 978-0-321-19770-2.

Leuthold, S. (2010) User Interface, Navigation Design and Content Representation: Three Perspectives on World Wide Web Navigation. Doctoral Thesis, University of Basel, Switzerland.

Li J., Cao H., Lin L., Hou Y., Zhu R., El Ali A. (2023) User Experience Design Professionals' Perceptions of Generative Artificial Intelligence, DOI: https://doi.org/10.48550/arXiv.2309.15237.

Lycett M., Marcos E., Storey V. (2007) Model-driven systems development: an introduction. European Journal of Information Systems, 16(4), 346–348. DOI: https://doi.org/10.1057/palgrave.ejis.3000684.

Mahatody T., Ilie M., Rapatsalahy M. A., Dimbisoa W. G., Ilie S. (2021) Metamodel based approach to generate user interface mockup from UML class diagram, Procedia Computer Science, Volume 184, 2021, Pages 779–784, ISSN 1877-0509, DOI: https://doi.org/10.1016/j.procs.2021.03.096.

Marzuoki (el) N. (2021) Model Composition in Multi-Modeling Approaches Based on Model Driven Architecture, PhD Thesis, la Faculte des Sciences Dhar El Maharaz Fes, Marocco.

Mbugua S. T., Korongo J., Samuel M. (2022) On Software Modular Architecture: Concepts, Metrics and Trends, International Journal of Computer and Organization Trends, vol. 12, no. 1, Jan–Apr. 2022, pp. 3–10, DOI: 10.14445/22492593/IJCOT-V12I1P302.

Molina P., Meliá S., Pastor O. (2002) User Interface Conceptual Patterns, 159–172, DOI: http://doi.org/10.1007/3-540-36235-5_12.

Nacheva R. (2017) Prototyping Approach In User Interface Development, 2nd Conference on Innovative Teaching Methods, Bulgaria, 28–29 June, 2017, 80–87.

Narang P., Mittal P. (2022) Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture. Engineering, Technology & Applied Science Research, 12, 9726–9731, DOI: https://doi.org/10.48084/etasr.5315.

Nasiri S., Rhazali Y., Adadi A., Lahmer M. (2023) Generation of User Interfaces and Code from User Stories. In: Joshi, A., Mahmud, M., Ragel, R. G. (eds) Information and Communication Technology for Competitive Strategies (ICTCS 2021). Lecture Notes in Networks and Systems, vol. 400. Springer, Singapore. DOI: https://doi.org/10.1007/978-981-19-0095-2_38.

Newman M. W., Landay J. A. (2000) Sitemaps, storyboards, and specifications: a sketch of Web site design practice. In Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '00). Association for Computing Machinery, New York, NY, USA, 263–274. DOI: https://doi.org/10.1145/347642.347758.

Nguyen T. D., Vu P., Pham H., Nguyen  T. (2018) Deep Learning UI Design Patterns of Mobile Apps, 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER), Gothenburg, Sweden, pp. 65–68, DOI: https://doi.org/10.1145/3183399.3183422.

Nikiforova O., Gusarovs K. (2020) Anemic Domain Model vs Rich Domain Model to Improve the Two-Hemisphere Model-Driven Approach, Applied Computer Systems, 25(1), 51–56, DOI: https://doi.org/10.2478/acss-2020-0006.

Nikiforova O., Kirikova, M. (2004) Two-Hemisphere Model Driven Approach: Engineering Based Software Development. In: Persson, A., Stirna, J. (eds) Advanced Information Systems Engineering. CAiSE 2004. Lecture Notes in Computer Science, vol. 3084. Springer, Berlin, Heidelberg, DOI: https://doi.org/10.1007/978-3-540-25975-6_17.

Nikiforova O., Pavlova, N. (2011) Open Work of Two-Hemisphere Model Transformation Definition into UML Class Diagram in the Context of MDA. In: Software Engineering Techniques: Lecture Notes in Computer Science. Vol. 4980. Berlin: Springer Berlin Heidelberg, pp. 118–130. ISBN 9783642223853.

Nikiforova O. (2002) General Framework for Object-Oriented Software Development Process. Applied computer systems, Vol. 13, pp. 132–144, ISSN 1407-7493.

Nikiforova O. (2009). Two Hemisphere Model Driven Approach for Generation of UML Class Diagram in the Context of MDA. e-Informatica, 3, 59–72, https://www.e-informatyka.pl/attach/e-Informatica_-_Volume_3/eInformatica2009Art4.pdf.

Nikiforova O., Babris K., Kristapsons J. (2020) Survey on Risk Classification in Agile Software Development Projects in Latvia. Applied Computer Systems, Vol. 25, No. 2, pp. 105–116, ISSN 2255-8683, e-ISSN 2255-8691, DOI: https://doi.org/10.2478/acss-2020-0012 (Web of Science).

Nikiforova O., Babris K., Madelāne L. (2021) Expert Survey on Current Trends in Agile, Disciplined and Hybrid Practices for Software Development, Applied Computer Systems, vol. 26, no. 1, 2021, pp. 38–43, DOI: https://doi.org/10.2478/acss-2021-0005 (Web of Science).

Nikiforova O., Babris K., Mahmoudifar F. (2024a) Automated Generation of Web Application Front-End Components from User Interface Mockups, Proceedings of International Conference on Software Technologies (ICSOFT 2024, July 8–10), SCITEPRESS Digital Library, 2024, pp. 100–111, DOI: 10.5220/0012759500003753 (SCOPUS).

Nikiforova O., Babris K., Guliyeva A. (2024b) Definition of a Set of Use Case Patterns for Application Systems: A Prototype-Supported Development Approach. Applied Computer Systems, Vol. 29, No. 1, 2024, pp. 59–67, DOI: 10.2478/acss-2024-0008 (*Web of Science*).

Nikiforova O., El Marzouki N., Gusarovs K., Vangheluwe H., Bures T., Al-Ali R., Iacono M., Orue Esquivel P., and Leon F. (2017) The Two-Hemisphere Modelling Approach to the Composition of Cyber-Physical Systems" – Proceedings of International Conference on Software Technologies (ICSOFT 2017), 24–26 July, 2017, Madrid, Spain. SCITEPRESS Digital Library, pp. 286–293, DOI: https://doi.org/10.5220/0006424902860293.

Nikiforova O., Iacono M., El Marzouki N., Romanovs A. and Vangheluwe H. (2020) Enabling Composition of Cyber-Physical Systems with the Two-Hemisphere Model-Driven Approach, In: Multi-Paradigm Modelling Approaches for Cyber-Physical Systems. B. Tekinerdogan, D. Blouin, H. Vangheluwe, M. Goulão, P. Carreira, V. Amaral ed. 125 London Wall, London EC2Y 5AS, United Kingdom: Academic Press is an imprint of Elsevier, 2020. pp. 149–168, ISBN 978-0-12-819105-7 (Scopus).

Nikiforova O., Kozacenko L. and Ahilcenoka D. (2013) UML Sequence Diagram: Transformation from the Two-Hemisphere Model and Layout, Applied Computer Systems, vol. 14, no. 1, pp. 31–41, DOI: https://doi.org/10.2478/acss-2013-0004.

Nikiforova O., Kozacenko, L., Ahilcenoka, D., Gusarovs, K., Ungurs, D., Jukss, M. (2015) Comparison of the Two-Hemisphere Model-Driven Approach to Other Methods for Model-Driven Software Development, Scientific Journal of Riga Technical University: Applied Computer Systems, 18, 5–14, DOI: http://doi.org/10.1515/acss-2015-0013.

Nikiforova O., Sukovskis U., Gusarovs K. (2015). Application of the two-hemisphere model supported by BrainTool: Football game simulation. AIP Conference Proceedings, 1648, DOI: https://doi.org/10.1063/1.4912557.

Nikiforova O., Zabiniako V., Kornienko J., Rizhko R., Babris K., Gasparoviča-Asīte M. (2021a) Efficiency Monitoring of Engineering System Designer Work Based on Multi-System User Behavior Analysis with AI/ML Algorithms, 2021 IEEE 62nd International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON), 2021, pp. 1–6, DOI: https://doi.org/10.1109/RTUCON53541.2021.9711720 (SCOPUS).

Nikiforova O., Zabiniako V., Kornienko J., Rizhko R., Babris K., Nikulsins V., Garkalns P., Gasparoviča-Asīte M. (2021b) Solution to On-line vs On-site Work Efficiency Analysis on the Example of Engineering System Designer Work, Applied Computer Systems, vol. 26, no. 2, pp. 87–95, DOI: https://doi.org/10.2478/acss-2021-0011 (SCOPUS).

Osis J., Asnina E. (2011) Model-Driven Domain Analysis and Software Development: Architectures and Functions. 1 edition. Hershey – New York, USA: IGI Global, 2011, 514 p. ISBN13: 9781616928742, DOI: http://doi.org/10.4018/978-1-61692-874-2.

Pastor O., Abrahao S., Molina J.C., Torres I. (2001) A FPA-like Measure for Object Oriented Systems from Conceptual Models, Current Trends in Software Measument, Ed. Shaker Verlag, pages 51–69, Montreal, Canada, 2001.

Pavlova N. (2008) Platformneatkarīga modeļa izstrādes pieeja modeļvadāmas arhitektūras ietvarā, promocijas darbs, Rīgas Tehniskā universitāte, 2008.

Pelechano V., Pastor O., Insfrán E. (2002) Automated code generation of dynamic specializations: an approach based on design patterns and formal techniques, Data & Knowledge Engineering, Volume 40, Issue 3, pp. 315–353, ISSN 0169-023X, DOI: https://doi.org/10.1016/S0169-023X(02)00020-4.

Pimentel J., Castro J., Mylopoulos J., Angelopoulos K., Souza V. S. (2014) From requirements to statecharts via design refinement. Proceedings of the ACM Symposium on Applied Computing, DOI: https://doi.org/10.1145/2554850.2555056.

Planas E., Daniel G., Brambilla M. et al. (2021) Towards a model-driven approach for multiexperience AI-based user interfaces. Softw Syst Model 20, 997–1009, DOI: https://doi.org/10.1007/s10270-021-00904-y.

Riaz S., Arshad A., Band S. S., & Mosavi A. (2022) Transforming Hand Drawn Wireframes into Front-End Code with Deep Learning, Computers, Materials & Continua, 72, 4303–4321, DOI: https://doi.org/10.32604/cmc.2022.024819.

Riccio V., Jahangirova G., Stocco A. et al. (2020) Testing machine learning based systems: a systematic mapping. Empir Software Eng 25, 5193–5254, DOI: https://doi.org/10.1007/s10664-020-09881-0.

Rivero J., Rossi G., Grigera J., Luna R. E., Navarro A. (2011) From Interface Mockups to Web Application Models, 6997, 257–264, DOI: http://doi.org/10.1007/978-3-642-24434-6_20.

Rudd J., Stern K., Isensee S. (1996) Low vs. high-fidelity prototyping debate. interactions 3, 1 (Jan. 1996), 76–85, DOI: https://doi.org/10.1145/223500.223514.

Sharp H., Rogers Y., Preece J. (2019) Interaction Design: Beyond Human Computer Interaction (5th Edition), ISBN: 9781119547259, Wiley, 2019.

Silva (da) T. S., Martin A., Maurer F., Silveira M. (2011) User-Centered Design and Agile Methods: A Systematic Review, 2011 Agile Conference, 2011, pp. 77–86, DOI: https://doi.org/10.1109/AGILE.2011.24.

Stige Å., Zamani E. D., Mikalef P., Zhu Y. (2023) Artificial intelligence (AI) for user experience (UX) design: a systematic literature review and future research agenda, Information Technology & People, DOI: https://doi.org/10.1108/ITP-07-2022-0519.

Stompff G., Smulders F. (2015) The Right Fidelity: Representations That Speed Up Innovation Processes. Design Manag J, 10, 14–26, DOI: https://doi.org/10.1111/dmj.12019.

Suleri S., Pandian V. P. S., Shishkovets S., Jarke M. (2019) Eve: A Sketch-based Software Prototyping Workbench. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA '19). Association for Computing Machinery, New York, NY, USA, Paper LBW1410, 1–6, DOI: https://doi.org/10.1145/3290607.3312994.

Sunitha E. V., Samuel P. (2019) Automatic Code Generation From UML State Chart Diagrams, in IEEE Access, vol. 7, pp. 8591–8608, 2019, DOI: https://doi.org/10.1109/ACCESS.2018.2890791.

Thomas D. (2004) MDA: Revenge of the Modelers or UML Utopia? IEEE Softw. 21, 3 (May 2004), 15–17, DOI: https://doi.org/10.1109/MS.2004.1293067.

Trehan V., Chapman C., Raju P. (2015) Informal and formal modelling of engineering processes for design automation using knowledge based engineering. J. Zhejiang Univ. Sci. A 16, 706–723, DOI: https://doi.org/10.1631/jzus.A1500140.

Uyanık B., Sayar A. (2024) Analysis and Comparison of Automatic Code Generation and Transformation Techniques on Low-Code Platforms. In Proceedings of the 2023 5th International Conference on Software Engineering and Development (ICSED '23). Association for Computing Machinery, New York, NY, USA, 17–27, DOI: https://doi-org.resursi.rtu.lv/10.1145/3637792.3637795.

Virzi R. A., Sokolov J. L., Karis D. (1996) Usability problem identification using both low- and high-fidelity prototypes. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '96). Association for Computing Machinery, New York, NY, USA, 236–243, DOI: https://doi.org/10.1145/238386.238516.

Völter M., Bettin J. (2004) Patterns for Model-Driven Software-Development. Proceedings of the 9th European Conference on Pattern Languages of Programms (EuroPLoP '2004), Irsee, Germany, July 7-11, 2004, 525–560.

Wellner P. D. (1989) Statemaster: A UIMS based on statecharts for prototyping and target implementation. SIGCHI Bull. 20, SI (March 1989), 177–182, DOI: https://doi.org/10.1145/67450.67486.

Yigitbas E., Jovanovikj I., Biermeier K. et al. (2020) Integrated model-driven development of self-adaptive user interfaces. Softw Syst Model 19, 1057–1081, DOI: https://doi.org/10.1007/s10270-020-00777-7.

**Kristaps Babris** was born in Riga in 1986. He received a Bachelor's degree (2015) and a Master's degree (2018) in Computer Systems from Riga Technical University with the qualification of programming engineer. Since 2012, he has been working at "IT Sapiens", holding the position of Head of the IT Department, and since 2016 – at Riga Technical University holding position of scientific assistant. Scientific interests include application of model-driven engineering principles to Web system development.