TECHNISCHE
UNIVERSITÄT
DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair for Computer Networks

Master Thesis

Support of Lecturers in Modeling Complex Learning and Teaching Scenarios in Audience Response Systems

Sinthujan Thanabalasingam

Born on: 7th November 1993 in Hachenburg

to achieve the academic degree

Master of Science (M.Sc.)

Supervisors

Dr.-Ing. Iris Braun

Tommy Kubica, M.Sc.

Supervising professor

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Submitted on: 25th February 2021

Acknowledgements

I would like to thank everyone who supported and motivated me during the preparation of this master thesis. First and foremost, I would like to thank Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill, who made it possible for me to write this master thesis. For the supervision of the thesis, the helpful suggestions and constructive criticism during the writing of this thesis I would like to thank Tommy Kubica M.Sc. and Dr.-Ing. Iris Braun. A big thank you also goes to the participants of the two evaluation rounds, without whose feedback this thesis would not have been possible. Furthermore, I would like to thank my girlfriend, my brother and my two cats, who gave me moral support especially during the Covid-19 pandemic that raged while I was writing this thesis.

TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät Informatik Institut für Systemarchitektur Professur Rechnernetze

AUFGABENSTELLUNG FÜR DIE MASTER-ARBEIT

THEMA: Unterstützung des Lehrenden bei der Modellierung komplexer Lernszenarien in Audience Response Systemen

Name:	Thanabalasingam, Sinthujan	Studiengang:	Master Informatik
Matrikel-Nr.:	3946479	Projekt/Fokus:	stARS/TEL
verantwortlicher Hochschullehrer:	Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill		
Betreuer*in:	Tommy Kubica, M.Sc., Dr.-Ing. Iris Braun		
Beginn am:	17.07.2020	einreichen bis:	18.12.2020

ZIELSTELLUNG

Am Lehrstuhl Rechnernetze wurden in den vergangenen Jahren mehrere prototypische Untersuchungen zum Einsatz technischer Werkzeuge im Lehrbetrieb durchgeführt. Unter anderem wurden Werkzeuge in Vorlesungen und Übungen getestet; es wurden für den jeweiligen Einsatz valide Ergebnisse gewonnen.

Zur Verwendung technischer Werkzeuge in beliebigen Anwendungsszenarien, die im Lehrbetrieb auftreten, wird ein kontext-sensitives System angestrebt. Im ersten Schritt wurde ein Metamodell entwickelt, dass in der Lage ist, diese Anwendungsszenarien zu beschreiben. Im zweiten Schritt wurde ein graphischer Editor konzipiert und prototypisch implementiert, der den Benutzern des Systems, spezieller gesehen Lehrenden mit unterschiedlichen Modellierungskenntnissen, ermöglicht, individuelle Abläufe, sogenannte Anwendermodelle, auf Grundlage des Metamodells zu modellieren. Während die Modellierung von einfachen Anwendermodellen ohne großen Aufwand verbunden ist, stellt sich die Modellierung von komplexen Anwendermodellen, wie sie in realen Szenarien auftreten können, als aufwendig heraus.

Im Rahmen dieser Masterarbeit soll diese Problemstellung vertieft untersucht werden. Dazu muss im ersten Schritt der aktuelle Stand des graphischen Editors analysiert und Problemstellen, die den Lehrenden an der Modellierung komplexer Szenarien hindern, aufgedeckt werden. Danach sollen Anforderungen zur Lösung der gefundenen Problemstellen aufgestellt werden. Anschließend müssen verwandte Arbeiten aus den Bereichen Workflow-Modellierung und Grafikeditoren nach Lösungskonzepten untersucht werden, bevor das vorhandene Konzept des grafischen Editors um ein Konzept zur Modellierung komplexer Szenarien erweitert wird. Das entstandene Konzept soll prototypisch implementiert und anschließend auf seine Intuitivität und Wirksamkeit evaluiert werden.

A. Schill

Digital unterschrieben von
Alexander Schill
Datum: 2020.07.17 12:01:02
+02'00'

Prof. Dr. A. Schill (betreuender Hochschullehrer)

SCHWERPUNKTE

- Analyse des aktuellen Editor-Konzepts sowie der Implementierung,
- Definieren von Anforderungen und Bewertungskriterien zur Anforderungserfüllung,
- Untersuchung verwandter Arbeiten aus dem aktuellen Stand der Technik,
- Erweiterung des Konzepts um die Modellierung komplexer Szenarien,
- Proof-of-Concept Implementierung,
- Evaluation und Auswertung der Ergebnisse.

TECHNISCHE UNIVERSITÄT DRESDEN

EINGEGANGEN 23. NOV. 2020

Fakultät Informatik Prüfungsamt

Antrag auf Verlängerung der Bearbeitungszeit der Masterarbeit (ab PO 2010)

Studiengang:	Master Informatik (2010)		
Name:	Thanabalasingam	Vorname:	Sinthujan
Matrikelnummer:	3946479	Studienjahrgang:	2013
Betreuender HSL:	Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill		
Beginn:	17.07.2020	Abgabe:	18.12.2020
Dauer der Verlängerung *(max. 13 Wochen)*:	10		
Neuer Abgabetermin:	26.02.2020		

Begründung der Verlängerung:

Sehr geehrte Damen und Herren,
durch die Corona-Pandemie hat sich der Start der Implementationsphase nach hinten verlagert, da eine hierzu benötigte Vorevaluation deutlich mehr Zeit in Anspruch genommen hat als geplant. Es mussten zusätzliche Vorkehrungen getroffen werden um die Evaluation abzuhalten. Neben technischen Problemen während der Evaluation gab es auch Verzögerungen durch Absagen von Probanden. Da nach der Implementationsphase eine weitere abschließende Evaluation geplant und notwendig ist, welche angesichts der allgegenwärtigen Situation ebenfalls online abgehalten werden muss, ist auch hier mit Verzögerungen zu rechnen. Daher wäre ich Ihnen sehr dankbar, wenn Sie die Verlängerung der Masterarbeit genehmnigen könnten.

18.11.2020, S. [illegible]
Datum, Unterschrift Studierende/r

i. A. Braun
Zustimmung betreuender HSL

Dieser Antrag ist mit einer Kopie der Anmeldung zur Masterarbeit fristgerecht im Prüfungsamt vorzulegen.

23. NOV. 2020
Datum

Technische Universität Dresden
Fakultät Informatik
Prüfungsamt
01062 DRESDEN
[illegible]
Unterschrift Prüfungsamt

Entscheidung des Prüfungsausschusses:

Dem Antrag auf Verlängerung der Masterarbeit wird stattgegeben / ~~nicht stattgegeben~~.

02. Dez. 2020
Datum

[illegible]
Unterschrift Prüfungsausschuss

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair for Computer Networks

Abstract

The use of digital technology in education has become increasingly common over the last decades. Audience Response Systems (*ARS*) represent a class of digital tools that aim at improving the communication between lecturers and their audience. By leveraging the increasing popularity of smartphones and other similar mobile devices, several ARS were created that support and enhance individual teaching strategies. However, most of the systems currently available lack the level of customization needed and expected by lecturers.

Currently, only a few static functions are provided by the majority of systems, causing lecturers to adapt their strategy to comply with the system. However, highly situational learning scenarios such as Peer Instruction, Just-in-Time-Learning, or the Jigsaw-Puzzle are not supported by most available systems. *stARS* is an ARS that relies on a scenario- and model-based approach to lectures and teaching strategies. *stARS* gives lecturers more flexibility with regards to when and how exactly functions of the ARS are used. The system provides lectures with a graphical editor, allowing the creation of models of the lectures' schedule. These models are based on a highly expressive meta-model that was designed with the objective of providing maximum flexibility to end-users. While the creation of simple models with *stARS* is working as expected, building more complex models and therefore exploiting the full potential of the meta-model represents a challenge to users due to the prototypical state of its graphical editor. The editor's capabilities are limited when it comes to creating and managing models for complete lectures, as resulting models tend to get structurally complex and big in size.

This work aims at continuing and improving on the ideas of *stARS* by extending the functionality of the web editor to ease the creation of more complex learning scenarios. By using techniques from the domain of user-centered design (*UCD*) and by investigating solutions provided by other graphical modeling tools, a concept for such an extension is designed and evaluated by potential end-users. The final result of this work is an implementation of an expansion for the *stARS* web-editor that eases the creation of highly-customized and potentially complex models of learning scenarios.

TECHNISCHE UNIVERSITÄT DRESDEN

Statement of Authorship

I hereby certify that I have authored this Master Thesis entitled

"Support of Lecturers in Modeling Complex Learning and Teaching Scenarios in Audience Response Systems"

independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. In the selection and evaluation of the material and in the production of the manuscript, I received assistance from the following persons:

Dr.-Ing. Iris Braun, Tommy Kubica, M.Sc.

There were no additional persons involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, 25th February 2021

[illegible]

Sinthujan Thanabalasingam

Contents

List of Figures

List of Tables

1. Introduction

Figure 1.1.: Laptops and other mobile devices are a common sight in lecture halls [1].

1.1. Motivation

Active learning techniques continue to improve over time as new technologies become available. Where students once raised their hands or colored flashcards or provided feedback via student response systems ("clickers"), the use of digital technology has now become a fixture in many lecture halls and classrooms around the world. Motivated by the widespread use of mobile devices such as smartphones and tablets [2], more and more lectures are being delivered with the support of **Audience Response Systems** (ARS) [3]. Several ARS have now been designed with a variety of useful functions and an increasing number of educational institutions are incorporating this new technology into their infrastructure. The use of an ARS is primarily motivated by the benefits that such systems bring to the typical classroom scenario. These benefits include improved audience attentiveness and engagement levels [4], better audience-lecturer communication, anonymous student participation without the fear of giving a wrong answer or asking a wrong question [5], and collection of feedback data that can be evaluated by lecturers. Looking at ARS from the lecturer's perspective, an ARS should complement and support the way the lecturer teaches. However, most of the tools available have weaknesses in terms of individual approaches and strategies for teaching. Many tools neglect the importance of established, individual teaching strategies that lecturers may have. Some may have a pop quiz at the beginning, middle, and end of each lecture, while others may include several group discussions depending on the results to an introductory question.

Often, however, instructors are constrained by the few options offered by the tools they use to design the flow of their lectures. Instead of implementing their individual teaching strategies with the support of an ARS, lecturers feel forced to adapt their strategy to the limited capabilities of the system. Contrary to expectations, only static building blocks and procedures are provided, offering little flexibility or adaptability. A more instructor-oriented ARS should offer more tools for modeling lectures in terms of individual teaching strategies.

Instead of handling lectures as static blocks, it could be beneficial to regard them as dynamic procedures or **workflows** that can be modeled with the help of software. This is the approach that ***stARS***[1] (**s**cenario-**t**ailored **A**udience **R**esponse **S**ystem), a new web-based ARS currently under development at the Technische Universität Dresden [6], is exploring. As opposed to most other solutions, *stARS* focuses on offering more flexibility by enabling the creation and supported execution of custom *learning scenarios*. The scenarios define the general structure and schedule of lectures and regulate, how and when features of ARS (such as quizzes and polls, group formations or audience feedback) are supposed to support an ongoing lecture. For this purpose, a meta-model was developed that formally defines a modelling language for the most common features that occur in many ARS implementations [7].

Additionally, a graphical **web editor** was conceived in [8] and implemented in a prototypical manner. It intends to enable lecturers, regardless of technical background or experience in using modeling software, to model their lectures and to see them through the lens of workflows. Workflows are extensible, customizable, easy to understand, and (when paired with the meta-model), their high level of expressiveness allows them to resemble individual teaching strategies more closely. Supporting lecturers by offering them an adequate and properly engineered web editor will enable them to create their own, customized learning scenarios. The benefits from which lecturers may profit include:

[1]The project can be accessed via `https://stars-project.com`. last successful access: 2021-02-25

1. Easy **creation**, **(re-)use**, **editing** and **sharing** of learning scenarios
2. Reduced **system intrusion** through reduction of workarounds
3. Expression of highly **customized**, **situational** and/or **complex** scenarios
4. Viable usage of **known, established didactic strategies** and
5. Overall **increased effectiveness** of the ARS and its functionality

1.2. Objectives

More sophisticated didactic strategies and scenarios can become convoluted or require features that the majority of existing ARS do not support. The structural complexity of some scenarios becomes clear when looking at them as workflows that should be modeled by software. Some mentionable examples that are further elaborated on in Chapter 2 include

- **Peer Instruction** [9], a teaching strategy developed by Prof. Eric Mazur of Harvard University that involves one or more **ConcepTests** and **Peer Discussions**,
- the **Jigsaw Teaching Strategy** [10] that involves splitting students into groups and having members of the groups each work on a separate topic, and
- **Collaborative Learning**, a strategy that consists of the three phases **Think**, **Pair** and **Share**.

Several individuals at the Technische Universität Dresden recognized the potential of an ARS that fully supports strategies as the ones mentioned before. An enormous amount of preliminary work was done to push the *stARS* project forward. With the introduction of both the **meta-model** for *stARS* and its **web editor** front-end application, a solid foundation was created to further expand on and explore the possibilities that derive from workflow-based ARS usage.

On the one side, the meta-model of *stARS* has reached a vastly refined state and grants high expressiveness, as models for big scenarios with hierarchical structures can be created with it. However, on the other side, the web editor makes it difficult for lecturers to fully take advantage of the possibilities that the meta-model offers. At the time of writing, the web editor remains in a prototypical state with several drawbacks that make the creation of complex workflows and scenarios unnecessarily complicated, while the creation of small, simple workflows involves only little effort.

Peer Instruction[2] is a suitable example that illustrates the drawbacks the current version of the web editor has. The *stARS* web editor generally supports the creation of a model that resembles this didactic strategy. However, the modeling process is small-scale and time-consuming, especially for inexperienced users. Each model element has to be created and modified individually which increases both the time needed and the likelihood to make a mistake.

Furthermore, there is no easy way to use structures that were already modeled in the past. This means that it is required to model Peer Instruction multiple times if it is intended to be used with different pop-quizzes, questions, and answers. There exists no straight forward way to copy, import, or otherwise embed an existing structure to a new model. Complicated workflows therefore have to be re-created on re-use.

In summary, it can be said that the current state of the web editor serves as an ideal starting point, allowing the introduction of new concepts that make it easier to create, manage, reuse, embed, and share more sophisticated scenarios.

The main goal of this work is to **simplify the modeling process for complex scenarios**. When using *stARS*, lecturers should be able to use the full expressiveness of the meta-model and benefit from improved support for existing popular didactic strategies. To accomplish this objective, this work intends to extend the *stARS* web editor by features that concern the following aspects:

- Component **Reuse**,
- Component **Sharing**,
- Support for **Nested Components**, and
- Assistance for creating **Connected Components**

[2]There exist several variations of Peer Instruction that vary in their structure.

1.3. Structure

The rest of this work is organized as follows: In Chapter 2, the fundamentals required to understand the remainder of this work are elaborated on. Specifically, relevant information to the terms *Audience Response System*, *Workflow Modeling* and *Graphical Modeling Tools* is provided, before presenting additional background information regarding *Didactic Strategies* and the field of *User Interface Engineering*.

After covering the fundamentals, Chapter 3 proceeds to evaluate the current state of the art in the context of ARS. This is done both to understand how other systems work and to document and analyze the current version of *stARS* and its web editor, focusing primarily on its problems and drawbacks when used to model complex learning scenarios. This is then followed by a requirements analysis that presents the results of this chapter and summarizes criteria for improving the current version of the web editor.

These results form the basis for Chapter 4, in which a concept for improving the support of modeling complex learning scenarios with *stARS* is presented. During the concept development phase of this work, several mock-ups and drafts for user interfaces (intended to be integrated into the web editor) were created. The main focus of this chapter therefore lays on presenting these solutions and to elaborate on how these solutions were iteratively improved and evaluated.

Having developed and refined the concept for extending the *stARS* web editor, the implementation phase followed. This phase had the goal to realize the concept in a prototypical manner. Chapter 5 will describe the development process by providing background information to specific details of the implementation, necessary workarounds, issues and problems. Additionally, this chapter will also describe how several adaptations and changes to the concept itself were necessary to deliver a functioning prototype that can be evaluated.

After the implementation phase was concluded, the extension of the web editor had to be evaluated. In this phase, real users tested the enhanced version of the web editor with regards to user expectations and intuitiveness of the new features. The evaluation process and its results will be described in Chapter 6.

Finally, this work is concluded by Chapter 7, in which the findings of this thesis are summarized and an outlook on future work regarding *stARS* and the web editor is provided.

2. Fundamentals

Chapter 2: Fundamentals

This chapter will provide more context and background information, serving as a glossary for fundamental technicalities that the remainder of this work relies on. First, an overview and definition for the terms *Audience Response System*, *Workflow Modeling* and *Graphical Modeling Tool* will be given. Next, different examples for *Didactic Strategies* intended to be supported by the *stARS* project will be elaborated on. Finally, for the reason that a major part of this work deals with the design of user interfaces, the concluding section of this chapter will briefly introduce relevant terms from the field of *User Interface Engineering*.

2.1. Audience Response Systems

The difficult task of institutions such as schools and universities is to provide educational services in the most effective way possible. Typically, a teaching scenario consists of a single person – the lecturer – on one side and a multitude of people – the audience – on the other. In many cases, a uni-directional communication takes place, namely the presentation of instructional material by the lecturer to the audience. Depending on various factors of the learning environment (such as the size of the audience, the size of the lecture hall, or the variable attention level of the audience), both lecturer and audience may feel disconnected from each other. A lecturer may lose the audience by speaking too quietly or too quickly, or by not explaining difficult topics thoroughly enough. Because of these problems, the audience may have difficulty following the lecturer. As a result, ambiguity and questions arise in the audience. However, with a larger audience, it becomes especially difficult for individuals to communicate a question or feedback to the lecturer, which ultimately leads to a decline in audience attention.

Several various feedback systems that address these issues are constantly developed and improved over time as new technology becomes available. Some of the first feedback systems were very basic and had students raise their hands or use colored flashcards to give feedback to the lecturer. Other systems had students push a button on a "clicker" to participate in polls during lectures. Nowadays, digital ARS are an increasingly popular technology in education and teaching. More specifically, an ARS can be described as a set of supportive technical tools that mainly provide the audience with capabilities to *respond* and communicate with the lecturer and vice versa. Using an ARS enables the establish-

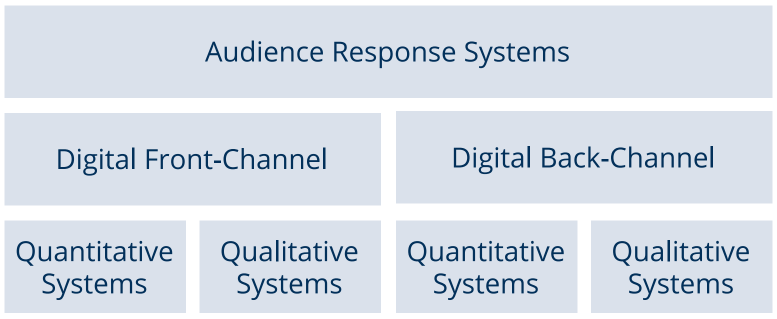


Figure 2.1.: Overview of different forms of Audience Response Systems [11] (translated into English).

ment of a tighter feedback loop between the lecturer and the audience, increasing both the audience's attention and level of activity, and the lecture's options to react on audience demands. **Web-based** ARS have gained popularity over the last decade as a result of being platform-independent. One of the reasons for this development is the increasing presence of mobile end devices that, instead of being banned from lecture halls, were embraced by many ARS. Accessing diverse functions that are offered by a web-based ARS is easy, as most of the audience members already carry the necessary hardware, complete with a pre-installed web-browser with them when they enter the classroom. Therefore, web-based ARS represent the most accessible and cost-effective solutions. Consequently, many ARS depend on the presence of end devices in the audience to function properly. However, several classes of systems with different ways and directions of interaction were conceived. In [11], a classification of ARS into **Front-Channel-Systems** and **Back-Channel-Systems** (see Figure 2.1) is proposed that will be elaborated on in the following.

2.1.1. Digital Front-Channel

Front-Channel Systems play an important role in question-oriented teaching strategies. These systems typically require lecturers to interrupt their lectures at specific points in time to make use of the system's features. At a certain moment during the lecture, the lecturer pauses the topic presentation and the system is used to ask the audience questions that can be answered and evaluated in real-time. This illustrates the most common form of functionality provided by the majority of ARS, as most of them offer similar Front-Channel features. Digital Front-Channel Systems can be further classified into **qualitative** and **quantitative** systems.

Qualitative Front-Channel Systems Qualitative Front-Channel Systems allow students to freely formulate feedback, for example to answer questions. The system does not provide predefined answers or other choices for feedback.

Quantitative Front-Channel Systems Quantitative Front-Channel Systems provide predefined feedback options. Therefore, lecturers can only make quantitative assumptions about the collected feedback. An example of this would be a multiple-choice question with several answer options.

2.1.2. Digital Back-Channel

In contrast to Front-Channel Systems, Back-Channel Systems operate silently in the background without major intrusions to an ongoing lecture. This means that the lecture is not periodically interrupted at certain intervals by the system itself. Instead, the audience uses the system to provide feedback while the lecture takes place. By analyzing the data collected from the audience, lecturers can adapt different aspects of their presentation style such as speed or the volume of their voice. Similar to Front-Channel Systems, a distinction is made between **qualitative** and **quantitative** systems.

Qualitative Back-Channel Systems Qualitative Back-Channel Systems offer textual feedback mechanisms to members of the audience. These mechanisms allow individual listeners to either address the lecturer directly or to share questions that might concern the whole audience.

Quantitative Back-Channel Systems Quantitative Back-Channel Systems offer predefined answer choices instead of free-text feedback. Therefore, lecturers can only make quantitative assumptions about the collected feedback. An example of a feedback mechanism with predefined choices are questions with Likert scales.

2.2. Workflow Modeling

The introduction of this work stated that *stARS* introduces the idea of models for didactic scenarios. A multitude of definitions and classifications for the term **model**, which vary according to discipline, are known in the literature (for example [12], [13], [14] and [15]). In general, a model is an abstraction of an object or a structure in the real world. Although the abstraction of objects almost always means losing details, it is much easier to work with simplified objects in software. In computer science, abstraction by the usage of models is the standard way of representing real-world references in a digital manner. In the context of this work however, the main object of interest will be the **lectures** themselves, more specifically the structure of the lectures that will be abstracted by and modeled with the help of **workflows** (see Section 2.2.2).

2.2.1. Meta-model

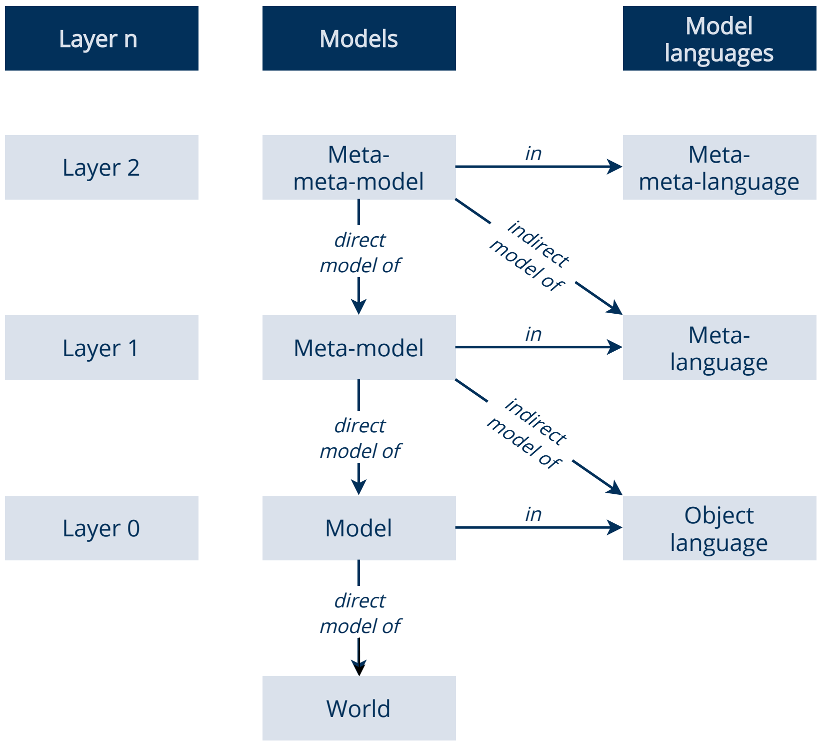


Figure 2.2.: Definition of meta-models based on meta-model languages [16].

Objects of the real world are not the only subject of abstraction – **meta-modeling** describes the act of constructing a descriptive model of another model. A more precise definition by *Strahringer* says:

"According to the language-based meta-modeling concept, a model is a meta-model for another model, if there is a descriptive model of the language in which this model is formulated" [16].

In fact, each abstraction layer n can be abstracted further by introducing another layer $n+1$ above it (see Figure 2.2). In [7], a meta-model was created that describes the elements and functionalities that most ARS have in common. This meta-model resides on layer $n = 1$. Therefore, it allows for the creation of models (consequently residing on layer $n = 0$) of the vast majority of imaginable lecture scenarios. The models represent an abstraction of the lecture that takes place in the real world (on layer $n = -1$). The connection between the meta-model, a modeled didactic schedule, and a live lecture that takes place is illustrated in Figure 2.3.

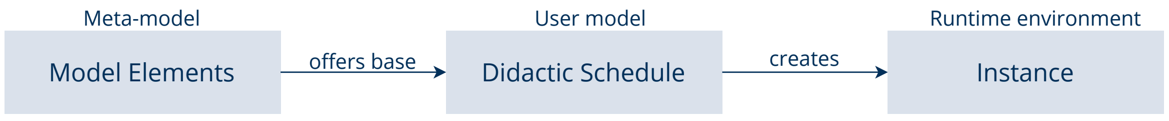


Figure 2.3.: Connection between model elements, a modeled didactic scenario and a live instance [7].

2.2.2. Workflows and Scenarios

Many lecturers plan the schedule of their lecture (e.g. moments when to ask questions, present a specific topic or when to do a group discussion) carefully. This is especially important when a *Digital Front-Channel System* is used. As described earlier, *Digital Front-Channel Systems* rely on interruptions of the lecturer's talk during which the system finds usage. If

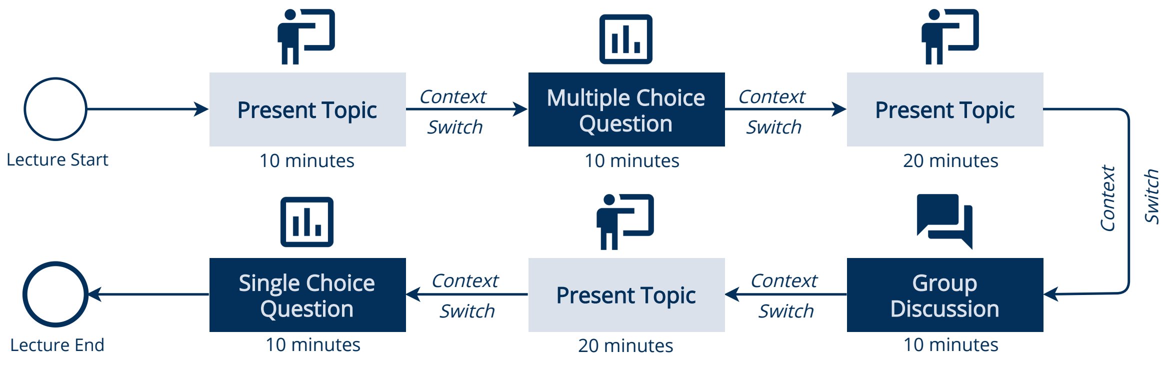


Figure 2.4.: An example of a lecture scenario supported by a Digital Front-Channel system visualized as a workflow. A *Context Switch* occurs whenever usage of the ARS starts or stops.

viewed from a more abstract perspective, a lecture **scenario** (involving a lecturer, his/her audience and the use of an ARS) represents a *sequence of interdependent processes or tasks*. This description is similar to how the term **workflow** is often defined: Workflows are a widely-used abstraction for simulations, data analysis, business processes, scientific computations, and other structures composed of interdependent steps [17]. The tasks that make up a lecture scenario typically involve either the presentation of teaching material by the lecturer or the utilization of other features provided by the ARS (see Figure 2.4). Hence, workflows represent a metaphor suitable for the abstraction and modeling of lecture scenarios. The terms *workflow*, *scenario* and *model* will therefore be used interchangeably in the remainder of this work.

2.3. Graphical Modeling Tools

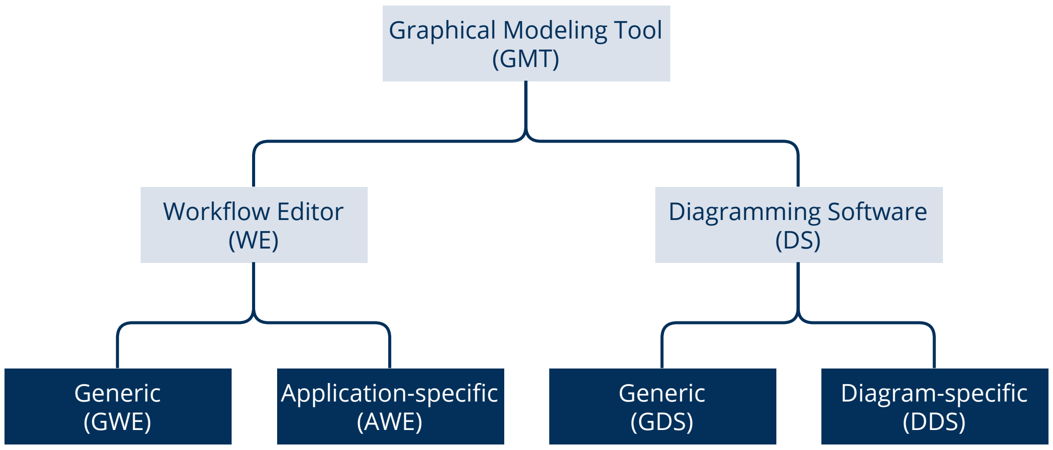

Graphical Modeling Tool (GMT)
Workflow Editor (WE)
Diagramming Software (DS)
Generic (GWE)
Application-specific (AWE)
Generic (GDS)
Diagram-specific (DDS)

Figure 2.5.: Classification of Graphical Modeling Tools. The terms *GWE*, *AWE*, *GDS* and *DDS* are introduced by this work to make categorization of tools related to the *stARS* web editor easier.

To understand how the *stARS* web editor can be improved to ease the creation of complex learning scenarios, it seems plausible to investigate how related tools solve certain problems and shortcomings. For this, a custom hierarchy, by which related tools can be classified, is introduced (see Figure 2.5). This hierarchy was developed during the state of the art research phase of this work which will be covered in Chapter 3. The classification puts the *stARS* web editor more into context and relates it to other modeling tools. First, the term **Graphical Modeling Tool** is defined as follows:

A **Graphical Modeling Tool** (GMT) is a piece of software that allows for the creation and manipulation of models. GMTs can be characterized by supported modeling languages, graphical representations of model elements, application domain, the kinds of artifacts one can create with the tool, and by provided tools that allow the manipulation of model elements.

The classification splits GMTs into two categories: **Workflow Editors** (WE) and **Diagramming Software** (DS).

2.3.1. Workflow Editors

On the one hand, *Workflow Editors* are further differentiated in the following way:

Generic Workflow Editor

A **Generic Workflow Editor** (*GWE*) is a software tool that allows to create and manipulate workflow models that are independent of an application-specific domain. This means that the provided modeling language and its elements are generic (e.g. they do not represent types that live in a given domain). Typically, one or more modeling languages are supported. The artifacts created by these tools are full-fledged models that

- can be persisted by a **textual representation**,
- can be **validated** against the underlying rules of the modeling notation used,
- can be **processed** by other software such as workflow-engines.

Application-specific Workflow Editor

An **Application-specific Workflow Editor** (*AWE*) is a software tool that *only* allows to create and manipulate workflow models that live in an application-specific domain. These tools have the same properties as *GWEs*, except that typically, only one modeling language (often customized to better suit the application domain) is supported. An example for this would be the *stARS* web editor, as it allows to create workflows that live in the domain of learning scenarios. The workflows are graphically represented by the only supported, custom modeling language that will be referred to as ***stARS*-Modeling Notation** (*stARS-MN*) for the remainder of this work.

2.3.2. Diagramming Software

On the other hand, the category *Diagramming Software* is further differentiated in the following way:

Generic Diagramming Software

The term **Generic Diagramming Software** (*GDS*) describes any software tool that is designed for the creation of diagrams of any kind. Typically, many sets of icons and shapes from different modeling languages are supported. However, these tools differ from *WEs*, as artifacts created with these tools do not represent input for other software (e.g. a validation tool or a workflow-engine). Instead, these artifacts are only visual models that illustrate certain things such as relations between different objects or complex processes.

Diagram-specific Diagramming Software

The category of **Diagram-specific Diagramming Software** (*DDS*) describes any software tool that allows for the creation of diagrams of a specific type *only*. Just one modeling language/set of modeling elements is supported.

2.3.3. Graphical Workflow Representations

Workflow Modeling Tools depend on the graphical representation of models to function properly. In many cases, the graphical representation determines the features that the modeling tool has to support. There exist several graphical representations that can be used to visualize workflows. One example is a static representation in form of a directed, acyclic graph (DAG) that is sufficient to visualize and understand a complex sequence of interdependent tasks [17]. DAGs usually consist of *nodes* (depicted as circles with a textual label) and transitions between them, hence their expressiveness is rather limited (see Figure 2.6). *Flow Charts* are another simple yet expressive type of diagram that is easy to

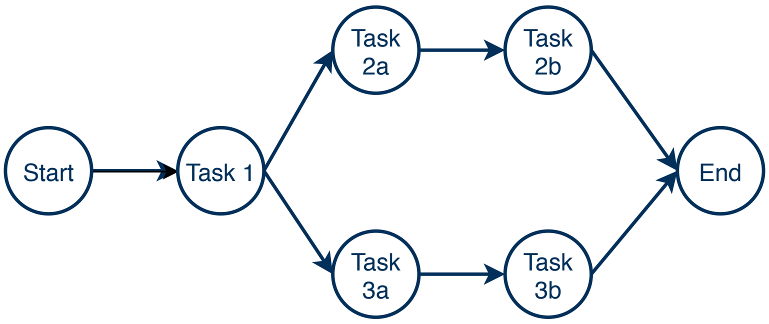


Figure 2.6.: Example of a directed acyclic graph (DAG).

understand. Traditionally, flow charts were and still are used to describe the execution flow of computer programs and pseudo-code (see Figure 2.7). Compared to DAGs, flow charts can be more expressive, as their graphical notation consists of operations, transitions, and an additional symbol for conditional branching [18].

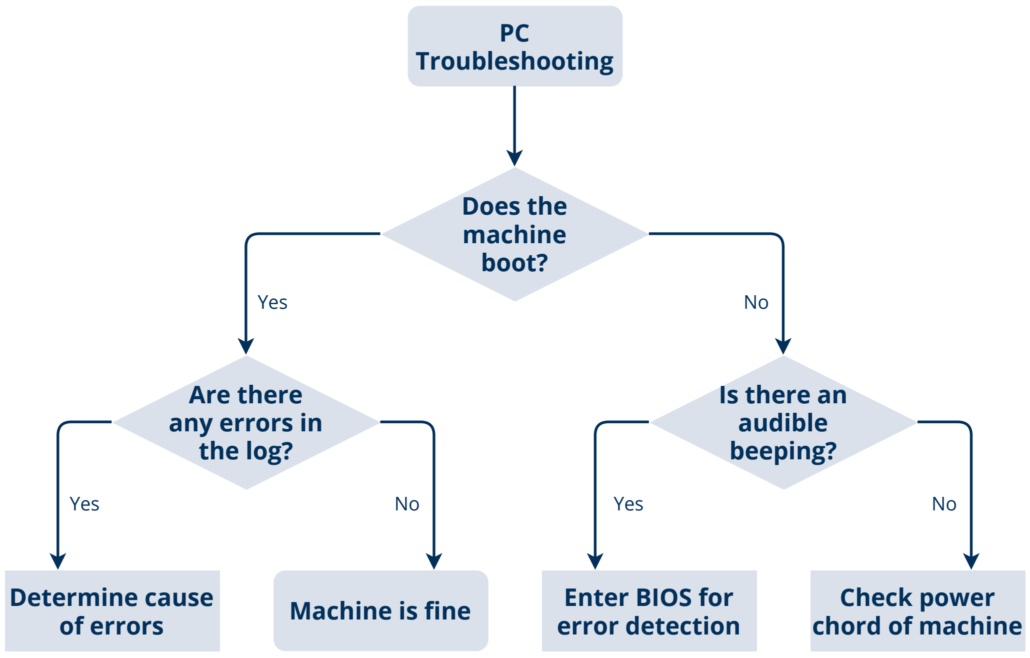

PC Troubleshooting
Does the machine boot?
Yes
No
Are there any errors in the log?
Is there an audible beeping?
Yes
No
Yes
No
Determine cause of errors
Machine is fine
Enter BIOS for error detection
Check power chord of machine

Figure 2.7.: Example of a Flow Chart.

Business Process Model and Notation

One reason that lead to the development of more sophisticated graphical notations was the need to describe increasingly more complex things. A feature-rich graphical notation commonly used for modeling and visualizing business processes and workflows is the **Business Process Model and Notation** (BPMN). BPMN describes a method and the corresponding symbolism for the graphical representation of the individual steps of a planned business process. The current version of the specification (*BPMN 2.0*) offers a diverse set of graphical modeling elements, enabling the creation of expressive models. The following paragraphs will elaborate on the features of BPMN that are important for this work, however further technical information and details on BPMN 2.0 can be found in [19].

The most fundamental notation elements of BPMN are **Activities** that represent a task, **Sequence Flows** that connect different tasks with each other and **Gateways** that merge several control flows. Furthermore, **Nested Tasks** or activities that consist of several individual sub-tasks can be modeled by using **subprocesses** (see Figure 2.8).

BPMN plays an important role for this work, as ***stARS-MN*** (see Section 3.3.2), the customized graphical representation used in *stARS* is heavily influenced by several notation features of BPMN 2.0. A summary of these features of BPMN 2.0 can be found in Table 2.1.

Besides the modeling elements mentioned here, some additional notation elements such as **Pools** and **Lanes** exist that are used to model collaborative processes between multiple participants of a business process. However, these elements are not covered here as they fall out of the scope of this work. Further information about Pools and Lanes can be found in [20].

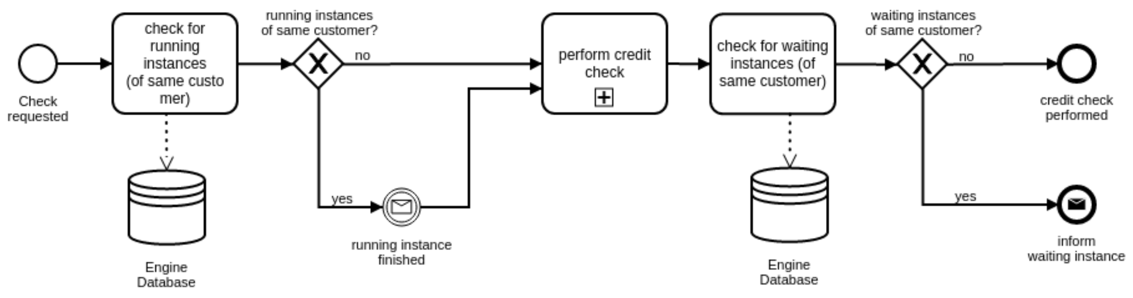

Check requested
check for running instances (of same customer)
Engine Database
running instances of same customer?
no
yes
running instance finished
perform credit check
check for waiting instances (of same customer)
Engine Database
waiting instances of same customer?
no
yes
credit check performed
inform waiting instance

Figure 2.8.: Example of a Business Process modelled with BPMN [21].

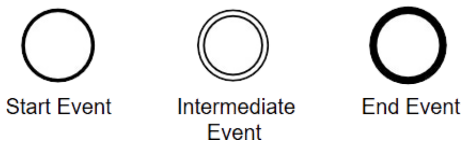
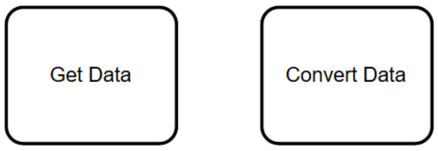
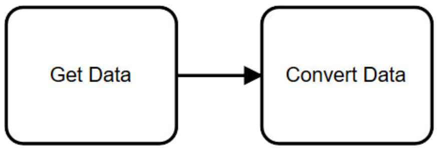
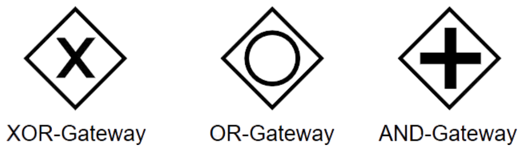
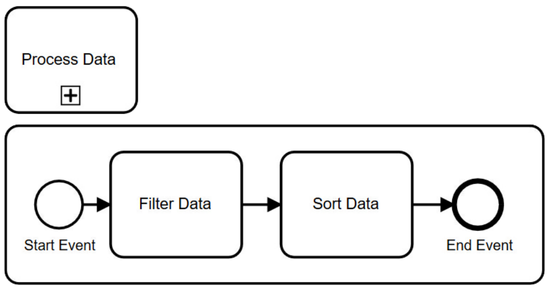
Category	Description
Events 	An **Event** represents something that can happen during a business process, for example the arrival of a message, the reaching of a certain date or the occurrence of an exceptional situation. The **Start Event** and **End Event**, represent the start and end of a process respectively, whereas an **Intermediate Event** is any event that occurs between a start and an end event.
Activities 	An **Activity** describes a task to be performed in a business process. It is represented as a rectangle with rounded corners. An elementary activity is also called **Task**. In the didactic context, an activity can be the presentation of a topic or a group discussion.
Sequence Flows 	**Sequence Flows** are typically depicted as arrows between two model elements. They represent the connections between activities, gateways and events and determine the order in which activities are executed.
Gateways 	A **Gateway** represents a decision point (split/fork) or a point where different control flows converge (join/merge). Gateways can serve either as an **AND**-, an **OR**- or an **XOR**-gateway depending on the symbol used inside the rhombus. Other symbols within the square can be used for event-based or complex gateways.
Subprocesses  Process Data Start Event Filter Data Sort Data End Event	A more complex activity consisting of several individual activities is the **Subprocess**. It is visualized like an Activity, but a **+** symbol distinguishes it from other activities. Subprocesses are generally used to either break down a diagram and make it more readable or to describe repeated activities. Subprocesses can be visualized either in their **collapsed** or **expanded** state. On the left, the activity "Process Data" is displayed as a collapsed subprocess at the top. The expanded counterpart at the bottom reveals the inner workings of the activity.

Table 2.1.: Overview of the graphical modeling elements offered by *BPMN 2.0* [19].

2.4. Didactic Strategies

Lecturers and teachers use **Didactic Strategies** that are often tailored to eliminate difficulties of understanding or to close the gap between lecturer and audience. Furthermore, these strategies often are

- designed to pursue **specific learning objectives**,
- potentially derived from **individual experience**, and
- ultimately determining the **sequence of actions** that take place during a lecture.

Notably, some of them (like **Peer Instruction**) were scientifically derived and proven to be effective, so there is a valid reason to use them. Additionally, collaborative strategies like the **Jigsaw Teaching Technique** are valid approaches to facilitate the self-reliant exchange of knowledge between students. However, there exist several limiting factors that hinder a more wide-spread application of didactic strategies in educational settings.

Firstly, some strategies can not be used viably (without the support of technologies like ARS) during lectures, because specific conditions in the learning environment complicate this procedure. Considering the example of a group work task as part of the Jigsaw Teaching Technique, it becomes clear that *manually* splitting a small number of students sitting close together in a classroom into groups is generally easier than doing the same with hundreds of students scattered in a lecture hall. The same applies to an online studying scenario where students can not meet in person and communication takes mostly place in an uni-directional fashion. For example, in a video-chat-based lecture, voice chat or other means of communication are often disabled for the audience in order to prevent abuse of these features.

Secondly, only partially supported features can be the limiting factor for the successful execution of didactic strategies when using ARS. Looking once more at the example of Peer Instruction, many ARS only provide some of the required features required (e.g. posing questions) while other features like automatic creation of student working groups for a *Peer Discussion* or displaying the results of a *ConcepTest* on student devices is impossible [22][1].

For these reasons, it could be beneficial to lecturers and audiences that ARS offer full support for the realization of both *known/established* and *custom* didactic strategies. To provide more context and some tangible examples, the following sections present a selection of established didactic strategies that find use in universities and classrooms.

[1]The terms Peer Discussion and ConcepTest are elaborated on in Section 2.4.4.

2.4.1. Learning Questions

A basic strategy that lecturers use in various learning environments (e.g. in classrooms and lecture halls) is to pose one or more **Learning Questions** that concern a specific topic. In [23], *Kapp et. al* describe that integrating Learning Questions interactively by using a web-based platform is a strategy that can (when used correctly) significantly improve learning success of students. However, the authors state that learning success depends on how questions are constructed and designed. Posing several Learning Questions *during* a lecture at fixed points in time is a reasonable and common use case.

An example for this would be to strategically place questions at the *start*, *middle* and *end* of the lecture. One or more questions at the start of the lecture can be used to evaluate the knowledge of the audience before new learning material is presented. After the presentation block, one or more questions follow that check on the audience's understanding of the new topic that was recently explained. After that, another presentation block can follow. Finally, at the end of the lecture, one or more concluding questions can serve as a final check for audience understanding or to provoke new thoughts.

Moreover, it is stated in [23] that Learning Questions can also be used *away* from classrooms and lecture halls. The authors show that students who prepared for upcoming lectures at home by answering Learning Questions provided via a web-based platform performed significantly better in subsequent tests than students who were given a reading assignment instead. The latter scenario can be compared to **Just in Time Teaching**.

2.4.2. Just in Time Teaching

The concept of Just in Time Teaching (JiTT) can be regarded as a refinement of the Learning Question strategy. JiTT focuses on posing learning questions that concern an upcoming topic *in advance* (i.e. before the lecture, in which that topic is discussed, takes place). For this, questions and learning material are typically distributed via a learning platform or ARS over the internet. Lecturers and teachers can analyse the answers before the lecture takes place, giving them the possibility to foresee difficulties in understanding as well as "*allowing them to create an interactive classroom environment that emphasizes active learning and cooperative problem solving [...]*" [24].

2.4.3. Cooperative Learning

Cooperative Learning describes a class of collaborative learning methods in which students think through questions during several distinct phases that encourage individual participation. One possible variant consists of three phases: **Think**, **Pair** and **Share**. This method aims at promoting critical thinking and articulate communication in the classroom. In the first phase, each student independently thinks about a solution to a given problem. Afterward, the learners are grouped in pairs to discuss and exchange their ideas. In the final phase, the couples present their results of the discussion in front of the audience [25].

2.4.4. Peer Instruction

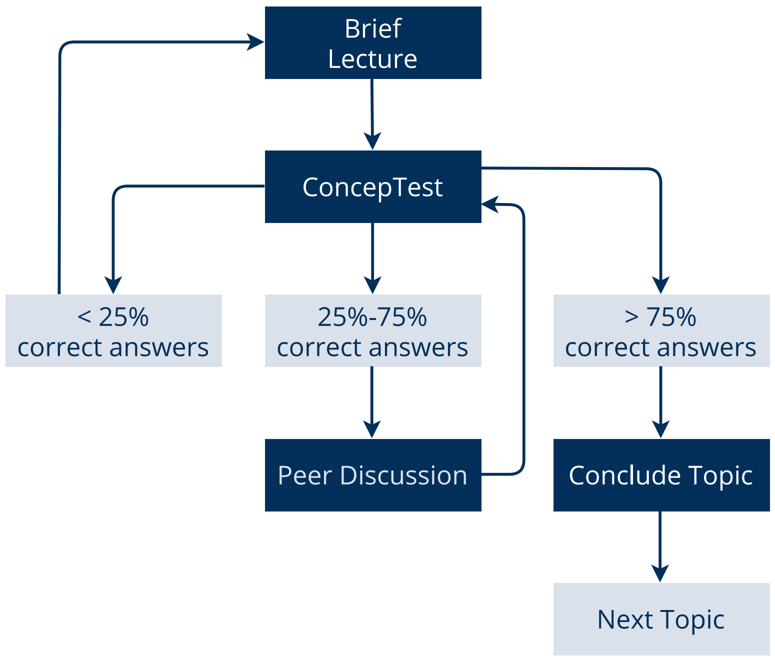


Figure 2.9.: One instance of Peer Instruction visualized as a workflow ($t_r = 0.25$ and $t_a = 0.75$).

Peer Instruction is a didactic concept developed by *Eric Mazur*, Professor of Applied Physics at Harvard University. The general idea of Peer Instruction is to engage students during class through activities that require each student to apply the core concepts being presented, and then to explain those concepts to their fellow students (see [26] and [27]). The strategy encourages learners to actively reflect on the teaching materials instead of just listening to it passively. By utilizing a more structured questioning process that involves every student in the class, lecturers are supported in their efforts to reduce comprehension difficulties among students. Prof. Mazur himself summarizes this phenomenon in his book as follows:

> "At times, it seems that students are able to explain concepts to one another more effectively than are their teachers [...]. A likely explanation is that students who understand the concept when the question is posed have only recently mastered the idea and are still aware of the difficulties involved in grasping that concept. Consequently, they know precisely what to emphasize in their explanation" [9].

There exist several variants of Peer Instruction – a very common version (that is depicted in Figure 2.9) is structured in the following way: First, the lecturer gives a short presentation (**Brief Lecture**) of about 10-15 minutes. After that, it's the audience's turn to take a **ConcepTest**: The ConcepTest represents a question concerning a single important concept that was explained in the preceding presentation. After the answers were collected, the results of the question are either summarized and displayed for all participants in the audience or communicated verbally by the lecturer. The important part here is the rate

t_c of students that answered the question correctly. Peer Instruction uses two different thresholds for t_c that govern how the strategy continues: the *acceptance threshold* t_a and the *rejection threshold* t_r[2]. Depending on the performance and understanding of the audience (i.e. on the value of t_c), the strategy continues in three different ways:

- If $t_c < t_r$, many students did not have the correct answer and the topic presentation should be repeated. One could say that the topic was *rejected* by the audience, and further explanation and clarification by the lecturer is needed. This repeat-when-necessary approach prevents a gulf from developing between the lecturer's expectations and the student's understanding – a gulf that, once formed, only increases with time until the entire class is lost [27].

- If $t_r < t_c <= t_a$, there exists an ambiguity in the audience's understanding of the subject matter. Some students understood the concept and answered correctly while others still have difficulties. Therefore, the audience continues with a **Peer Discussion** and discusses the question for a few minutes. While discussing with the person sitting next to them, the students re-evaluate their stance on the topic and try to convince each other of the correct answer. Following the discussion, a further ConcepTest is conducted. This process can be repeated to increase the number of correct answers in each round.

- If the vast majority of answers were correct ($t_c > t_a$), the topic was *accepted* (understood) by the audience and the lecturer can continue with the next topic.

Typical values for t_r and t_a are $t_r = 0.25, t_a = 0.75$ or $t_r = 0.3, t_a = 0.7$ or even $t_r = 0.25, t_a = 0.8$. The whole strategy can be carried out for each topic of the lesson, i.e. several times per session (see Figure 2.10).

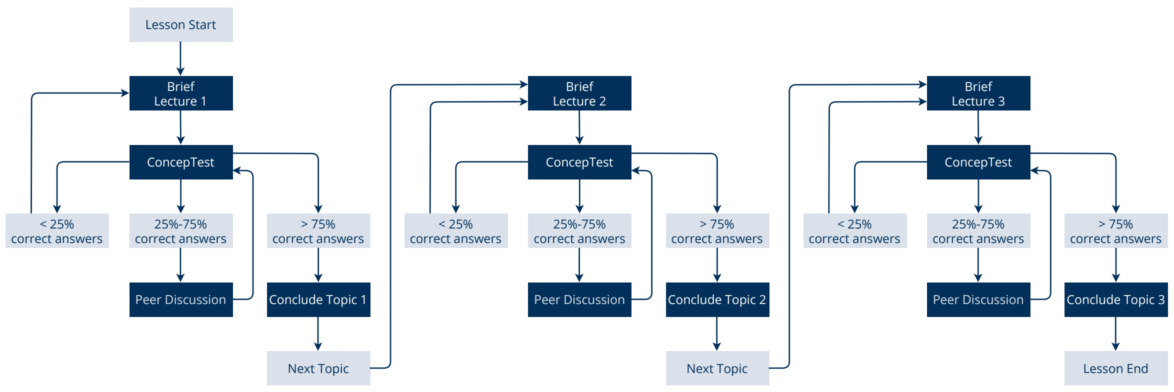


Figure 2.10.: A lecture consisting of three chained units of Peer Instruction.

[2]The terms acceptance threshold and rejection threshold were introduced here to simplify the explanation of Peer Instruction.

Role of Peer Instruction for this work

Peer Instruction is not natively supported by many ARS. It therefore serves as the most important example of a more complex didactic strategy for this work, playing a key role in

1. Documenting how only a few related ARS support Peer Instruction (coming up in Section 3.1).
2. Documenting problems and weaknesses of the *stARS* web editor when dealing with a scenario that involves multiple Peer Instruction units (coming up in Section 3.5).
3. Deriving the concept that extends the functionality of the *stARS* web editor to better meet the requirements for Peer Instruction and similar didactic strategies (coming up in Chapter 4).
4. Carrying out a pre-evaluation used to validate mock-ups for the concept part of this work (coming up in Section 4.1).

2.4.5. Jigsaw Classroom

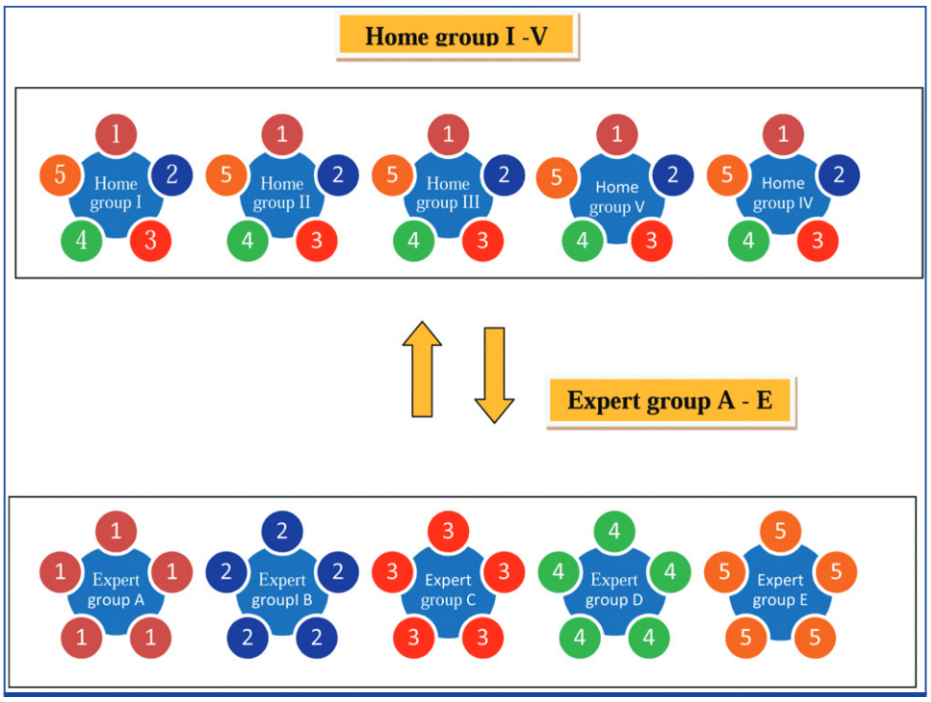


Figure 2.11.: The Jigsaw Teaching Classroom: Each assignment / topic is denoted by a respective color [10].

The group puzzle, also known as the **Jigsaw Classroom** is a form of group work that focuses on collaboration between students. Originating in the 1970s, it was designed and first described by social psychologist *Elliot Aronson* in an attempt to facilitate learning while defusing hostility between student groups. The key aspect of the strategy is an assignment (concerning a specific topic) that is split into several smaller assignments. As illustrated by Figure 2.11, the students are mixed and split into arbitrary groups that each have to solve the overall assignment. In this example, students are split into groups of five. However, each individual member of a group works on only one of the smaller assignments. Typically the Jigsaw Classroom (as described by *Aronson* in [28]) is carried out in three distinct phases: In the first phase, the students work in their *Home Groups*. Each student becomes an "*expert*" of their topic or area by learning about it and solving their small assignment. As a result, each group now has one expert for each of the smaller assignments.

In the second phase, the experts gather into *Expert Groups* divided by topic: All experts of one sub-area meet to compare and discuss their results. They synthesize information and create a final report that is enriched by the knowledge of all experts. Finally, in the third phase, all experts return to their original group to present their specialty to the other group members. The presentation provides the other group members with an understanding of their own material, as well as the findings that have emerged from topic-specific group discussions.

2.5. User Interface Engineering

The main goal of this work is to design a functional extension for the *stARS* web editor. Therefore, a major concern is the introduction of new user interface elements (e.g. dialog windows, buttons, menus, views) and concepts to the existing version of the *stARS* web editor. These additions and modifications are made to facilitate the support for complex didactic scenarios. For this reason, the following section introduces relevant terms from the field of *User Interface Engineering*.

2.5.1. User-Centered Design

As mentioned earlier in the introduction to this work, many existing ARS do not offer enough flexibility to lecturers, who then have to either adapt their didactic strategies to cope with the limited system support or not use the system at all. This might indicate that a lot of ARS were not designed with regards to the variable use cases that the end-user might have in mind. In terms of *product design*, a solution to this problem is to involve the end-user in the design process from the very beginning. This can help to avoid situations in which users are forced to change their behavior and expectations to adapt to the final product.

A product design philosophy that represents exactly this idea is **User-Centered Design** (UCD) [29]. The term describes several processes and techniques that aim at designing interactive products in such a way that they have the highest **Usability** possible. The main objective of UCD is to bridge the gap between the expectations of *product designers* (individuals that create and design the products) and *product users* (individuals that want/have to use the product). The majority of the design process is strongly centered around the user's capabilities, intentions, and needs that he or she has for using the product. In the context of UCD, these aspects have therefore a strong influence on how the product design takes shape.

Product designers need to constantly validate their assumptions concerning user behavior. This is achieved by real-world tests (involving real end-users) that allow them to analyze and envision the way users are likely to consume the product. A well established technique used in UCD is testing the viability of a concept with **mock-ups** and/or **prototypes**.

2.5.2. Usability and User Experience

Extending the concept of the *stARS* web editor to support complex scenarios imminently affects its **Usability** and **User Experience** (UX). Both terms stand in relation to the perspective the user has when interacting with the user interface of a given system. The *International Organization for Standardization* defines Usability as follows (*DIN ISO 9241*):

> "Usability is the extent to which a system can be used by certain users in a specific usage context to achieve goals effectively, efficiently, and satisfactorily" [30].

User Experience on the other side is defined as follows:

> "Perceptions and reactions of a person that are derived from the actual and/or the expected use of a product, system, or service result. [...] This includes all emotions, ideas, preferences, perceptions, physiological and psychological reactions, behaviour and performance that occur before, during, and after use" [30].

Usability thus concentrates on the fulfillment of tasks, on achieving the goal, and on avoiding negative feelings while doing so. UX on the other hand deals with the positive emotions associated with the usage of the product. Although usability and UX have very distinct definitions, both rely on the existence of one another. Great usability is the foundation for a great UX, but having only great usability does not necessarily mean that the UX is great too.

In summary, this work's objective can also be formulated in terms of usability and UX: The goal is to improve the usability of the *stARS* web editor by making it *easier* to create and reuse more complex scenarios while at the same time providing a good user experience.

2.5.3. Usability Heuristics

As described previously, a major concern of this work is the design, evaluation and implementation of user interface components that provide access to new features. Back in 1990, the **10 Usability Heuristics** [31] were defined by *Jakob Nielsen* that have established themselves as a golden standard guideline on how to design systems in such a way that usability problems can be identified as early as possible with minimum effort. The following sections present the original usability heuristics as they played a key role in the design of many of the UI mock-ups that were created during this thesis.

UH 1: Visibility of system status The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

UH 2: Match between system and the real world The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

UH 3: User control and freedom Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

UH 4: Consistency and standards Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

UH 5: Error prevention Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

UH 6: Recognition rather than recall Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

UH 7: Flexibility and efficiency of use Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

UH 8: Aesthetic and minimalist design Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

UH 9: Help users recognize, diagnose, and recover from errors Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

UH 10: Help and documentation Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

3. State of the Art

Chapter 3: State of the Art

Besides providing fundamentals on the topics ARS, workflow modeling, and user interface engineering, the previous chapter illustrated the complexity of more sophisticated didactic strategies. However, as will be discussed in this chapter, the current state of the web editor lacks vital features that allow the creation and utilization of models of such strategies. As simplifying the modeling process with *stARS* is the main objective of this work, this chapter will first elaborate on existing ARS with modeling capabilities in order to compare *stARS* with them. Afterward, the current state of the *stARS* project and the modeling capabilities of the *stARS* web editor will be discussed. Thereafter, other GMTs will be analyzed to identify key features that ease the creation of sophisticated models. This information will then be used in the final part of this chapter, where a *Requirements Analysis* is conducted, resulting in a summary of requirements that a suitable extension of the *stARS* web editor should ideally fulfill.

3.1. Other Audience Response Systems

The support of didactic strategies by digital tools like ARS is a desirable feature. However, not many ARS currently available on the market were designed in anticipation of didactic strategies. This section quickly elaborates on how these ARS that are very similar to *stARS* provide support for some of the didactic strategies that were presented in Chapter 2.

3.1.1. ARSnova

ARSnova [32] is a web-based ARS developed and maintained by the Technische Hochschule Mittelhessen. The system allows lecturers to create polls and quizzes that can be answered by students before or during a lecture via mobile devices. Supported question types include single- and multiple-choice, true/false- and free-text-feedback questions. Furthermore, an anonymous back channel system allows for feedback and questions from the audience.

Regarding the compatibility with didactic strategies, ARSnova partially supports Peer Instruction. A function that allows the execution of the same poll *twice* in a row with a subsequent comparison of results is provided, which can be used to mimic Peer Instruction. However, as stated in Section 2.4.4, Peer Instruction can go through more than two iterations. ARSnova does not provide a way to automatically rerun a poll if the survey results are not good enough. However, ARSnova does offer functionalities that enable the realization of JiTT. Lecturers can define and provide preparation quizzes and materials that students can work with to prepare for an upcoming lecture, therefore enabling lecturers to anticipate difficulties in understanding.

3.1.2. PINGO

Another example for a web-based ARS is **PINGO** [33], a free-to-use application maintained by the Universität Paderborn. The system was developed specifically with the intention to support Peer Instruction in scenarios with large audiences (i.e. lectures in universities). The most prominent feature of PINGO is the possibility to repeat polls as often as needed. Poll results are visualized in a before/after fashion, allowing the lecturer to decide with a single click (based on the results) whether a repetition of the poll is needed.

Furthermore, PINGO allows to define question catalogues that are persisted in the cloud and attached to the user account of lecturers. While the support of Peer Instruction is provided, more complex scenarios with different didactic strategies or variations of Peer Instruction are difficult to realize. Special features like providing access to polls in advance to support strategies like JiTT are not provided either. Nonetheless, PINGO does enable lecturers to realize Peer Instruction in a straight-forward manner. A detailed evaluation of PINGO that highlights the benefits of using it during lectures can be found in [34].

3.1.3. tweedback

tweedback [35] is an web-based ARS that was initially developed by individuals at the Universität Rostock. The application is platform independent and allows lecturers to pose single-choice, multiple-choice and numerical-feedback questions. A special feature of tweedback are the two distinct back-channel functionalities provided by the system. The first comes in the form of "panic buttons" that allow audience members to quickly give feedback on the lecture speed, volume, or on difficulties in understanding. The second feature is an anonymous chat wall allowing the formulation of textual feedback that is visible to both audience and lecturer. Advantages of tweedback are its easy setup process and straightforward user interface. However, additional features necessary to support more advanced didactic strategies (like Peer Instruction or JiTT) or more complex scenarios are missing.

3.1.4. FreeQuizDome

An example of an ARS that is not purely web-based is **FreeQuizDome** [36]. The system that was initially developed by individuals at the Universität Bielefeld requires lecturers to download a client for their platform before being able to use it. Polls that support a multitude of question types can be created with the client. Audience members can then access and participate in the polls via their mobile devices by scanning a QR-code or by opening a URL in a web-browser. An advantage of FreeQuizDome is its support of different media types such as photos, videos and emojis, that can be integrated into the polls.

In addition to that, FreeQuizDome offers some support for didactic strategies. For example, JiTT can be realized with the help of an option that allows audience members to access polls at all times (i.e. from home). However, repeating polls multiple times depending on poll results (i.e. support for Peer Instruction) is not possible and requires additional setup.

3.2. *stARS*: scenario-tailored Audience Response System

stARS is a research project for an ARS with modeling capabilities and better support for didactic strategies. The project is currently a work-in-progress and active development takes place at the Technische Universität Dresden. As already mentioned, both the meta-model and the graphical web editor that were introduced (in [7] and [8] respectively) represent preliminary work that was done to bring the *stARS* project to its current state. The following sections will elaborate more on the various components that make up *stARS* in its present state.

3.2.1. Architecture

stARS offers a web-based front-end application that grants access to the main system functionality. Both lecturers and students can access this application with the web-browsers of their devices. The front-end application consists of two parts: on the one hand, a user-interface for students is provided that allows them to participate in the learning scenarios that are executed during the usage of *stARS*. On the other hand, lecturers are provided the ***stARS* web editor** that enables them to create workflows for their lectures. Since the *stARS* web editor is the main topic of this work, it will be covered in great detail in Section 3.3.

Furthermore, a back-end server manages and provides access to the meta-model, a database persistence layer, and other administrative functions. *stARS* is capable of running an arbitrary amount of cloud servers to provide high performance to multiple lecturers and hundreds of students at the same time. The inner workings of the back-end architecture of *stARS* are not subject of this work, but more details can be found in [6].

3.2.2. Objective

As illustrated by the previously mentioned examples, many other ARS do not offer mechanisms that allow lecturers to model and orchestrate the schedule of their lecture. *stARS* has the goal to serve as a viable alternative to this by providing more flexibility to lecturers, enabling the creation of models for custom learning scenarios. Contrary to the approach of handling lectures as static blocks, *stARS* makes use of the workflow metaphor to represent lecture scenarios. The workflows are then executed during a live lecture that is accompanied by the usage of *stARS*. In the long term, additional functions are planned to be sup-

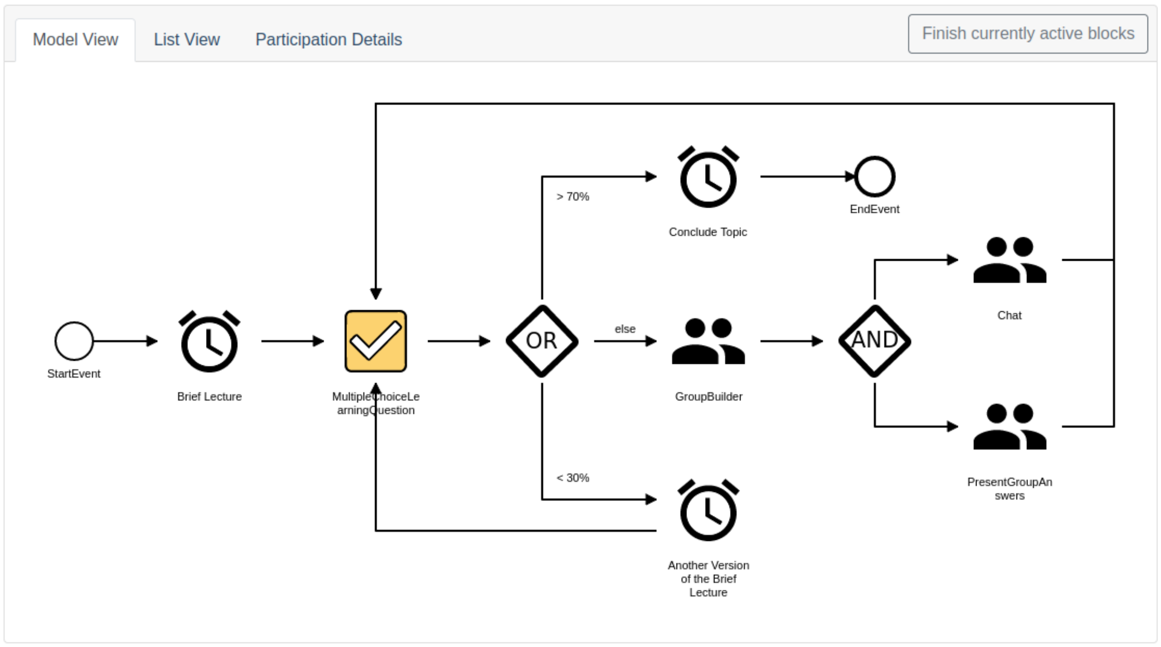


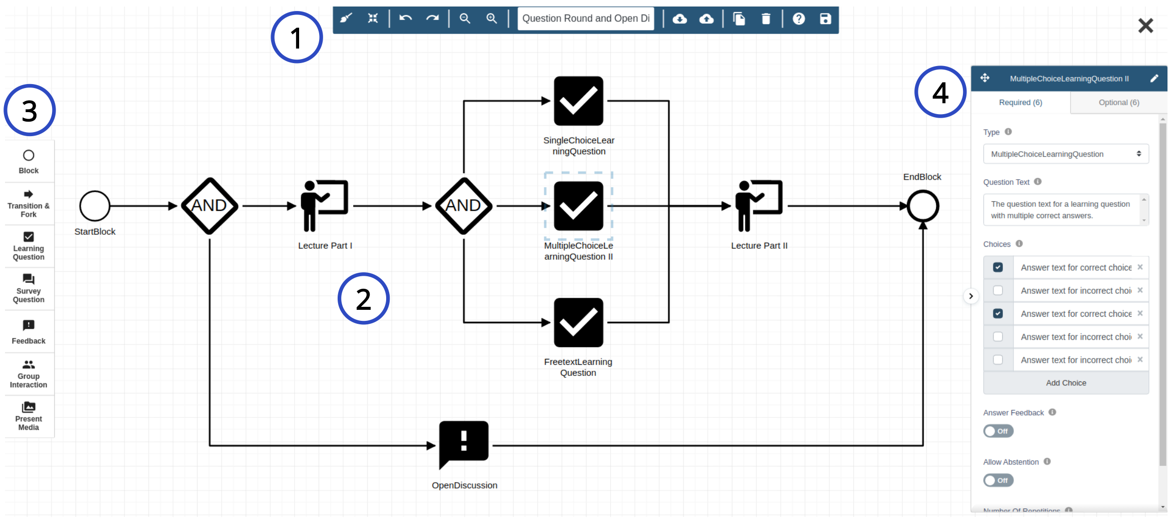
Figure 3.1.: Peer Instruction in the Control View of *stARS*: The block "Multiple Choice Learning Question" highlighted in yellow is active, meaning that a multiple choice learning question is currently displayed on the audience's mobile devices.

ported. One idea is to allow for more customization regarding the formation of groups, for example by building groups based on student attributes. Furthermore, in future versions of *stARS* the functionality of conditional flows might be extended. This will allow for more expressiveness in the context of modeling collaborative learning strategies. This extension could allow to model a group work scenario in which a bigger task is split into smaller ones, assigning one of them to each group, similar to the Jigsaw Classroom.

3.2.3. System Operation

From the viewpoint of lecturers, *stARS* is operated in the following way: The lecturer creates a learning scenario with the *stARS* web editor that resembles the lecture intended to be supported by *stARS*. The scenario is saved before it is available for execution during a live lecture (see Section 3.3.1). Once the lecture takes place, the lecturer uses the *Control View* to advance the scenario (see Figure 3.1). Advancing the scenario changes which Function Block[1] is currently active and therefore which functionality the ARS provides at a given point in time. For example, if the current active block is a *Single Choice Learning Question*, the respective question is shown on the devices present in the audience. When the lecture concludes (i.e. once one of the *End Blocks* is reached), the scenario is stopped and the involvement of *stARS* in the lecture ends. The scenario remains available in the system and can be restarted at a later point in time.

3.3. The *stARS* Web Editor



Type: MultipleChoiceLearningQuestion

Question Text: The question text for a learning question with multiple correct answers.

Choices:
- ✓ Answer text for correct choice
- Answer text for incorrect choi
- ✓ Answer text for correct choice
- Answer text for incorrect choi
- Answer text for incorrect choi

Answer Feedback: Off

Allow Abstention: Off

Figure 3.2.: The *stARS* web editor: the *Main Menu* on top, *Element Palette* on the left, *Properties Panel* on the right and the Model Canvas in the center.

The *stARS* web editor (see Figure 3.2) represents one of the main functionalities offered by *stARS*, and is the central topic of this work. Its purpose is to provide a set of tools that allow for the creation of model-based learning scenarios. This section will elaborate on all components of the web editor's user interface, explaining relevant functions in more detail.

[1] More information on Function Blocks follows in Section 3.3.2.

3.3.1. User Interface

The user interface of the *stARS* web editor consists of four main components: (1) the **Main Menu**, (2) the **Model Canvas**, (3) the **Element Palette** and (4) the **Properties Panel** (see Figure 3.2). As these components represent the central places of user interaction during work with the web editor, their purpose will be explained in the following.

(1) The Main Menu

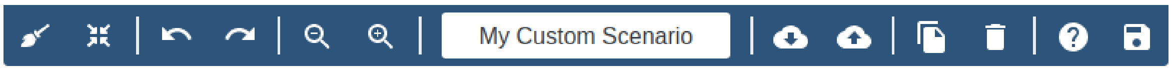


Figure 3.3.: The *Main Menu* on top of the *stARS* editor window.

The *Main Menu* can be found at the top of the editor window (see Figure 3.3). The buttons offer the following functions (from left to right):

- **Reset Model Canvas:** Deletes all model elements that currently reside in the Model Canvas. On confirmation, all changes previously made are lost permanently.
- **Center Model Canvas:** Re-orients the model canvas by centering the model.
- **Undo/Redo:** Allows to undo or redo the last change made to the model.
- **Zoom Out, Zoom In:** Used to manipulate the zoom level of the Model Canvas.
- **Scenario Name Field:** Can be used to give the current model a name.
- **Import Scenario, Export Scenario:** Allows importing a set of predefined scenarios provided by *stARS*. The export function allows to download the scenario as a vector graphic.
- **Copy Element:** Allows to copy the currently selected model element.
- **Delete Element:** Allows to remove the currently selected model element.
- **Help:** Provides a help dialog that explains all available Function Blocks and their features.
- **Save Scenario:** Allows to save a scenario for execution during a lecture. A scenario can only be saved if it is valid. For example, a scenario is invalid if it contains *Abstract Function Blocks* (see the paragraph **Abstract Function Blocks** in Section 3.3.2).

(2) The Model Canvas

The Model Canvas holds and displays the model that is currently being edited. Model elements can be selected from the *Element Palette* and be placed inside the Model Canvas (see Figure 3.6). The viewport of the Model Canvas can be manipulated by left-clicking and dragging the mouse.

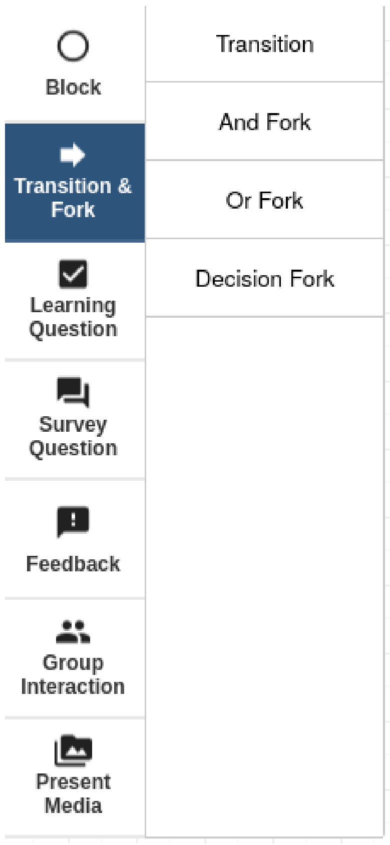


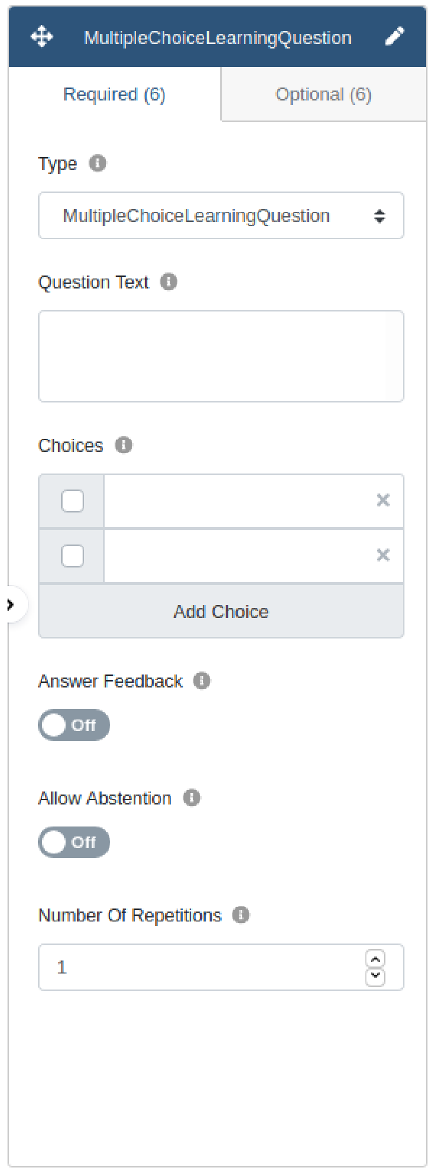
Figure 3.4.: The Element Palette.

③ The Element Palette

The *Element Palette* is located on the left side of the editor window (see Figure 3.4). It offers access to all available Structural Blocks and Function Blocks. Blocks can be selected from here and placed inside the Model Canvas. A summary of the available Structural Blocks and Function Blocks can be found in Table 3.1 and Table 3.2 respectively.

④ The Properties Panel

Each Function Block has several properties that influence the behavior of said block, providing more possibilities for flexibility and expressiveness. When selecting a Function Block, the Properties Panel is shown on the right side of the editor window (see Figure 3.5). It displays all properties and their current values which can be edited from here.


MultipleChoiceLearningQuestion

Required (6) | Optional (6)

Type: MultipleChoiceLearningQuestion

Question Text:

Choices:

Add Choice

Answer Feedback: Off

Allow Abstention: Off

Number Of Repetitions: 1

Figure 3.5.: The Properties Panel.

3.3.2. *stARS-MN* - The Graphical Workflow Notation for *stARS*

Chapter 2 briefly mentioned that the modeling language of the *stARS* web editor relies on a meta-model that was conceived in [7]. In the web editor, elements of the meta-model are represented conceptually and graphically by **Function Blocks** and **Structural Blocks** that were to large extents inspired by BPMN. They represent the elemental building blocks that workflows consist of. The modeling notation used in the *stARS* web editor is described in the following sections and will be referred to as ***stARS-MN***.

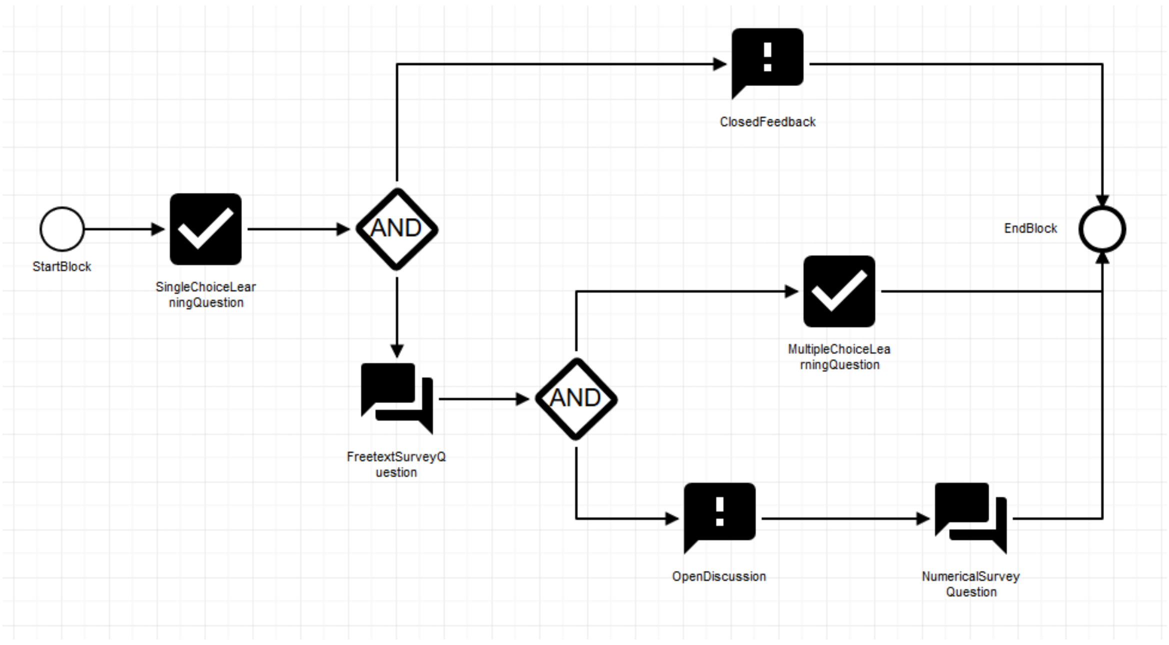


Figure 3.6.: The Model Canvas of the web editor: An example workflow with one *Start Block* and one *End Block*.

Structural Blocks

A commonality of valid workflows (regardless of graphical representation or model language used) is the existence of structural elements like one *Start Node* and *End Nodes*, because workflows need to start sometime and end under certain circumstances. In the context of *stARS* each valid workflow uses exactly one *Start Block* and an arbitrary amount of *End Blocks*. An overview of all **Structural Blocks** is given in Table 3.1.

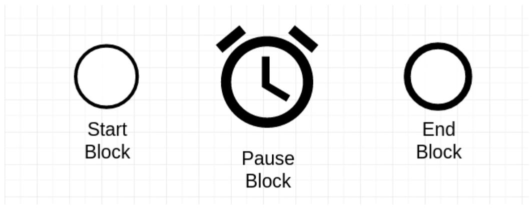
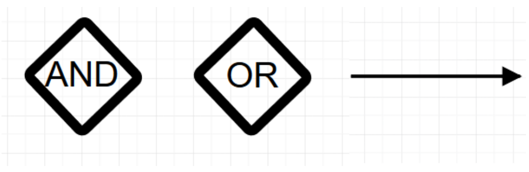
Structural Block Category	Description
Start Block, Pause Block, End Block 	This category contains blocks that deal with temporal aspects of a learning scenario, such as the beginning and end of a scenario. As already described earlier, a scenario has exactly one **Start Block** but can have multiple **End Blocks**. The **Pause Block** is used to signify that during certain intervals, the use of the ARS is *paused*. This is for example the case when a lecturer resumes with presenting his/her lecture after using *stARS* to ask the audience a Learning Question.
Transitions and Forks 	Blocks of this category are used to control the execution flow of scenarios. **AND-Forks** are used to design parallel activities by splitting the control-flow into several sub-flows. The **OR-Fork** is used when multiple paths in the workflow can be taken based on the outcome of a connected function block. Finally, **Transitions** connect different function blocks and are used to control the execution flow.

Table 3.1.: Structural Blocks of ***stARS-MN***: An overview of the modeling elements that give structure to a learning scenario[2].

Function Blocks

Between Start Blocks and End Blocks, any number of Function Blocks can be utilized to describe a specific lecture (see Figure 3.6). All elements (i.e. Start Blocks, End Blocks and Function Blocks) are connected by *Transitions*, which determine the order of the activities of a lecture. Because the tech-stack of the web editor (more on that in Chapter 5) relies on a framework called **bpmn-js**, the graphical notation is heavily influenced by BPMN. A summary of the currently available[3] Function Blocks can be found in Table 3.2.

Function Block Parameterization As described earlier, each Function Block has a set of parameters/properties that describe it more concretely. These properties enhance the flexibility given to lecturers when modeling a scenario, as different values for each property change the behavior of the respective Function Block. Examples for Function Block properties include question and answer texts of LearningQuestion Blocks, the image URL for PresentImage Blocks or a flag that allows or disables abstention from SurveyQuestion Blocks. More information about Function Block properties can be found in [7]. Users can manipulate properties with the help of the Properties Panel (see Figure 3.5).

[2]This list documents only blocks that were present at the time of writing (September 2020).
[3]The meta-model defines more concrete *Learning Question* types that are currently not supported by the web editor.

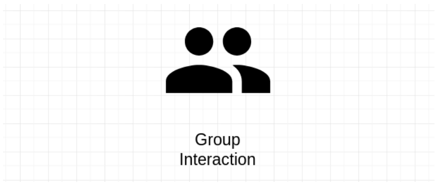
Function Block Category	Description
Learning Question Learning Question	**Learning Questions** have one or more correct answer(s) and can be solved by the students. The *stARS* web editor currently offers single-choice, multiple-choice, numerical and free-text answering mechanisms for Learning Questions.
Survey Question Survey Question	**Survey Questions** can be used to poll an opinion of the audience. Similar to Learning Questions, the *stARS* web editor currently offers single-choice, multiple-choice, numerical and free-text answering mechanisms.
Feedback	The **Feedback** category contains two distinct function blocks: **Closed Feedback** and **Open Discussion**. The Closed Feedback block gives students the opportunity to provide instant feedback on predefined feedback dimensions. It resembles a quantitative back-channel. An Open Discussion block allows students to ask own questions and discuss them with other students. Both blocks represent examples of classical Back-Channel functions.
Group Interaction 	This category contains blocks that deal with group work. The **Group Builder** block models the act of splitting the audience into groups. The **Chat** block can be used to model group discussions. The **Present Group-Answers** block represents students stepping in front of the audience/class to present results of a group work. The **Group Voting** allows group members to select a group answer to a question by vote. The **AudioVideoChat** block allows group members to join a video conference and to exchange ideas via voice and text chat.
Media Presentation Present Media	The **Media Presentation** block allows sharing different types of media with the audience during a lecture. Supported media types include pictures, videos, text or a timer. Additionally, the results of previous learning or survey questions can be displayed on students' devices using the **Present Result** block.

Table 3.2.: Function Blocks of *stARS-MN*: Overview of modeling elements and their graphical representation.[4]

[4]This list documents only blocks that were present at the time of writing (September 2020).

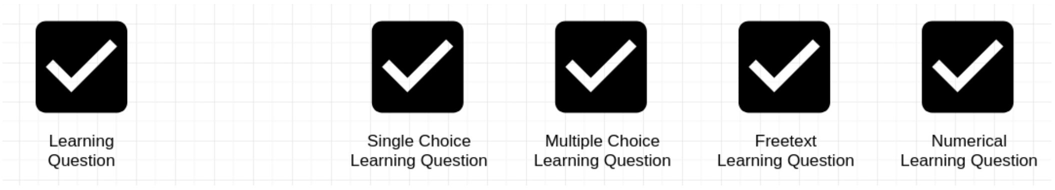

Learning Question
Single Choice Learning Question
Multiple Choice Learning Question
Freetext Learning Question
Numerical Learning Question

Figure 3.7.: Abstract and concrete Function Blocks: An *Abstract Learning Question* on the left and all four concrete counterparts currently supported by the web editor.

Abstract Function Blocks In addition to the concept of parameterization through properties, *stARS* supports the concept of **Abstract Function Blocks**, that represent a whole category of Function Blocks (see Figure 3.7). They are *valid* model elements and can be used as placeholders that indicate the structure of a concrete workflow. However, workflows that contain abstract Function Blocks can not be executed by *stARS*: as long as one or more abstract Function Blocks are present, it can not be instantiated and executed during a lecture. To make a workflow *executable* and therefore usable, it is necessary to specify the concrete type of the abstract Function Blocks. Later in Chapter 4, the concept of Function Block abstraction is explored further in the context of template creation.

3.4. Graphical Modeling Tools

With respect to the classification defined earlier (see Figure 2.5), the *stARS* web editor can be categorized as an *Application-specific Workflow Editor* (AWE) for learning scenarios. This work's objective is to extend the *stARS* web editor in such a way that more sophisticated learning scenarios are fully supported. For this reason, it seemed reasonable to first identify *which features* make the creation, manipulation, reuse, and overall work with complex models easier. This was achieved by studying other already existing modeling tools. Once these features and their purpose were identified, a basis was created on which *stARS* and its web editor can be compared against.

Researching applications that are available on the web resulted in a total of **19** different free-to-use GMTs that were experimented with. Most of the identified applications were web-based, which makes them very similar to *stARS* in terms of platform and runtime environment. These applications were analyzed to determine features that contribute to making work with complex models easier. While working with more and more of these tools, similarities between all GMTs were noticed in the approaches on (1) **Support For Nested Structures** and (2) **Component Reuse**. The following paragraphs will shortly motivate potential benefits of supporting nested structures and better support for component reuse, before elaborating on the different solutions to both of these problems that were identified while using the GMTs.

On the one hand, supporting **Nested Model Structures** has the following benefits:

- *Composition*: Support for composite models that consist of smaller models
- *Efficiency*: Easier reuse of models in bigger composite models
- *Effectiveness*: Better support for established didactic scenarios
- *Expressiveness*: Easier creation of models that have repeating or looping parts
- *Abstraction*: Collectively abstract parts of a model in a single modeling element

On the other hand, enhancing the possibilities offered for **Component Reuse** could likewise have the following benefits:

- *Efficiency*: Faster creation of new models
- *Variability*: Edit, adapt and reuse existing models
- *Templating*: Faster creation of several similar models based on templates
- *Collaboration*: Sharing of models between lecturers

All investigated GMTs offer different solutions to both of these problems that can be of interest for *stARS* and its web editor. These solutions and approaches are covered in more detail in the following sections.

1 Features related to Support For Nested Model Structures

Parent Containers

The term Parent Container describes special model elements that are responsible for embedding one or more model elements into another model element. GMTs that support nested model structures usually also visualize the embedded contents inside a Parent Container. Parent Containers can either originate from supported modeling notations (e.g. the subprocess of BPMN) or custom model elements (e.g. bounding boxes or containers) dedicated to nesting elements.

Expandable/Retractable Containers

Some GMTs that support nested model elements allow for these structures to be hidden on demand. Available Parent Containers can be expanded or retracted when needed. This feature is useful for reducing visual clutter and avoiding bloated model canvases. Additionally, retracting all Parent Containers in a model yields a top-level overview of the model.

Dedicated UI Elements

The term Dedicated UI Elements describes whether certain user interface elements exist that are dedicated to the visualization of embedded elements. An example for this would be *Tabs* that each render one BPMN subprocess.

Model Composition

Model Composition describes whether a model can be embedded into a parent model without replacing the contents of the parent model, effectively creating a composite of two or more models.

2 Features related to Component Reuse

Templates & Template Creation

Templates can serve as valuable shortcuts during the creation of new or more complex models. A template of a model commonly contains the structure and arrangement of model elements or placeholders thereof. Users can save time on the creation of models by using templates, which avoids the need to recreate a structure that should be used multiple times, although with slight modifications.

Template Repository

Template Repositories often provide templates for popular models or diagram types. There exist variations that solely provide public templates and variations were user generated templates can be shared. They represent mechanisms by which a collection of several templates is provided from which the user can choose.

Save, Load & Edit

This feature allows users to save a model, reload it, and edit it as many times as necessary.

Import

This term describes whether a model can be imported into the GMT. For this, models usually have to be present in a textual file format. Many web-based GMTs offer this feature via a file upload function. Importing a model allows users to view, alter and therefore reuse an otherwise acquired model (e.g. a model that was provided to them by someone else).

Export

An Export feature allows to generate a *textual* or *graphical* representation of the model. Many web-based GMTs allow downloading the model as a file to the user's file system. This feature allows to share models with others on the base of exchanging files.

Comparing the *stARS* web editor to other GMT

By identifying a set of features that most GMTs support, it was possible to compare *stARS* and its web editor to other GMTs. In conclusion, the *stARS* web editor does not support the majority of the features described above. A summary of the analysis of all GMTs that illustrates to which degree the aforementioned features are supported can be found in Table 3.3. One can say that the absence of almost all features that other GMTs support will likely make the creation of complex models with the *stARS* web editor more difficult. The following section will therefore elaborate concretely on the problems the web editor has by looking at the process of modeling an example workflow through the lenses of nested structures and component reuse.

General Info			Graphical Notation		Support For Nested Model Structures				Component Reuse				
Tool	**Type**	**Platform**	**Supported Notations**	**BPMN 2.0**	**Parent Containers**	**Expandable/ Retractable**	**Dedicated UI Elements**	**Model Composition**	**Template Creation**	**Template Repository**	**Save, Load & Edit**	**Import**	**Export**
Lucidchart [37]	*GDS*	*Web*	*multiple*	✓	*BPMN Sub-process*	×	*Tabs*	−	✓	✓	✓	✓	✓
Gliffy [38]	*GDS*	*Web*	*multiple*	✓	−	×	×	×	×	✓	✓	✓	✓
Visio [39]	*GDS*	*Desktop*	*multiple*	✓	*BPMN Sub-process*	×	*Tabs*	×	×	×	✓	✓	✓
bpmn.io [40]	*GWE*	*Web*	*BPMN 2.0*		*BPMN Sub-process*	×	×	×	×	×	✓	✓	✓
Camunda [41]	*GWE*	*Desktop*	*BPMN 2.0, CMMN, DMN*	✓	*BPMN Sub-process*	×	×	×	×	×	✓	✓	✓
Kissflow [42]	*AWE*	*Web*	*custom notation*	×	×	×	×	×	×	×	✓	✓	✓
draw.io [43]	*GDS*	*Web*	*multiple*	✓	−	✓	×	✓	✓	✓	✓	✓	✓
Alfred [44]	*AWE*	*Desktop (macOS)*	*custom notation*	×	×	×	×	×	×	✓	✓	✓	✓
CPN Tools [45]	*GWE*	*Desktop*	*Colored Petri-Nets*	×	×	✓	Sub-pages *and Tabs*	×	×	×	✓	×	×
Visual Paradigm [46]	*GDS*	*Desktop*	*multiple*	×	✓	✓	*Sub-pages*	✓	×	×	✓	✓	✓
cardanit [47]	*GWE*	*Web*	*BPMN 2.0*		*BPMN Sub-process*	×	*Tabs*	×	×	✓	✓	✓	✓
Activity Modeler [48]	*GWE*	*Web*	*BPMN 2.0*		*BPMN Sub-process*	×	×	×	×	×	✓	×	×
creately [49]	*GDS*	*Web*	*multiple*	×	×	×	×	×	×	✓	✓	$	✓
Flowable [50]	*GWE*	*Web, Desktop*	*BPMN 2.0*		*BPMN Sub-process*	×	×	×	×	✓	×	✓	✓
HEFLO [51]	*GWE*	*Web*	*BPMN 2.0*		*BPMN Sub-process*	✓	Separate Browser Tab	×	✓	✓	✓	✓	✓
Bonita Studio [52]	*GWE*	*Desktop*	*BPMN 2.0*		*BPMN Sub-process*	✓	Pools and Lanes	×	✓	✓	✓	✓	✓
cacoo [53]	*GDS*	*Web*	*multiple*	−	×	×	One Sheet per Diagram	×	×	✓	✓	✓	✓
smartdraw [54]	*GDS*	*Web, Desktop*	*multiple*	✓	*BPMN Sub-process*	×	*Tabs*	×	✓	✓	✓	✓	✓
mydraw [55]	*GDS*	*Desktop(Windows, macOS)*	*multiple*	✓	×	×	*Tabs*	×	×	✓	✓	✓	✓
stARS	*AWE*	*Web*	*stars-MN*	×	×	×	×	×	×	×	×	×	−

Table 3.3.: A summary of different *GMTs* and supported features. ✓ : Fully supported, − : partly supported, × : not supported, $: available in paid plan

3.5. Problems and Drawbacks of the *stARS* web editor

Creating learning scenarios with the *stARS* web editor is sometimes not possible without major inconveniences. To identify drawbacks and limitations of the current implementation, the *stARS* web editor is compared to the other GMTs whose relevant features were described in the previous section. The following sections will elaborate on weaknesses of the web editor in more detail by referring to an example model. It resembles a theoretical but realistic learning scenario designed purposefully to highlight the weaknesses of the *stARS* web editor in the areas of *Component Reuse* and *Nested Model Structures*. This *Example Scenario* can be described as follows:

Example Scenario

The scenario starts with a **Topic Presentation** during which the lecturer explains a specific topic to the audience. In the next step, a **Peer Instruction** unit is conducted (i.e. to validate audience understanding of the new topic). After the topic was understood by the audience, the lecturer wants to give the audience the option to determine how the lecture will continue. For this, a **Survey Question** with two answer options (*Topic 2A* or *Topic 2B*) is posed to determine which subject matter the students are more interested in. Depending on the outcome of the *Survey Question*, the lecture either continues with **Peer Instruction 2A** for *Topic 2A* or **Peer Instruction 2B** for *Topic 2B*. On conclusion of the second Peer Instruction unit, the scenario ends.

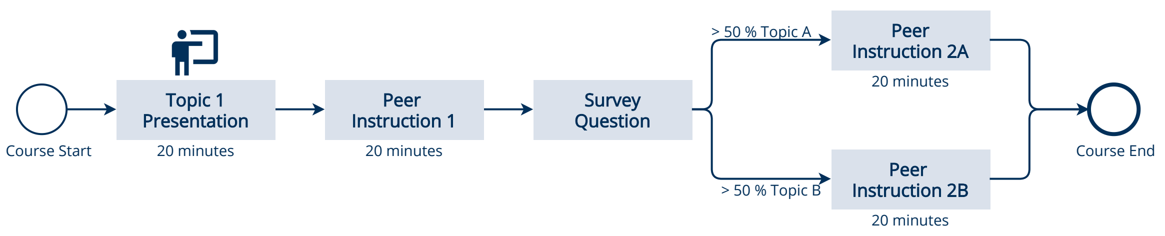


Figure 3.8.: A basic *Example Scenario* that illustrates how multiple instances of Peer Instruction can be used during a lecture supported by an ARS.

3.5.1. Component Reuse

A workflow modeled with the web editor can be seen as a component that (once it was created) should be reusable without the need to recreate it again from scratch. Looking at the *Example Scenario* in Figure 3.8 reveals that it consists of three different Peer Instruction units, meaning that Peer Instruction is reused *three times* in the same model. Modeling this structure with the current version of the *stARS* web editor is difficult and time-consuming. It requires users to manually create multiple identical copies of the Peer Instruction structure.

Templates & Template Repositories

Lectures tend to have a very similar but not the exact same structure (e.g. asking questions in consecutive lectures is useful, but asking *the same questions* might not be). To accommodate for these slight variations, reusing scenarios while applying minor modifications is a necessary feature. An example of a use case that illustrates this is the following situation:

Use case for Modification after Save

A lecturer creates the *Example Scenario* (see Figure 3.8) that contains multiple Peer Instruction units with the *stARS* web editor. Then, he or she conducts the lecture with the support of *stARS*. Once the lecture is finished, the scenario remains in the system as an executable model, meaning that it can be started again in a future lecture. During the next lecture, the lecturer wants to reuse the same scenario, though with a slight modification. This modification could be swapping the questions of the *ConcepTest* for another or changing the values of t_r and t_a.

The *stARS* web editor allows to modify a scenario after it was saved. The current implementation enables lectures to create the workflow depicted in Figure 3.8 and to alter it for the next lecture. However, the original scenario is lost once the modifications are saved. This can especially be a problem if the changes are bigger or more complicated. Jumping back to the original version of the scenario is not possible, as the undo stack is lost when saving the scenario. The lecturer therefore is forced to recreate the model from scratch. Because the model for Peer Instruction is rather complex, the recreation of it will be labor-intensive.

Many other GMTs offer a feature that circumvents the need to recreate models (and parts thereof) from scratch. They support the creation and usage of templates. Templates allow to quickly derive as many structurally identical copies of the same model as needed. For example, swapping a question for another would then only come down to editing the properties of a model element that is placed in the already existing model structure of the template. In addition to support for templates, many GMTs also offer Template Repositories that provide a variety of starting points for new models. The *stARS* web editor however does not allow for the creation of templates, although Abstract Function Blocks, a class of model elements suitable to represent templates, already exist. Furthermore, only a very limited Template Repository is offered in the import function. GMTs that support templates also often allow to load and edit templates once they were saved. However, the current Template Repository is not flexible enough as it only displays a fixed set of predefined templates to choose from.

Import & Export

Moreover, the majority of other GMTs offer **Import** and **Export** features that make reusing already existing components easier. Commonly, the export feature allows downloading a copy of the model to the user's file system. Consequently, the import feature allows for the inverse action (i.e. importing a model from the user's file system back into the GMT). Currently, an import and export feature is already part of the *stARS* web editor. However, the prototypical implementation of the editor offers only limited options. It is already possible to import exactly one workflow from a series of predefined templates. However, importing one of the available templates simply replaces the current content of the Model Canvas. This means that importing several, different sub-components into the same model is not supported. Lecturers are therefore unable to make use of component reuse on the base of combining different models and components into a bigger model. A realistic use case motivating a more mature import and export feature is the exchange and sharing of workflows and components among lecturers. The following table summarizes problems and drawbacks concerning **Saving**, **Loading**, **Importing** and **Exporting** scenarios as well as **Template Support**:

Name	Description
No Partial Export	Only the export of whole workflows as .bpmn or .svg files is supported.
No Import From File System	The import feature does not allow to import .bpmn files from the users file system back into the system.
No Model Composition	Embedding one model into another (Model Composition) is not possible, because importing a model always replaces whatever currently resides in the Model Canvas.
No Template Creation	The creation of Templates is not supported, although Abstract Function Blocks are already present.
No Extensible Template Repository	The import feature provides a Template Repository that consists only of a small series of predefined templates. It can not be extended by user-created Templates.
No Scenario Sharing	Sharing scenarios (for example via the Template Repository) is not possible. Because the import of .bpmn files is not supported, manual sharing of scenarios (e.g. via thumb-drives) between lecturers is also not possible.

Table 3.4.: Drawbacks of the *stARS* web-editor concerning *Component Reuse*.

3.5.2. Nested Structures

Sophisticated learning scenarios can often have a composite structure consisting of multiple, smaller sub-scenarios. Consequently, models of these learning scenarios have to map composite structures by **nesting model elements** in other model elements (e.g. by introducing a parent-child relation between model elements).

In theory, model elements can either be **atomic** or **composite**. Atomic model elements on the one hand lack any internal expandable structure (e.g. a *Survey Question* or any other Function Block would be an atomic model element). On the other hand, composite model elements do have an expandable internal structure that consists of several inner atomic or composite model elements (e.g. a whole Peer Instruction unit can be seen as a composite model element).

For instance, the *Example Scenario* illustrated at the beginning of this section has a rather clear and basic structure, including composite model elements (the three Peer Instruction units). Despite the model's simplicity, it is difficult to create models of similar composite structures with the *stARS* web editor. To illustrate the problems and limitations that occur during such a modeling process, the *Example Scenario* is recreated with the *stARS* web editor. In general, it is possible to create a model that closely resembles the *Example Scenario*. However (as can be seen in Figure 3.9) a problem that immediately arises is the **visual complexity** of this rather simple model.

Model Composition

In theory, as illustrated by the *Example Scenario*, combining (potentially different) existing models into one (i.e. Model Composition) is a valid use case. In practice, however, Model Composition is not possible with the *stARS* web editor. As already described in Section 3.5.1, the import feature of the web editor does not allow to combine multiple existing models into one as it replaces the contents of the Model Canvas on import.

Looking at the *Example Scenario* as it is depicted in Figure 3.9, it is plausible to imagine that it was created by importing and embedding an already existing model of Peer Instruction three times. Instead, every model element had to be placed manually, rendering the creation of this rather simple model very time-consuming. To speed up the modeling process, other GMTs provide functionalities that permit to embed a compatible model (or parts of it) into another model.

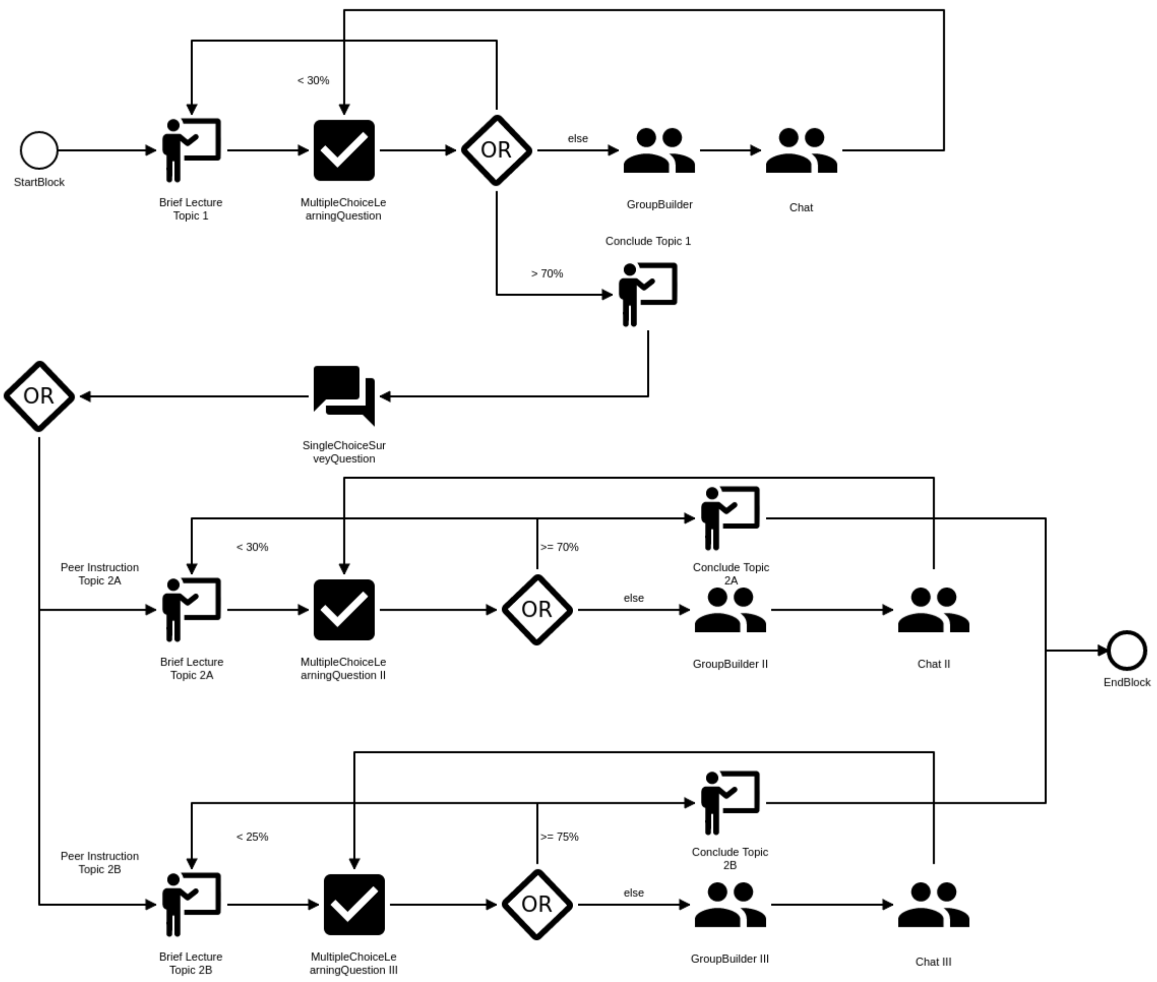


Figure 3.9.: The *Example Scenario* recreated with *stARS*: Peer Instruction introduces a lot of complexity to the model's visual representation.

Another reasonable use case is to *swap* one embedded model with another. For example, replacing one of the three Peer Instruction units with another structure (e.g. a Jigsaw Puzzle) for a future lecture could be a functionality desired by lecturers. However, with the *stARS* web editor, a model swap is not doable in a straight-forward manner since the elements that make up the embedded model are not treated as *connected* or *belonging together*. Instead, it is necessary to manually delete all Function Blocks of the sub-scenario, then place the Function Blocks of the replacement scenario and connecting them properly to the rest of the model while managing model boundaries. The very same can be said about the use case of simply *removing* an embedded model from the composite structure.

Parent Containers

Referring again to Figure 3.8, the three Peer Instruction units can be seen as sub-scenarios that are embedded into the structure of the *Example Scenario*. When looking at the model recreated with the *stARS* web editor in comparison (see Figure 3.9), it is evident that these embedded structures are not identifiable as *connected* model elements that *belong together* (at least not at first glance). The lack of some sort of *visual grouping* causes context information of Function Blocks and about their relation to one another to get lost. For example, it is not clear at a glance that any of the depicted structures resemble one or more Peer Instructions. Furthermore, the *boundaries* of the sub-scenarios remain unclear. The *Survey Question* could belong to the Peer Instruction preceding it or to one of the Peer Instructions that follow it or not to any other structure at all (which it does not). The structural complexity of the embedded Peer Instruction units makes it difficult to maintain an overview of the whole model and creates a visually overloaded Model Canvas.

Some other GMTs avoid these problems by offering special Parent Containers dedicated to embedded and composite structures. For example, a few GMTs use the BPMN subprocess in its *retracted* state to hide the complexity that the embedded subprocess has. In general, abstraction and simplification can help to reduce the complexity of large models. However, it is likewise in the interest of users that nested components of the model remain fully visible and editable. To avoid losing the possibility to edit embedded structures, the retracted container can be *expanded* again. Additionally, these containers help to visualize the boundaries of different sub-scenarios, removing potential ambiguities in the model structure.

Name	Description
No Model Composition	Embedding one model into another (Model Composition) is not possible, because importing a model always replaces whatever currently resides in the Model Canvas.
No Parent Containers	Parts of a composite or complex model can not be abstracted away as no Parent Containers (like the BPMN Subprocess) exist.
Difficult Management of Embedded Models	Removing a sub-scenario or swapping it for another is difficult because embedded models are not treated as connected structures that belong together.
No Clear Grouping of Connected Model Elements	In composite models that consist of several sub-scenarios it is difficult to see which Function Blocks belong to which sub-scenario.
Visual Clutter	More complex models can cause a visually crowded and overloaded Model Canvas that is difficult to maintain an overview of.

Table 3.5.: Drawbacks of the *stARS* web-editor concerning *Nested Structures*.

Dedicated UI Elements

Moreover, several GMTs make use of extra user interface elements (besides the main Model Canvas) dedicated specifically to the visualization of embedded and nested structures. These can range from **sub-pages** inside the Main Canvas, multiple canvases by the usage of **tabs** up to extra **windows** in which embedded model elements reside. Some web-based GMTs even introduce separate browser tabs as a means to visualize embedded contents. In contrast, the *stARS* web editor only has the Model Canvas dedicated to visualize the whole model at once.

A summary of all identified drawbacks regarding **Nested Structures** can be found in Table 3.5.

3.5.3. Context-sensitive Function Blocks

Some Function Blocks provided by *stARS* have context-sensitive properties, meaning that the semantics and purpose of these elements depend on the existence of other blocks. An example of this is the Chat Block that represents a group discussion. The Chat Block can not be used in a meaningful way without using the Group Builder block in the first place, as no group discussion can take place if no groups exist.

However, so far the editor does not treat these blocks as connected components. Both the creation and manipulation of these blocks as connected components in a single step is not supported. Furthermore, no visual indication of dependencies between blocks does exist.

3.6. Requirements Analysis

The previous sections elaborated on the drawbacks of the *stARS* web editor and problems that occur when creating, using, re-using and in general working with complex or composite models. This section will analyze the **requirements** that a solution to these problems has to fulfill.

3.6.1. Non-Functional Requirements

NFR1: Embeddable Solution

To maximize the reuse of already existing code and to avoid a complete rewrite of the *stARS* web editor, the concept for the solution should fit (from a software architectural perspective) into the already established structures. Therefore, the implementation and integration of the concept have to be realized as an extension of the web editor's existing code base.

NFR2: **Increase Usability of Existing Features**

A must-have requirement for the solution is the simplification of the modeling process, especially when it comes to modeling complex, large and composite workflows. Repetitive and time-consuming tasks such as modeling the same component multiple times manually should be simplified.

NFR3: **Intuitiveness of New Features**

The solution should avoid introducing new features that are either unintuitive, distracting, or that have no value to the end-user. New features introduced by the solution should not confuse users or make it even more difficult to use the *stARS* web editor.

3.6.2. Functional Requirements For Component Reuse

The solution should allow for the reuse of existing workflows in an efficient and flexible manner. This means especially that the support for export, import, saving, loading, and editing of existing workflows must be extended.

FR1: **Save & Export**

To increase possibilities for model reuse, the solution should give users the ability to save any given model as a Template. The solution has to offer a way of saving templates while retaining the ability to edit the template at a later point in time. Additionally, the export feature of the solution has to allow for the export of either executable models or templates.

FR2: **Load & Import**

The solution has to provide an import feature that enables the import of .bpmn files from the user's file system into the *stARS* web editor. The import of templates should be supported as well. Furthermore, re-opening (**loading**) an existing model should be possible.

FR3: **Templates & Template Repository**

The solution has to enable users the creation of templates. Any given model created with the web editor therefore has to be transformable into a reusable Template. Furthermore, to maximize flexibility and reusability, the solution has to provide options for transforming the model into a more abstract version that can be re-imported and refined in a future modeling process. A Template Repository that offers a way of browsing existing templates has to be provided. Templates have to be importable regardless whether they originate from the user's file system or from the Template Repository.

FR4: Scenario Sharing

The solution has to enable users to share scenarios (either as executable models or as templates that need to be refined).

3.6.3. Functional Requirements For Support of Nested Structures

The solution should enable users to profit from the expressiveness of the meta-model in the best way possible. The support for Nested Structures is essential to reach this goal.

FR5: Model Composition & Embeddable Models

The solution has to provide an easy way to compose new bigger models via combining multiple existing models. The number and kind of models that can be embedded side by side into **the first model layer** should not be limited by the solution. The solution has to treat models or parts thereof as embeddable components. Whenever a model is embedded into another one, the solution has to create special Parent Containers that hold the embedded model elements.

FR6: Adequate Visualization of Embedded Model Structures

Furthermore, the solution has to indicate and visualize embedded structures as connected and belonging together. When visualized by the solution, model elements that belong to one specific embedded structure have to be grouped and visualized in a specific manner to indicate parent-child relation (i.e. in such a way that it is clear which model elements belong to which *composite* model element)

FR7: On-demand Abstraction of Embedded Models

To reduce visual clutter introduced by complex and nested models, the solution has to enable the on-demand abstraction of nested structures. To gain an overview of complex models, the solution has to enable users to hide composite model elements in such a way that a top-level view of the model remains.

FR8: Modifiable Embedded Models

The solution has to guarantee that embedded models (or parts thereof) remain modifiable after they have been embedded into another model to provide as much flexibility as possible. The ability to hide parts of a model (see FR7) should not impede the ability to modify the model (e.g. both the structure and arrangement of Function Blocks and their properties have to remain editable).

***FR9*: Exchangeable & Removable Embedded Models**

The solution has to provide a way to easily exchange whole embedded models with one another. Therefore, Parent Containers and their contents have to be treated as exchangeable. Furthermore, the solution has to enable users to easily remove embedded models completely from a bigger model.

3.6.4. Summary

The previous sections elaborated on the current state of the *stARS* web editor to understand where improvements are necessary to increase its usability. To accomplish this, other GMTs were investigated comparatively to identify techniques and best practices that ensure high functionality and usability in the context of *Component Reuse* and *Support for Nested Structures*. Finally, several requirements for the concept that will be proposed later in this work were identified and described. The following table summarizes these requirements and illustrates the importance and implementation effort of each requirement.

ID	Name	Importance	Implementation Effort
NFR1	Embeddable Solution	☆☆☆☆☆	☆
NFR2	Increase Usability of Existing Features	☆☆☆☆☆	☆☆
NFR3	Intuitiveness of New Features	☆☆☆☆☆	☆☆
FR1	Save & Export	☆☆☆☆☆	☆☆
FR2	Load & Import	☆☆☆☆☆	☆☆☆
FR3	Templates & Template Repository	☆☆☆☆☆	☆☆☆
FR4	Scenario Sharing	☆☆☆	☆
FR5	Model Composition & Embeddable Models	☆☆☆☆☆	☆☆☆☆☆
FR6	Adequate Visualization of Embedded Model Structures	☆☆☆☆☆	☆☆☆
FR7	On-demand Abstraction of Embedded Models	☆☆☆☆	☆☆
FR8	Modifiable Embedded Models	☆☆☆☆☆	☆☆☆
FR9	Exchangeable & Removable Embedded Models	☆☆☆	☆☆

Table 3.6.: Non-Functional and Functional Requirements for a concept that extends the *stARS* web-editor. (☆= low, ☆☆☆☆☆= very high importance / implementation effort)

4. Concept

Chapter 4: Concept

In the previous chapter, the current state of the *stARS* web editor was described to understand exactly, where improvements to the usability of the web editor can be made. The chapter concluded with a list of requirements that a suitable extension of the web editor should ideally fulfill to solve the current version's problems and drawbacks. In this chapter, the approach that was taken to derive a concept for such an extension and the results are presented.

Firstly, the methodology of the approach that mainly consists of user-centered design techniques is outlined. Secondly, the chapter presents the final versions of the mock-ups for several graphical user interfaces that provide new functions for **Component Reuse** and **Support for Nested Structures**.

4.1. Methodology

Essential features like exporting, importing, loading, and saving workflows as well as support for nested model structures were elaborated on in the previous chapter. The introduction of new graphical user interfaces like windows and dialog boxes is required to provide access to all of these features. To avoid a gap between end-user expectations and system designer assumptions, the features and their user interfaces have to be designed with the end-user in mind.

To identify the needs, opinions, and feelings of real end-users, an evaluation was conducted with professionals of the educational domain. Working closely with these individuals, the goal was to design a concept for every new user interface component and its intended functions and features before actually implementing them. This way, only user-approved (i.e. wanted and necessary) features would be realized in the implementation phase.

A total of **24** educationalists (lecturers, teachers, and professors) were asked to rate several mock-ups for a **Save & Export Feature**, a **Load & Import Feature** and proposals for the visualization of and interaction with **Nested and Embedded Model Structures**. The evaluation was conducted in two phases (18 participants in the first phase, 6 participants in the second phase) to allow for iterative development and improvement of the proposals.

Due to the outbreak of the Coronavirus, both evaluation phases were conducted online with the help of a survey and video conferencing. Furthermore, both evaluation phases followed the same structure. After questions that concern the participant's experience with using technology and ARS in their lectures, participants had to use the *stARS* web editor to create several models, one of them being an instance of Peer Instruction. Participants were then asked to rate the usability of the current version of the editor.

4.1.1. Mock-ups

In the final part of the evaluation, participants were shown mock-ups for the new features to be introduced by this work. Each mock-up was carefully created considering the above requirements and Nielsen's *Usability Heuristics*. The process was accelerated by using **pingendo** [56], an editor that enables rapid prototyping with **bootstrap**-based styles and HTML building blocks. One advantage of this approach is that only minor work was required to adapt the mock-ups to the established corporate design (e.g., button colors, fonts, spacing between different user interface elements) of the *stARS* web editor. Therefore, it was easier to follow the usability heuristic *UH 4: Consistency and Standards*. Since Pingendo is entirely based on bootstrap CSS classes, parts of the code of the mock-ups can be easily reused in the implementation phase, which is another major advantage of this approach. The UI components provided by bootstrap have a simple and accessible design that follows the usability heuristic *UH 8: Aesthetic and minimalist design*. The professionals were confronted with the mock-ups and asked for their feedback. Besides the quantitative rating of the proposals with the help of Likert-scales, qualitative feedback with open-text questions was collected. Using this feedback, the mock-ups were improved by eliminating unnecessary features, adding missing but requested features, and adapting the layout and alignment of proposed user interface elements. These improvements led to significantly more positive feedback from the participants of the second evaluation phase. The final results of the evaluation will be explained in the following sections, elaborating on how each proposed mock-up attempts to solve the functional requirements that were defined earlier.

4.2. Saving and Exporting Templates

An essential function of any GMT is being able to save a model, as stated by requirement **FR1** (*Save & Export*). Fulfilling this requirement enables users to persist their models in the cloud to re-open them for further refinement at a later point in time or sharing a model that was saved to the local file system. A new dialog window that combines both functions of saving and exporting templates into one view is proposed that specifically addresses requirement *FR1*. The view consists of two tabs, one for saving and one for exporting templates to the local file system. The final mock-up of the dialog is shown in Figure 4.1. The following sections will describe both tabs that will be referred to as the **Save Dialog** and the **Export Dialog**.

4.2.1. Save Dialog

The Save Dialog is intended to compliment the existing *save* functionality by allowing to save templates. Access to the new dialog is provided by the existing *Export Button* of the *Menu Bar*. The dialog consists of a place to enter (1) **Template Name And Description**, a (2) **Template Preview**, a (3) **Category Input** and several (4) **Template Options**.

(1) Template Name And Description

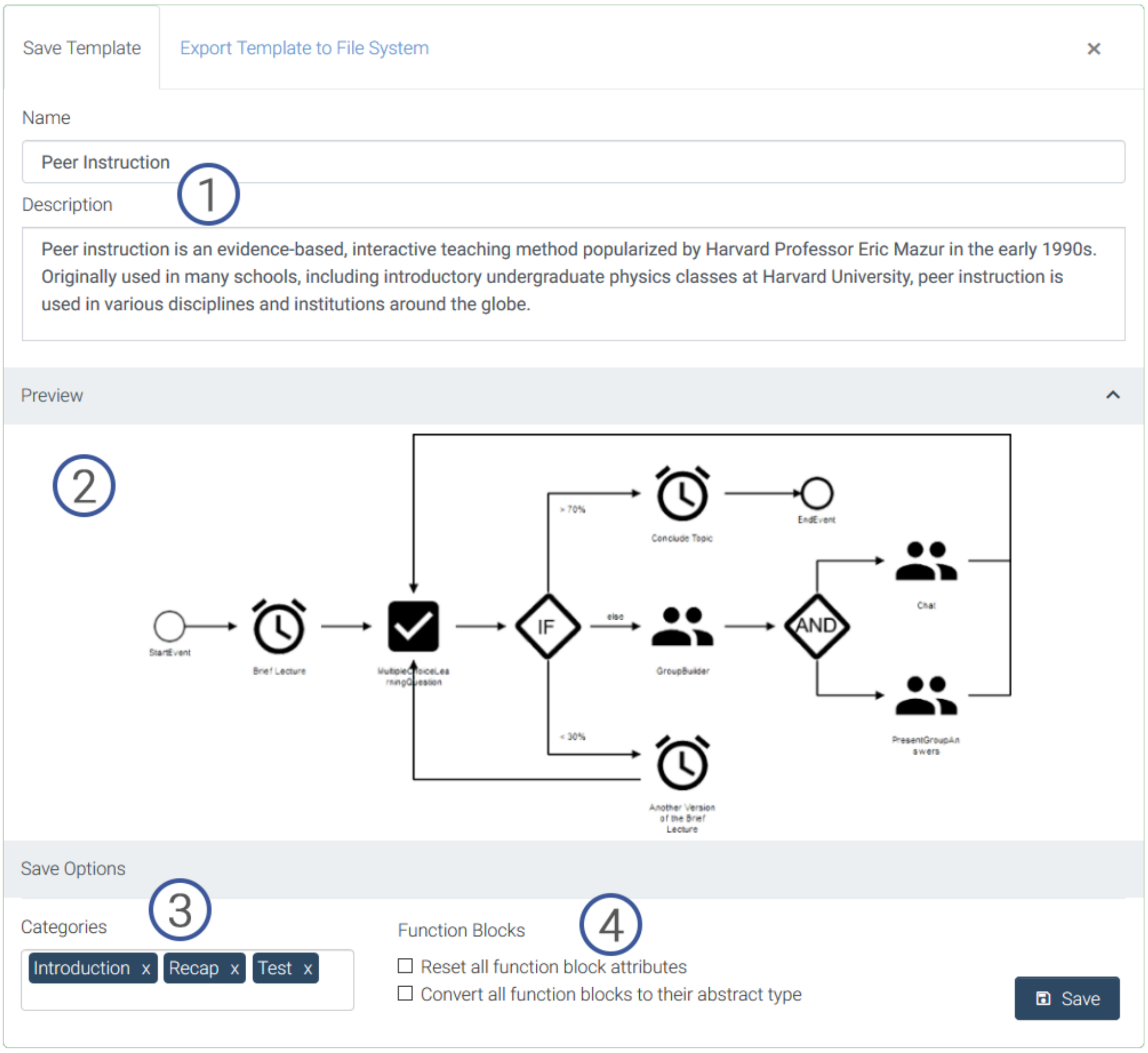
Besides the possibility to specify the name of the template, the user can add and edit a descriptive text. This meta-data is then used to locate the saved template at a later point in time, for example via a search functionality.

(2) Template Preview

Many participants rated the visualization of a preview of the workflow that should be saved as a template as useful. Therefore, below the description, a model preview is shown to the user. The preview container can be retracted and expanded.

(3) Category Input

Furthermore, several participants of the evaluation remarked that it would be useful if a better way of organizing their templates existed. Therefore, the concept of template **cat-**



Name

Peer Instruction

Description

Peer instruction is an evidence-based, interactive teaching method popularized by Harvard Professor Eric Mazur in the early 1990s. Originally used in many schools, including introductory undergraduate physics classes at Harvard University, peer instruction is used in various disciplines and institutions around the globe.

Categories

Introduction x Recap x Test x

Function Blocks

☐ Reset all function block attributes
☐ Convert all function blocks to their abstract type

Figure 4.1.: Save and Export Dialog for the *stARS* web editor.

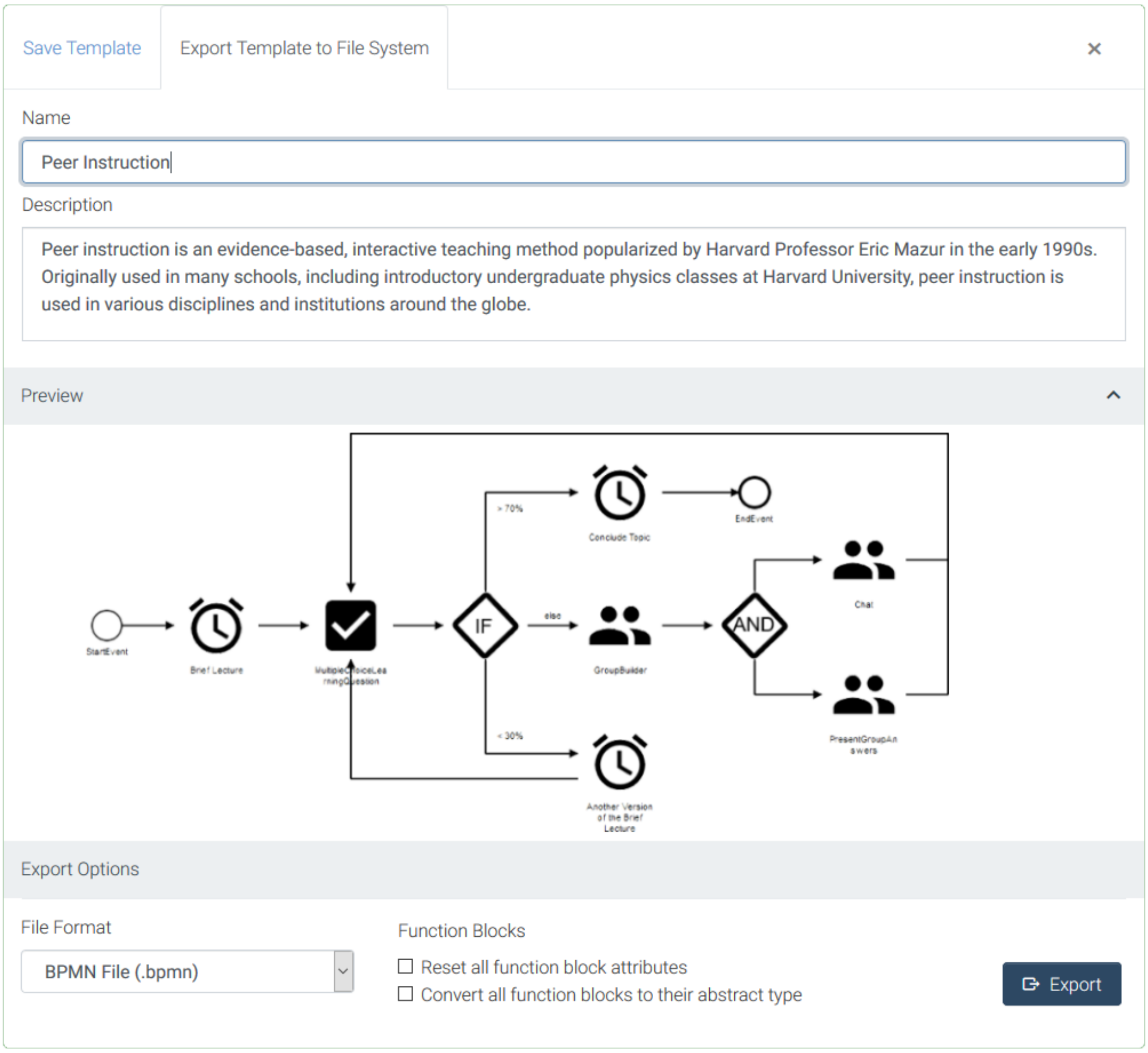
egories is introduced. When saving a template, users can assign a custom number of categories to any template via the category input box at the bottom of the dialog. Existing categories that were earlier created by the user are suggested in an auto-completion manner.

4 Save Options

Next to the category input field, two options can be checked to affect the behavior of the function for saving. The first option allows resetting all function block attributes. Checking this box will cause all attribute values for all function blocks to be cleared. In other words, all function blocks and the links between them will be kept, but the concrete values for question texts and answer formulations and other attributes will be removed. When the second check box is selected, all function blocks are converted to their abstract type. For example, all *SingleChoiceLearningQuestions* would be converted to *LearningQuestions*. This functionality can be used to create abstractions of models that can be refined in a future modeling process. Effectively, both options represent functions that realize parts of the

FR3 requirement by allowing users to create reusable templates from their models. Both checkboxes control how much the template is abstracted. If neither checkbox is selected, the model is simply saved as a template without any changes. 89% of the participants of the first evaluation phase rated this feature as useful, the most common remark being that "*this feature enables the creation of templates*".

4.2.2. Export Dialog


Save Template | Export Template to File System ×

Name: Peer Instruction

Description: Peer instruction is an evidence-based, interactive teaching method popularized by Harvard Professor Eric Mazur in the early 1990s. Originally used in many schools, including introductory undergraduate physics classes at Harvard University, peer instruction is used in various disciplines and institutions around the globe.

Preview

Export Options

File Format: BPMN File (.bpmn)

Function Blocks
- ☐ Reset all function block attributes
- ☐ Convert all function blocks to their abstract type

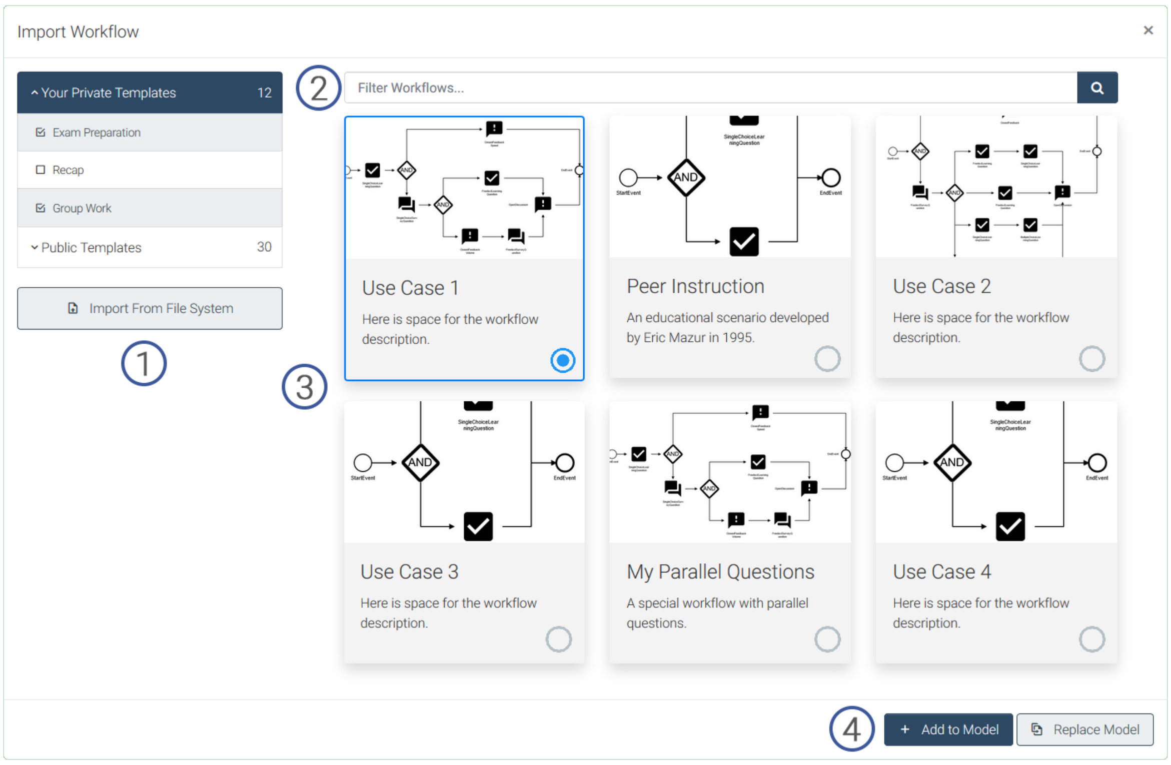
Export

Figure 4.2.: Export Dialog for the *stARS* web editor.

The *Export Dialog* allows the user to export models/templates to their local file system. The dialog offers the same functionality (i.e. name and description inputs, a model preview, and abstraction options) and is structurally identical to the Save Dialog, except that the category box is missing. This design decision was made intentionally to follow the usability heuristic *UH 6: Recognition rather then recall*. Similar to the *Save Dialog*, the user can assign a name and a description to the model. Finally, an option to select the export file type is given, which can either be .png or .bpmn.

4.3. Loading and Importing Workflows

Similar to the export and save features, a new dialog is proposed that allows users to load or import a workflow. The proposal concerns requirement **FR3**, as the proposed dialog is intended to function like a Template Repository. Both private and public templates can be browsed from this dialog. The final result of the iteratively improved mock-up is shown in Figure 4.3. The mock-up consists of a ① **Sidebar** on the left, a ② **Search Bar** on the top, a ③ **Template Browser** of available templates and ④ several **Import Options** on the bottom.



Use Case 1
Here is space for the workflow description.

Peer Instruction
An educational scenario developed by Eric Mazur in 1995.

Use Case 2
Here is space for the workflow description.

Use Case 3
Here is space for the workflow description.

My Parallel Questions
A special workflow with parallel questions.

Use Case 4
Here is space for the workflow description.

Figure 4.3.: Final mock-up of the Load and Import Dialog.

① Sidebar: Private and Public Templates, Import From File System

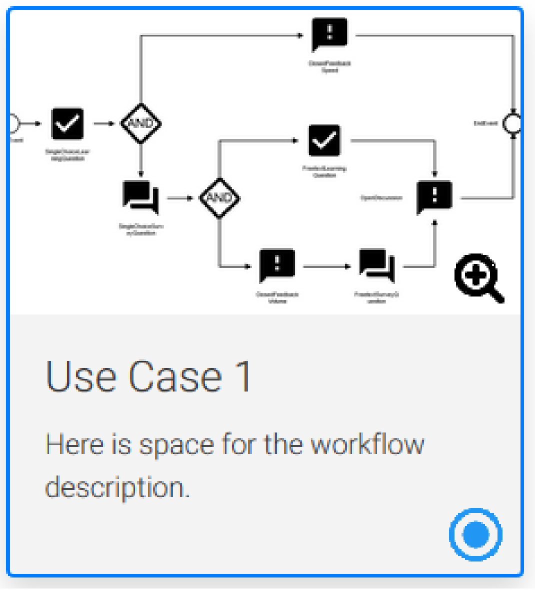
The mock-up proposes the introduction of categories for **private** and **public** templates. Each template created and saved by a user is a private template associated exclusively to the user's account. A retractable list of all template categories is displayed in the *Sidebar* on the left side of the import dialog. Expanding the private template section shows a list of user-defined categories that can be selected or deselected to filter the template previews that are displayed in the *Template Browser* on the right.

Below the private template section, a list of publicly available templates is shown. Public templates are provided by the *stARS* web editor itself and can be compared to the selection of workflows provided by the import feature currently in place. Public templates are created and managed by users with the role "admin". At the bottom of the *Sidebar*, a button is located that allows importing workflow templates from the local file system. Clicking the button will open an OS-based file browser that allows for the selection of a file to be uploaded.

2 Search Bar

At the top of the dialog, a *Search Bar* is located. The *Search Bar* can be used in combination with the categories in the *Sidebar* to quickly locate a template. Users can enter the name of a template, parts of the description or a category name to filter available templates. Upon entering a term into the *Search Bar*, the *Template Browser* updates accordingly. The proposal of the *Search Bar* was approved by the participants of the evaluation, as 89% of the participants stated that a *Search and Filter* functionality is necessary.

3 Template Browser


Use Case 1

Here is space for the workflow description.

Figure 4.4.: Cards for Templates: The blue radio button indicates that this template was selected.

The *Template Browser* visualizes templates that match filter terms and selected categories. Templates are visualized as *Cards*, showing the title, description, and a preview of the template (see Figure 4.4). This proposal was iterated and improved on in two ways during the evaluation. Firstly, users wished for a way to see the complete template before importing it, as the small preview of the *Card* is rather limited. Therefore, hovering over the preview will show a small magnifying glass to indicate to the user that this preview can be enlarged. Clicking anywhere on the preview image will display the template in a separate window in full size. Secondly, the results of the evaluation indicated that users would rather not be able to import multiple templates at once. In earlier drafts, multiple templates could be selected at once via check-boxes inside the *Cards*. However, only 28% of the participants rated a multi-import feature as desirable. Therefore, the proposal was modified by replacing the check-boxes with radio buttons, allowing for the maximum of one template to be imported at once. Clicking the radio button inside the *Card* will select it for import.

4 Import Options

Below the *Template Browser*, two buttons are displayed that each offer different options regarding the import. The first button labelled "Add To Model" specifically addresses the requirement **FR5** (*Model Composition*). Clicking this button will add the selected model to the existing model. This way, it is possible to import multiple different models into the Model Canvas, enabling the composition of bigger models from several smaller ones.

Contrary to this, if composing a bigger model from smaller ones is not the user's intention, the other button labeled "Replace Model" will replicate the behavior of the current implementation of the *stARS* web editor by replacing everything in the Model Canvas with the currently imported model. Choosing this option will cause the imported model to *not* be placed in a Parent Container.

The import options are also planned to be available when importing a template from the user's file system. After selecting and uploading the file, a dialog box will ask the user if the model should be added to the canvas or if it should replace the content of the canvas. In theory, with this feature in place, end-users can compose bigger models by combining their models with models that were shared by other users (e.g. models that were exported and provided over a network drive or thumb drive).

4.4. Nested and Embedded Model Structures

To address the requirements **FR5** to **FR9** that all deal with the support of *Nested and Embedded Model Structures*, several mock-ups that illustrate how these components should be visualized and interacted with were created and evaluated. Firstly, the results of the pre-evaluation confirmed that participants rate the support of *Nested and Embedded Model Structures* as useful and important. Secondly, participants were then asked to rate several proposals concerning the visualization of and interaction with such structures. The following sections will first go into detail about different alternatives and proposals that were presented to the participants, before concluding with the presentation of the final design.

4.4.1. Visualization of the Parent Container

As stated in requirement **FR6**, embedded and nested model structures need to be visualized adequately. To meet this requirement, a design for a Parent Container that holds embedded model components was proposed in the pre-evaluation. The concept was inspired heavily by the way BPMN subprocesses are visualized. In general, a nested model is represented visually by a rectangular container labeled with the element's name. An example of how a model or template named Peer Instruction will initially be visualized after it was imported into the Model Canvas is depicted in panel ① of Figure 4.5.

To meet requirement **FR7**, the container is designed to be expandable and retractable. Right after a model is imported, the container will rest in the contracted state to avoid a cluttered model canvas. A button with a plus symbol is displayed at the bottom of the model element container that allows expanding the container in place. This means that after expanding the container, the complete model is shown within the container.

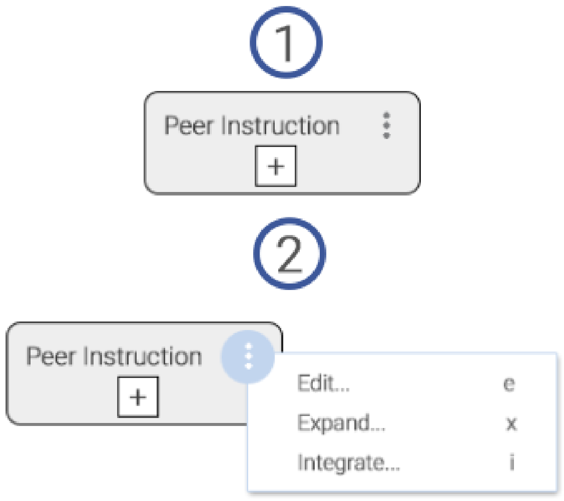


Figure 4.5.: An embedded model element in the collapsed state. The design was inspired by the BPMN subprocess. In panel (2), the expanded Context Menu is shown.

4.4.2. Usage of Dedicated UI Elements

While the concept for the contracted Parent Container visualization was approved by the majority of the participants, it was initially uncertain whether dedicated user interface elements should be used to visualize its contents. Furthermore, an open question was how to ensure that the embedded model stays modifiable as required by **FR8** (*Modifiable Embedded Models*) while avoiding an overloaded and cluttered Model Canvas. To answer these questions, participants of the first phase were given three different proposals for visualizing the contents of an embedded model via the usage of dedicated user interface elements. The results were analyzed and taken to construct the final proposal that was evaluated in the second evaluation phase.

In-Place Visualization The first proposal was to simply expand the Parent Container and display the embedded model content *in place*, i.e. without the usage of any dedicated user interface elements. 67% of the participants found that this option was the best. The main reason given by the participants was "*Being able to see the whole model at any given point in time*". The qualitative feedback made clear that participants highly valued the ability to see all model details at all times and that abstraction of embedded model elements might not be as important to the users as initially assumed.

Pop-Over Visualization The second proposal was to display the contents of an embedded model inside a *Pop-Over* window that hovers over the Model Canvas. This approach intended to keep the Model Canvas uncluttered and to avoid the positional displacement of other model elements when embedded contents were displayed. Interestingly, this approach was rated the worst, as only 11% of the participants found it appealing, with one participant remarking that this approach felt like a "*limited editor inside an editor*". Therefore, this proposal was excluded from further investigation.

Tabs The third proposal was to display each embedded model in a separate tab. One root tab for the whole model and one tab for each embedded model would be displayed at the bottom of the Model Canvas. Selecting the root tab would then display the whole model with all embedded models in their retracted state. Clicking on a Parent Container or selecting the tab at the bottom of the Model Canvas would then open the corresponding model in the tab. 22% of the participants rated this proposal as appealing, mainly remarking that tabs "*keep the model small and improve clarity*" and "*provide a clear division of hierarchies and a consistent way of editing embedded models*".

Final Result: In-Place and Tab Visualization The results of the first evaluation phase showed that the participants strongly favored the *In-Place Visualization* of embedded models, followed by the *Tab-Visualization*. While the In-Place Visualization was rated the best by a significant margin, qualitative feedback of the participants regarding the *Tab* proposal contained additional interesting remarks on the modification and editing of embedded models. Two participants suggested that they would feel more comfortable if modification and editing of embedded models took place in a separate tab. Due to these remarks, the final proposal combines the *In-Place* and the *Tab* approach. Embedded models are visualized and displayed in-place by expanding the Parent Container. A separate tab for each embedded model will be available which should make editing embedded models in an isolated manner easier.

4.4.3. Interaction with the Context Menu

In the initial draft for the Parent Container, no special way of interaction with the container was planned to be part of the concept. However, as illustrated by the previous sections, feedback from the pre-evaluation led to the introduction of a **Context Menu** that provides previously unconsidered additional functionality to the Parent Container. The functions of the Context Menu are explained in the following section.

A button for the Context Menu is visualized by three dots that are located on the top right of the model container. Clicking on it presents the user with three options: **Edit**, **Expand** and **Integrate**. While the *Expand* option behaves similarly to the *Plus Button* (clicking it will expand the Parent Container and show the embedded model in its entirety), the following sections will focus on the *Edit* and *Integrate* options.

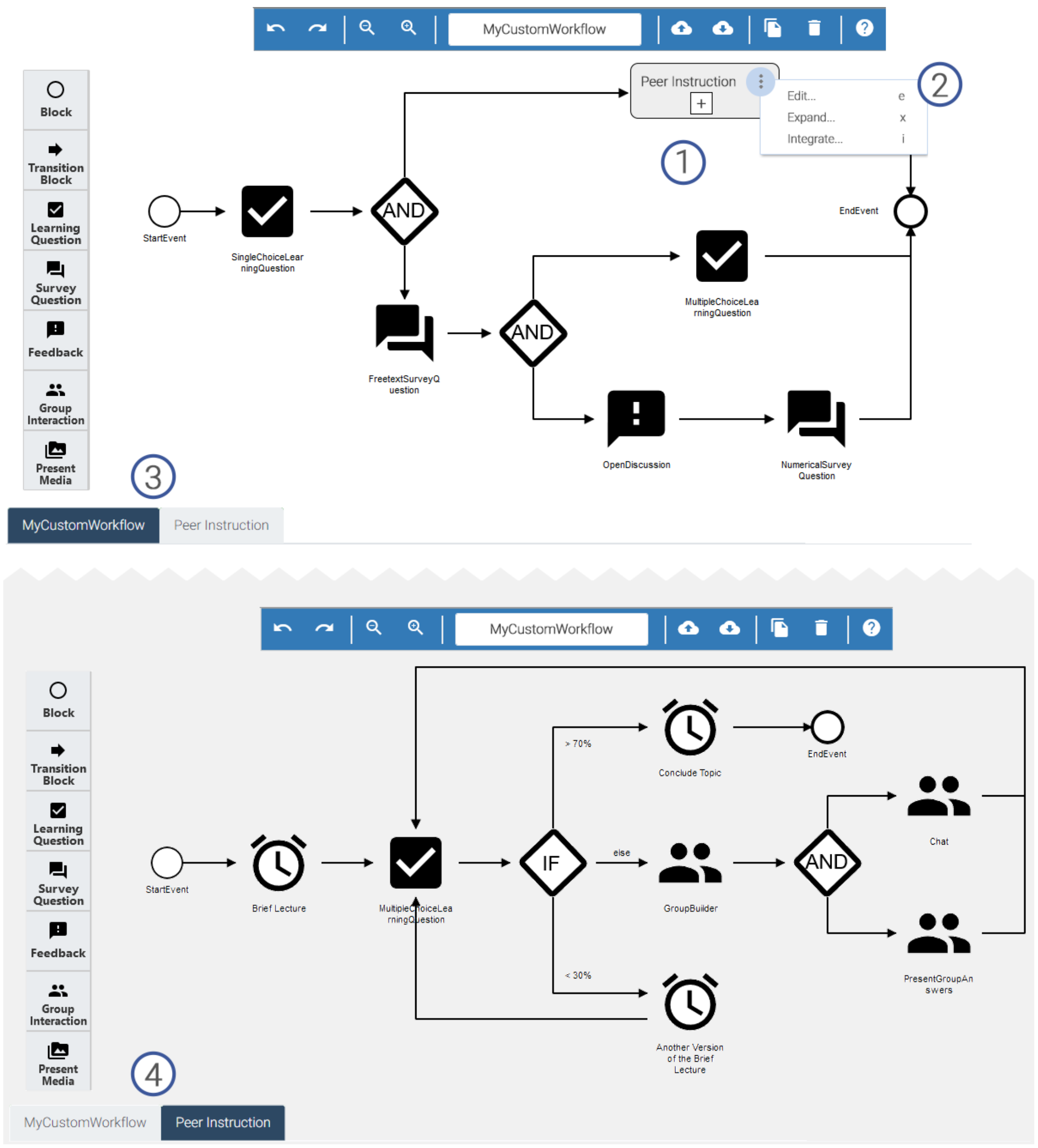


Figure 4.6.: Tab Visualization: Combination of ① *Parent Containers*, ② Context Menu and ③ *Tabs*. The top panel shows the root tab of the workflow *MyCustomWorkflow* that is currently selected. The bottom panel shows that the tab for the embedded model Peer Instruction is currently selected.

Context Menu: Edit Option With the introduction of *Tabs* to the Model Canvas, each embedded model has its own dedicated tab. The Context Menu offers the *Edit* option that enables users to modify embedded model structures. Once the *Edit* option is selected, the embedded model is opened in a separate tab as can be seen in Figure 4.6. All editor functions like the *Side Bar* and the *Properties Panel* will be usable regardless of which tab is currently selected. Once the embedded model was modified, it can optionally be integrated into the parent model.

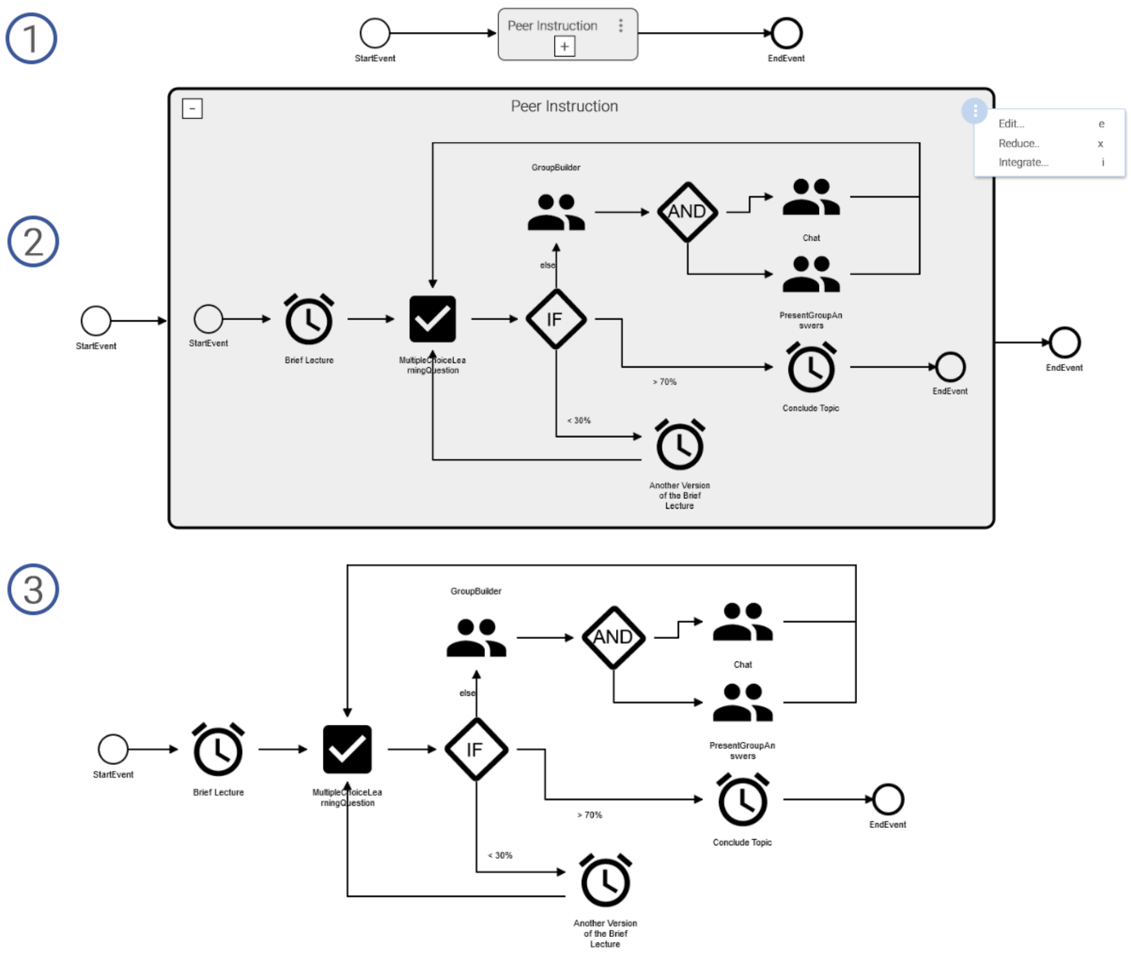


Figure 4.7.: An example for an embedded model that gets integrated to the rest of the parent model. In (1), the initial state of the Parent Container is shown. (2) shows the expanded Parent Container holding an instance of Peer Instruction and also showing the Context Menu. In panel (3), the result of the *Integrate* option is shown. The *Start Node* and *End Node* as well as the Parent Container of the embedded model have been removed.

Context Menu: Integrate Option Feedback from the evaluation led to the introduction of the *Integrate* option. This option causes the embedded structure to be integrated into the rest of the model. After integration, the embedded structure can no longer be retracted or expanded. Instead, the surrounding container of the nested structure is removed along with all of its *Start*- and *End Nodes*. Formerly incoming and outgoing connections to the Parent Container are now directly attached to the respective Function Blocks. An example of the integration process can be seen in Figure 4.7.

Exchangeable and Removable Embedded Models

Finally, the requirement **F9** (*Exchangeable and Removable Embedded Models*) is realized by allowing the user to delete an embedded model like any other block by simply clicking on the Parent Container and then pressing the delete key on their keyboard. However, the proposed concept does not offer a function that explicitly lets users swap embedded models with a single click. Currently, a deleted embedded model can be swapped with another model by first deleting it ant then importing another model. To improve this, another option could be introduced into the Context Menu. Selecting this *Swap Option* could open the *Import Dialog* that then allows users to select a substitution for the embedded model in question.

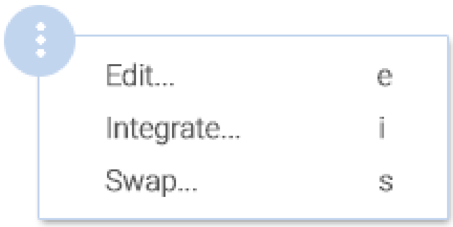


Figure 4.8.: Improved Context Menu: The redundant *Expand* options is replaced with a new *Swap* option.

4.5. Wizard for Parallel Connected Components

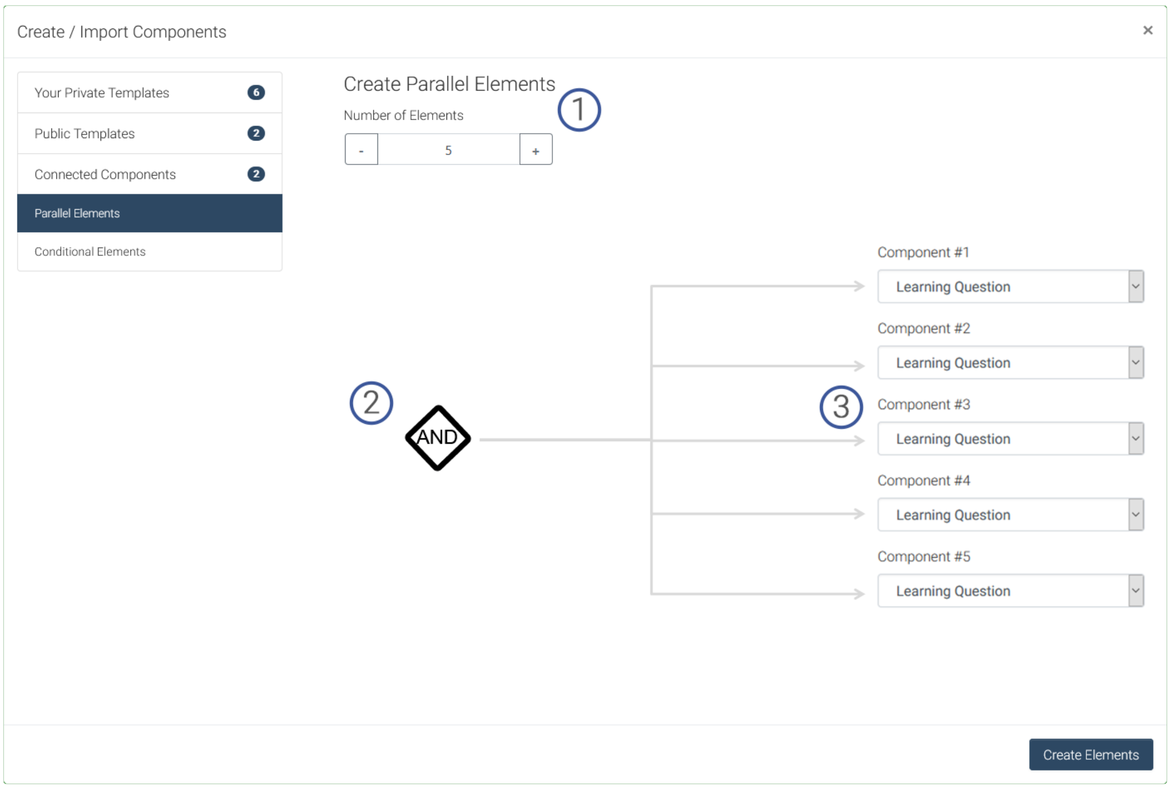


Figure 4.9.: The *Wizard for Parallel Connected Components*: the buttons in (1) allow to adjust the number of elements, (2) shows the Function Block in question (in this case the *AND* block) and (3) visualizes the branches of the *AND* block along with drop-down menus that allow to choose from available Function Blocks.

The last proposal that the participants of the pre-evaluation were shown was a mock-up for a wizard that aims at easing the creation of parallel flows (see Figure 4.9). The general idea is to avoid the manual creation of connections between Function Blocks, as this operation tends to be rather cumbersome and time-consuming. Instead, the proposed mock-up introduces a wizard that is embedded into the earlier proposed *Import Dialog*. Two buttons allow to increase or decrease the number of parallel elements connected to an *AND* block. For each branch of the *AND* block, the user can choose the Function Block to be attached.

Ranging from *1 – useless* to *5 – very useful*, the proposed mock-up was rated with an average score of 4.27 in the first phase and 3.83 in the second phase, meaning that overall, the proposal was rated as useful. Additionally, a significant portion of participants of the first phase rated a similar feature for conditionally connected components (i.e. for the *OR-Block*) as a desired feature (83%) while only 50% did so in the second phase. The decrease of popularity of the wizard led to the decision, that for the time being, the implementation will be attempted for the *AND* block to further evaluate the usefulness of the feature.

However, qualitative feedback of the first phase indicates that participants were not agreeing on the planned placement of the wizard inside the *Import Dialog*, indicating a gap between the design and user expectation. Remarkably, several participants stated that they would expect this functionality elsewhere (i.e. in the *Side Bar*). Furthermore, three participants suggested that an *AND* block should remain modifiable in the same manner after it was placed in the Model Canvas, meaning that clicking on a placed *AND* block should open the *Wizard* again.

4.6. Summary

In this chapter, a concept for an extension of the *stARS* web editor was developed to facilitate the reuse of models and workflows. Suggestions for improving support for more complex and nested model structures were also presented. A UCD-based approach consisting of the creation of mock-ups, prototypes, and a preliminary evaluation with real education professionals was conducted. Using their feedback, mock-ups for new user interfaces for the *stARS* web editor were designed and iteratively improved. The final user interface designs were refined by removing unwanted features from the concept and adding desired but previously missing features. In conclusion, the output of this chapter is a user-approved concept for an extension to the *stARS* web editor that addresses all of the previously identified requirements. All components of this concept will form the basis for the implementation phase, which is presented in the next chapter.

5. Implementation

Chapter 5: Implementation

The previous chapter provided insights to the concept design phase, which resulted in proposals for a variety of features by which the *stARS* web editor could be extended to make it suitable for creating complex models. In this chapter, the implementation phase that followed will be elaborated on. First, the tech stack will be presented, as it represents the conditions under which the implementation took place. Then, the general implementation approach is presented, describing the procedure that turned mock-ups into a running prototype. Furthermore, the implementation of several functions for data transformation is described in detail, focusing on transformation functions that enable model composition. Finally, some insights and the experience with package patching are presented before concluding this chapter with a list of issues that remained unresolved at the end of the implementation phase.

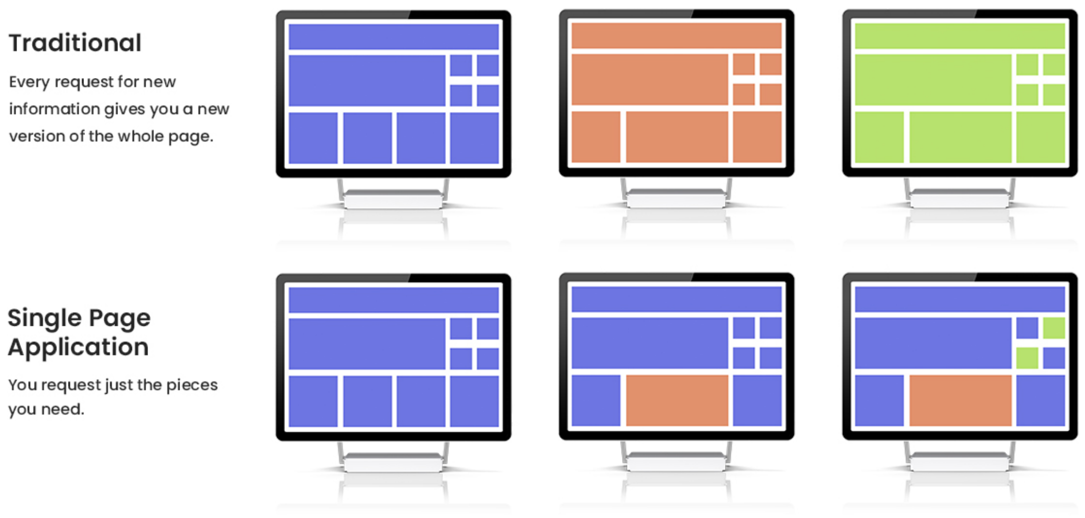

Traditional
Every request for new information gives you a new version of the whole page.
Single Page Application
You request just the pieces you need.

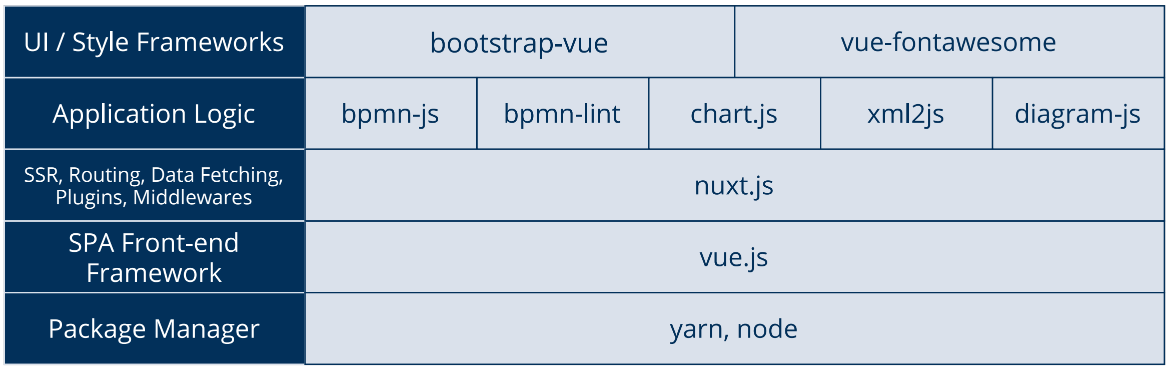
Figure 5.1.: SPA approach vs. the traditional approach to web applications: SPAs only update parts of their UI while the traditional approach relies of different versions of the whole page [57].

5.1. Tech Stack

Single Page Applications (SPA) represent a popular approach for realizing applications that run in a browser and feel like native apps (see Figure 5.1). These applications rely on a single HTML file whose contents change dynamically. SPAs aim at providing the same performance, short loading times and responsiveness that make native apps feel great to use. The *stARS* web editor was likewise implemented as a SPA by using **vue.js** [58], a popular JavaScript SPA framework. vue.js, together with the libraries and frameworks that were utilized to create the *stARS* web editor defined the technical conditions for the implementation phase of this work (see Figure 5.2).

Consequently, the majority of the front-end application logic (including the web editor) is written in JavaScript, HTML and CSS. **yarn** [59], a very popular package manager is used for managing the project structure, build scripts and dependencies. Additionally, the decision was made to include **nuxt.js** [60] on top of vue.js to ease development. nuxt.js offers countless features such as automatically generated routes, server-sided rendering (SSR), a library for asynchronous data fetching as well as countless plugins and middleware that can be enabled and managed via a central configuration file[1]. On top of vue.js and nuxt.js reside all libraries that provide the actual application logic. In [8], different front-end frameworks for developing a BPMN-based web editor were evaluated, and **bpmn-js** [61] was selected because it provided more introductory examples, making initial development easier. bpmn-js offers a modeling canvas and an element palette out of the box. All of these components

[1]More information can be found here: `https://nuxtjs.org/docs/2.x/features/rendering-modes`, last successful access: 2021-02-08



UI / Style Frameworks	bootstrap-vue			vue-fontawesome	
Application Logic	bpmn-js	bpmn-lint	chart.js	xml2js	diagram-js
SSR, Routing, Data Fetching, Plugins, Middlewares	nuxt.js				
SPA Front-end Framework	vue.js				
Package Manager	yarn, node				

Figure 5.2.: The tech stack of the *stARS* web editor.

are customizable to a certain degree. Other important libraries include **xml2js** [62] for data transformations between XML and JSON and **bpmn-lint** which used to define custom linting rules for the models created with the *stARS* web editor. Finally, special libraries are included that maintain a consistent UI style. *bootstrap* [63] provides a responsive grid layout that looks and feels adequate on various devices independent of screen resolution and size. For a seamless integration with vue.js, **bootstrap-vue** is used. *font-awesome* [64] is an icon font that provides vector-based pictograms and symbols. An integration with vue.js is provided by **vue-fontawesome**. Furthermore, several other smaller libraries make up the rest of the tech stack, including (but not limited to) a library for toggle buttons, a multiselect component, a library for generating universal unique identifiers, a library for creating QR codes, and a library for decoding JSON Web Tokens.

5.2. General Approach

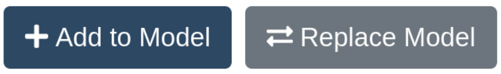


Figure 5.3.: The buttons located at the bottom of the Import Dialog will serve as an example compact enough to illustrate the implementation approach and source code transformations.

At the beginning of the concept finding phase, the mock-ups used for a user-centered evaluation (as described earlier in Section 4.1.1) were created in anticipation of the implementation phase. The mock-ups consisted of HTML and CSS code that was hoped to be reused in the implementation phase, speeding up the process. A major concern regarding the reuse of this code was whether the layouts proposed in the mock-ups would integrate themselves seamlessly into the rest of the existing application. Surprisingly, the intended layout of all mock-ups was deemed *compatible* by just inserting the mock-up source code without modifications into the existing codebase. However, the resulting code was unacceptable in terms of coding conventions and code quality, as it introduced a second style of defining layouts to the codebase. While the existing code avoids using div-containers and explicit bootstrap CSS classes and instead leverages corresponding vue.js components included with bootstrap-vue, the mock-up code relied on *explicit* div-containers and bootstrap CSS classes. Therefore, several transformations were necessary to make the mock-up code compatible with the rest of the codebase. Explicitly defined div-containers were replaced with their respective bootstrap-vue components. A comparison between the mock-up code for the button group seen in Figure 5.3 and and its final implementation code can be seen in Listing 5.1 and Listing 5.2 respectively. Besides performing code transformations, the actual application logic had to be implemented. In the example of the button group, this meant implementing adequate callbacks (that allow to either add a template to the rest of the model canvas or to replace all contents in the model canvas) and defining guards that prevent users from being able to click the buttons. For example, the buttons should not be clickable if no template was selected. This behavior was realized by leveraging vue.js props. In essence, props are pieces of data that control the state of components.

```
<div class="modal-footer">
  <div class="row">
    <div class="col-lg-12">
      <a class="btn btn-primary" href="#">
        <i class="mdi mdi-plus"></i>Add to Model
      </a>
      <a class="btn btn-primary" href="#">
        <i class="mdi mdi-file-replace-outline"></i>Replace Model
      </a>
    </div>
  </div>
</div>
```

Listing 5.1: Source code of the button group in the mock-up. This code was generated with Pingendo and uses conventional div-containers and bootstrap CSS classes.

```
<template #modal-footer>
  <b-btn variant="primary"
         :disabled="modelTemplate === undefined"
         @click="addTemplateToModel(modelTemplate)">
    <font-awesome-icon icon="plus" />Add to Model
  </b-btn>
  <b-btn variant="secondary"
         :disabled="modelTemplate === undefined"
         @click="loadTemplate(modelTemplate)">
    <font-awesome-icon icon="exchange-alt" />Replace Model
  </b-btn>
</template>
```

Listing 5.2: Source code of the button group in the final implementation. The div-containers have been replaced with dedicated bootstrap-vue components. Furthermore, callbacks (denoted with **@click**) that realize application logic have been added. A boolean prop (denoted with **:disabled**) guards the buttons' clickable state.

In summary, the strategy for implementing each new UI component consisted of the following steps:

1. Create a new vue.js component.

2. Insert the HTML and CSS scaffolding of the mock-up and validate the layout.

3. Transform bootstrap related CSS classes to bootstrap-vue components.

4. Implement the expected application logic.

Additional Design Decisions

Some additional design decisions were made that concern multiple proposed UI components to better address the previously mentioned usability heuristics. The usability heuristic *UH 5: Error prevention* proposes that user interfaces have to be designed in a way that prevents users from encountering errors. Therefore, before each critical operation, a confirmation dialog is shown to the user that explains the consequences of the said operation. This is the case for adding a model to the canvas, replacing the model, resetting the canvas or integrating an embedded model.

Furthermore, failed operations are indicated by toasts, a mechanism that was evaluated during the pre-evaluation by a different student effort. For example, this is the case for when the integration of an embedded model is not possible. The user is informed by a toast that states the concrete reason as to why the model can not be integrated. This decision was made to follow the usability heuristic *UH 9: Help users recognize, diagnose and recover from errors*.

Following this approach, first the *Export Dialog*, the *Save Dialog* and then the *Import/Load Dialog* were implemented successfully. While the implementation of the *Export Dialog* and the *Save Dialog* did not require immense efforts (as predicted by **FR1** in Table 3.6), implementation of the functions accessible from the *Import/Load Dialog* offered various challenges.

5.3. BPMN Schema and Data Transformations

A big part of the implementation phase was spent to understand the data format in which BPMN models are managed during runtime. This was necessary because some functions that were proposed by the mock-ups implied the requirement for heavy data manipulation behind the scenes. Earlier in Section 4.3, the functions “Add to Model” and “Replace Model” (in the following referred to as **ATM** and **RM** respectively) were described, which are both accessible from the *Import/Load Dialog*. While the existing RM function was reused with close to no modifications, the implementation of ATM required more effort and a deeper understanding of the underlying format of BPMN.

The existing libraries made it possible to modify individual elements of the loaded model. For example, it was possible to add new elements to the existing model or to change existing model properties programmatically with the help of code. This way of model modification is perfectly suitable for smaller model adjustments (such as changing individual property values) and therefore is utilized for similar purposes in many places in the codebase of the *stARS* web editor. In the context of a model embedding process, however, entire models consisting of potentially dozens of elements were to be added to existing models. It was quickly determined that the APIs offered were not suitable for this undertaking. An implementation using the existing APIs would require to parse the whole model to be added and to generate a list of programmatic instructions from the parsed model, which then would have adapted the model based on the API of bpmn-js. While this is technically feasible, it represents a very complex process. Therefore, a different solution approach was taken.

BPMN models are commonly defined in XML. The web editor is capable of rendering any valid BPMN XML file as a model in the canvas. During runtime, the web editor maintains this XML representation of the model currently residing in the canvas. One approach concentrated on the theory that it might be possible to modify the existing XML directly to reflect bigger, more complicated changes like embedding (i.e. adding) a model. To understand the format, experiments were carried out with the editor provided by **bpmn.io** [40], which also uses bpmn-js to manage BPMN models during runtime. An export function that allows to download models as an XML file was used to especially investigate, what happens if

- subprocesses are added to the data structure,
- parts of the model are duplicated,
- parts of the model are deleted, or
- attributes of model elements are modified or deleted.

These observations allowed to study the structure of the resulting XML files more closely. In general, a BPMN XML file consists of two major parts: The **Process Object Definition** describes the logical structure of the model. All function blocks and other objects along with transitions are defined in this section. Every element has a unique id that is used as a reference in the definition of transitions. The process object definition is followed by the **Diagram Object Definition**, which is responsible for the graphical representation of the objects defined above. It contains definitions for all shapes, edges and labels and references the elements of the process object by their ids. Figure 5.4 shows a small example workflow for which in Listing 5.3, its XML definition is provided, illustrating the data schema in detail.

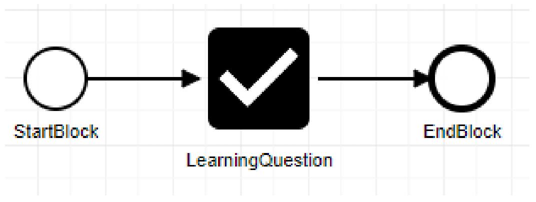


Figure 5.4.: Small example of a workflow with one *LearningQuestion*.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <bpmn2:definitions id="sample-diagram">
3    <!-- Various namespace definitions omitted for brevity -->
4    <!--Definition for the Process Object -->
5    <bpmn2:process id="Process_1" isExecutable="false">
6      <bpmn2:startEvent id="SE_1" name="StartBlock">
7        <bpmn2:outgoing>SF_1</bpmn2:outgoing>
8      </bpmn2:startEvent>
9      <stars:learningQuestion id="LQ_1" name="LearningQuestion">
10       <bpmn2:incoming>SF_1</bpmn2:incoming>
11       <bpmn2:outgoing>SF_2</bpmn2:outgoing>
12     </stars:learningQuestion>
13     <bpmn2:sequenceFlow id="SF_1" sourceRef="SE_1" targetRef="LQ_1" />
14     <bpmn2:endEvent id="EE_1" name="EndBlock">
15       <bpmn2:incoming>SF_2</bpmn2:incoming>
16     </bpmn2:endEvent>
17     <bpmn2:sequenceFlow id="SF_2" sourceRef="LQ_1" targetRef="EE_1" />
18   </bpmn2:process>
19   <!--Definiton for the Diagram Object-->
20   <bpmndi:BPMNDiagram id="BPMNDiagram_1">
21       <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Process_1">
22       <bpmndi:BPMNShape id="SE_1_di" bpmnElement="SE_1">
23         <dc:Bounds x="258" y="240" width="36" height="36" />
24         <bpmndi:BPMNLabel>
25           <dc:Bounds x="250" y="283" width="52" height="14" />
26         </bpmndi:BPMNLabel>
27       </bpmndi:BPMNShape>
28       <bpmndi:BPMNShape id="LQ_1_di" bpmnElement="LQ_1">
29         <dc:Bounds x="365" y="223" width="70" height="70" />
30         <bpmndi:BPMNLabel>
31           <dc:Bounds x="357" y="300" width="87" height="14" />
32         </bpmndi:BPMNLabel>
33       </bpmndi:BPMNShape>
34       <bpmndi:BPMNEdge id="SF_1_di" bpmnElement="SF_1">
35         <di:waypoint x="294" y="258" />
36         <di:waypoint x="365" y="258" />
37       </bpmndi:BPMNEdge>
38       <bpmndi:BPMNShape id="EE_1_di" bpmnElement="EE_1">
39         <dc:Bounds x="512" y="240" width="36" height="36" />
40         <bpmndi:BPMNLabel>
41           <dc:Bounds x="507" y="283" width="47" height="14" />
42         </bpmndi:BPMNLabel>
43       </bpmndi:BPMNShape>
44       <bpmndi:BPMNEdge id="SF_2_di" bpmnElement="SF_2">
45         <di:waypoint x="435" y="258" />
46         <di:waypoint x="512" y="258" />
47       </bpmndi:BPMNEdge>
48   </bpmndi:BPMNDiagram>
49 </bpmn2:definitions>
```

Listing 5.3: Example of a BPMN XML file. The listing illustrates the two main parts of a BPMN file, the process object (lines 5 - 18) and the diagram object (lines 20 - 48).

Ultimately, the goal was to replicate the transformations that the XML file underwent in custom code that would not be as complex as code that utilizes the APIs provided by bpmn-js. The data format represented the common ground on which new features that manipulate the model data had to be built around. After further experimenting with the the format itself, the following implications for every function that intends to manipulate model data were derived:

1. Data modification requires updating *both* the process object definition and the diagram object definition.
2. Every element has to have a unique id. Duplicate ids (as they could occur on element duplication or model embedding/adding) would cause errors in model visualization.
3. The link between elements in the process object definition and the diagram object definition has to be maintained.
4. Subprocesses are added as a special element to the process object. This element has the same structure as any other process object, containing function blocks and other elements.
5. Subprocesses require a special element in the diagram object definition. Without this element, the subprocess is visualized without a surrounding box.
6. The graphical representation of elements in a subprocess are simply added to the diagram object definition (without nesting these in a special object as opposed to the process object definition for subprocesses)

The ATM function should embed a selected model into the existing model, resulting in a parent-child relation between the two models. From a data format perspective, it became clear that leveraging the BPMN subprocess semantics would be the optimal solution. Some tests prior to the implementation confirmed that the *stARS* web editor would out-of-the-box visualize XML files that contain subprocesses correctly. However, as illustrated by the implications above, implementing the data manipulation routines ensuring the correct format turned out to be a challenge.

In the following, the XML representation of a model currently loaded in the model canvas will be referred to as the *target*. At a first glance, adding a new model (i.e. a *source*) to the target seems like a trivial task. A naive implementation would simply copy the process object of the source to the target, along with the diagram object. However, this approach would suffer from several deficits. For once, name and id clashes would occur if the same source is added multiple times to the target. These clashes would cause the diagram to be displayed incorrectly or even crash the application, as the identifiers for all elements in the model have to be unique.

Furthermore, the bpmn-js framework would not be able to recognize the added source as a subprocess (and automatically visualize it as such), because the source has to be added as a special **bpmn2:subprocess** element to the process object of the target. In addition to that, a definition for the graphical representation of the target (i.e. a shape for the subprocess) has to be manually created inside the diagram object of the target. Otherwise, the contents of the subprocess will be visualized without a surrounding box. The bounds (size, x- and y-coordinates) of this box and the displacement of other model elements in the target have to be calculated manually.

Some of these issues are hard to tackle if both the target and source XML strings are processed as-is (e.g. without parsing them first). For example, merging process object strings manually would require to detect the start and end of specific tags, which can be difficult. The same is true for manipulating attributes of individual nodes in an XML string, because the individual node has to be found first. Such manipulation of the XML structure becomes easier if the string is parsed into an object tree that can be traversed systematically. This greatly increases the possibilities to alter the model.

However, once an object tree was created, it becomes difficult to modify shared attributes. For example, renaming the ids of all elements in a parsed object tree would require to identify all the nodes (in the process object and diagram object) that share this attribute. The raw XML string however can be searched with a regular expression to alter the id globally.

Therefore, the approach that was taken to solve these issues combines both the advantages of working with a raw string and working with a parsed data structure. Figure 5.5 illustrates the algorithm by which the ATM function merges a source model to the target model.

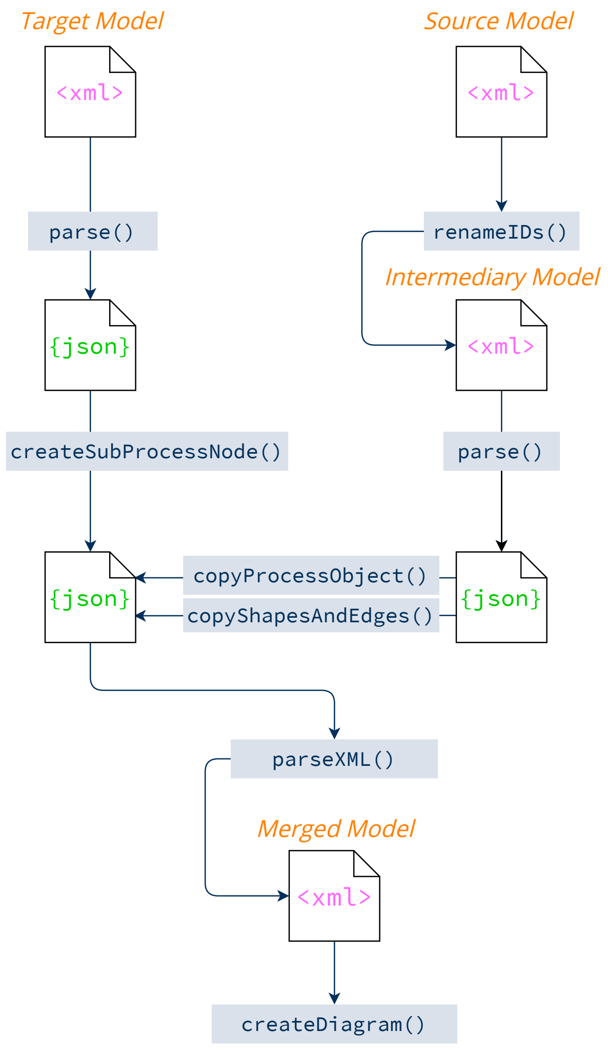


Figure 5.5.: Algorithm of the ATM function.

parse()
Parses any given XML to a JavaScript object tree.

renameIDs()
Identifies all elements in the source process objectB and then performs a search and replace operation on the source XML string to avoid name clashes.

createSubProcessNode()
Creates a parent node for subprocesses in the target process object (if it does not exist yet).

copyProcessObject()
Copies the process object of the source to the subprocess node that was created in the target process object.

copyShapesAndEdges()
Copies shapes and edges of the source diagram object to the target diagram object. Creates also the shape definition for the subprocess and updates names and labels accordingly.

parseXML()
Parses the merged JavaScript object tree back to a valid BPMN XML.

createDiagram()
Uses the resulting XML of the previous step to create its graphical representation in the model canvas.

5.4. Compatibility with the *stARS* Execution Engine

After tackling the aforementioned challenges, the introduction of subprocesses was realized in a straight-forward manner. Thanks to the already existing visualization mechanisms of bmpn-js for subprocesses, it was sufficient to ensure that the underlying data had the correct structure. Once a subprocess is placed on the canvas, it behaved like other elements and could be moved around, connected to other elements, edited, copied, and deleted. Selecting a subprocess even opened the properties panel with only minor additional modifications needed. However, while the models behaved as expected in the *stARS* web editor, once an instance of the model was created that contained subprocesses, it was not executable. The transformation of the model into a request that the back-end of *stARS* could process properly failed because no implementation was present that transformed the subprocess objects accordingly. Furthermore, the back-end did not support the special subprocess objects, because they are missing in the meta-model. For these reasons, and for the fact that the implemented solution should be embeddable to the rest of the system, (see **NFR1** in Table 3.6), some modifications had to be made to the model data that is sent to the back-end to make it compatible with the execution engine.

One solution to this problem was to transform the XML that is sent to the back-end server in such a way that subprocesses were integrated to the rest of the model. However, the graphical representation of the model which is saved separately is not transformed. The final result would be a scenario that is visualized to the user with all subprocesses in-tact while the underlying data structure that is sent to the server is flat. The visualization in the Control View is then able to show subprocesses as such while the data that resides on the execution engine does not contain any subprocesses at all. When advancing the scenario, all start and end blocks of the subprocess are intentionally skipped.

Fortunately, a function that integrates a subprocess into its parent is already required to be implemented for the Context Menu. The strategy therefore was to realize the function for the Context Menu and then reuse it to transform model data into a for the execution engine usable structure.

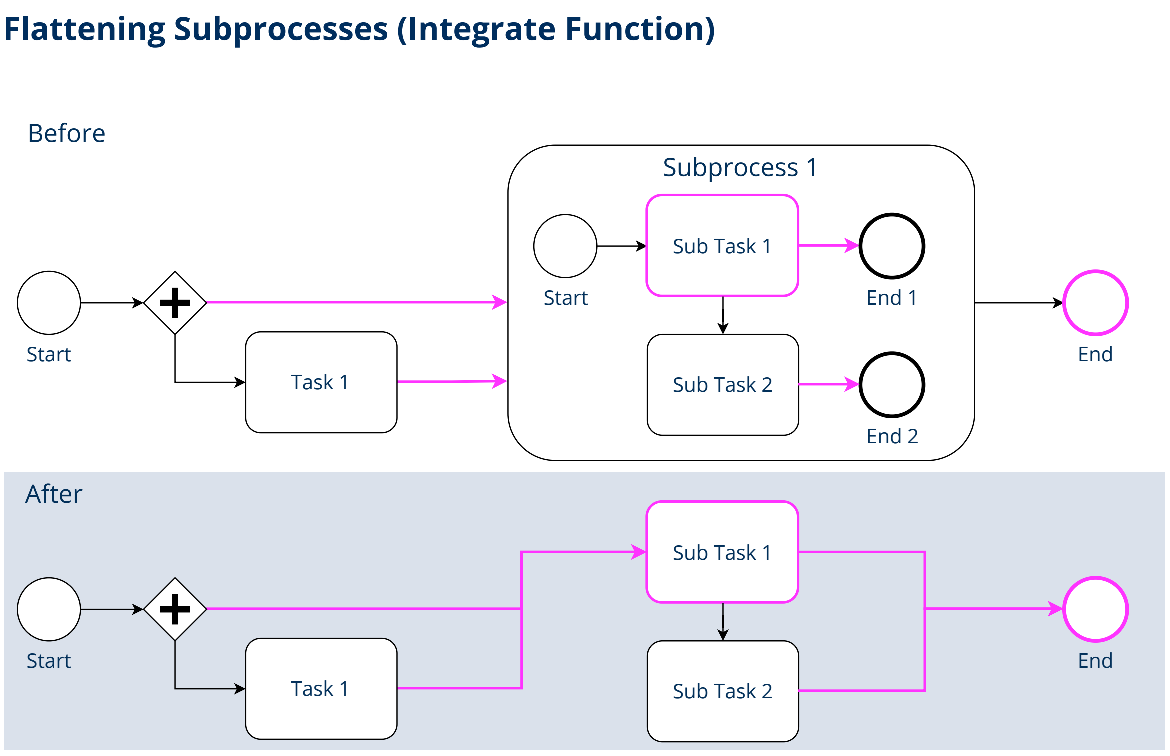


Figure 5.6.: An example workflow before and after it was flattened. The transitions and elements highlighted in pink are subject of change during the flattening. Thus, a way to query these elements from the parsed data structure is needed.

The function to integrate a subprocess flattens the model by removing the parent container, start- and end blocks of the subprocess and by connecting its remaining contents accordingly with the rest of the model. Figure 5.6 illustrates this process with an example. While it is conceptually easy to understand what the feature is intended to do, it turned out to be difficult to implement for various reasons. For once, the framework and libraries that are already part of the codebase did not offer any means to query a parsed object tree for specific elements in the process object or the diagram object respectively. For example, in order to redirect incoming and outgoing transitions, the source and target elements of these transitions had to be obtained in order to modify them. This was not possible with the APIs that the libraries expose.

In addition to that, no convenient way to identify and obtain all transitions that go into (or out of) a specific element was present. Such a function would simplify the code responsible for redirecting existing transitions that enter and exit the subprocess in question. Furthermore, start- and end blocks of the subprocess have to be removed on integration. Identifying these blocks, the transitions attached to them and the target function blocks was not possible without dedicated query methods. Consequently, these query functions had to be written and tested first before implementing the integrate feature, therefore increasing the implementation effort noticeably. A summary of the functions and their purpose can be found in Table 5.1.

Query/Helper Function	**Description**
`collectSubProcesses()`	Returns an array of all subprocess objects contained in the process object of the model currently loaded.
`findElementById(id, processObject)`	Returns the element specified by an **id** in a given **process object**.
`findFlowsForElement(id)`	Returns an array of all sequence flows for a given element referred to by **id**.
`findFlowsForElementsOfType(type)`	Returns an array of all sequence flows for a given element **type**. Used to find all incoming and outgoing transitions of all start blocks and end blocks of a given process object.
`getElementIds(processObject)`	Returns the ids of all elements that are part of a given process object.
`mergeSubProcess(id, po, do)`	Merges all the contents of a subprocess given by **id** with the process object **po** and the diagram object **do** of the parent.
`redirectFlow(flow, target)`	Redirects a sequence **flow**. This means modifying the target of a given transition to point to the given **target**.
`removeElementById(id, processObject)`	Removes an element by **id** from the **process object**.

Table 5.1.: Query and helper functions that were implemented to ease the development process.

Using these helper functions, an algorithm was developed that transforms any given BPMN XML containing a subprocess into a flat XML with equal semantics. The following listing illustrates this algorithm in pseudo-code:

Algorithm 5.1: Algorithm for flattening/integrating a subprocess. Explanatory comments start with the symbol #.

```
input: xml, subprocessID
output: flatXml
begin
  # First, parse the xml as an object tree.
  model ← parse(xml)
  # Gather both the process object and diagram object from the parsed object tree.
  processObject ← getProcessObejct(model)
  diagramObject ← getDiagramObject(model)

  # Get the subprocess object to modify it.
  subProcessObject ← findElementById(subprocessID, processObject)
  # Get all transitions that enter and exit the subprocess.
  subProcessFlows ← findFlowsForElement(processObject, subprocessID)

  # Find all flows that either enter or exit start- and end-events of the subprocess.
  startConnections ← findFlowsForElementsOfType('bpmn2:startEvent', subProcessObject)
  endConnections ← findFlowsForElementsOfType('bpmn2:endEvent', subProcessObject)

  # Merge all elements of the subprocess object with the parent process object.
  mergeSubProcess(subprocessID, procesObject, diagramObject)

  # Find both the entry model element inside the subprocess and the sink element
  # outside the subprocess.
  entryElement ← startConnections[0].outgoingFlows[0].targetRef
  sinkElement ← subProcessFlows.outgoingFlows[0].targetRef

  # Redirect all transitions that flow into the subprocess to point to the entry element
  # of the subprocess.
  foreach flow in subProcessFlows.incomingFlows do
    redirectFlow(flow, entryElement)
  # Redirect all transitions that pointed to end-events inside the subprocess to the
  # sink element outside the subprocess.

  foreach connection in endConnections do
    foreach flow in connection.incomingFlows do
      redirectFlow(flow, sinkElement)

  # Remove the subprocess object from the process object.
  removeElement(subprocessID, processObject)

  # Remove the start and end-events of the non-existent subprocess.
  foreach startEvent in subProcessObject do
    removeElement(startEvent, processObject)

  foreach endEvent in subProcessObject do
    removeElement(endEvent, processObject)

  # Parse the result as XML.
  flatXml ← parse(model)
  return flatXml
end
```

5.5. Patching Frameworks and Libraries

The *stARS* web editor codebase depends on many third-party libraries and frameworks, most of them being open-source projects. The big advantage of open-source libraries is that the code is free to use by anyone and for any project as long as appropriate credit is given. However, because such projects are mostly maintained by volunteers and interested individuals with limited time, some bugs will never get fixed. In the following, a similar situation is explained that was encountered while implementing the ATM function.

A bug in the framework bpmn-js caused subprocesses to be visualized in a strange manner. Expanding, moving and then collapsing the subprocess left the labels of the subprocess elements visible, although they should be hidden when the subprocess is collapsed. Investigating this bug further lead to the conclusion that this behavior had been a known issue in bpmn-js for years, but yet no official fix was provided by the community. Nonetheless, it was possible to identify the exact commit in which the changes were introduced that break the visualization of subprocesses. Therefore, the first idea was to downgrade the version of bpmn-js to avoid the piece of code that breaks the subprocess visualization. This fixed the visualization problem, however many other unintended side-effects were introduced, rendering the web editor virtually unusable. As a result, the implementation switched back to the more recent version of bpmn-js that still contained the bug.

Further investigations on the bug were conducted because the feature in question was deemed too important to leave it out of the final result of the implementation phase. Soon the discovery was made that it was possible to fix the bug manually, however only locally. Whenever the *stARS* web editor was deployed on testing infrastructure, the official version of bpmn-js was used (which does not contain the bug fix), so the bug remained on deployed versions. More research on this topic finally led to a solution, that was specifically engineered for similar situations: patch files. A patch file contains changes that should be applied to a given source code file. In the specific case of bpmn-js, it was possible to generate a patch file with a tool called **patch-package** [65] and incorporate it's usage into the build process. During the build phase, the problematic source code files are dynamically adapted by the contents of the patch file. This avoided the need to maintain a fork of the bpmn-js framework inside the *stARS* codebase, meaning that deployed versions of the editor finally also profit from the bug fix. This solution can be useful to developers who continue work on the *stARS* front-end and encounter similar problems in the future.

5.6. UI Concept Changes

During the implementation phase, some issues occurred that prevented parts of the concept from being implemented as proposed. These issues include small oversights in the concept itself that could impede the practicality of the implementation, and framework limitations. Therefore, to avoid a situation in which the final implementation would suffer from drawbacks that could hinder the evaluation process, changes were made to some of the proposed UI concepts.

Context Menu and Minimize / Expand Button

As mentioned earlier, most of the components provided by bpmn-js can be customized – some easier than others. A customization that turned out to be more difficult than anticipated was the creation of custom menus. Unfortunately, due to these difficulties and limitations of the framework, it was not possible to create both a button for expanding and minimizing the Parent Container (as described in Section 4.4) and the Context Menu (as described in Section 4.4.3). Nonetheless, the functionality the Context Menu intended to provide for subprocesses was implemented – only the place where the functions can be accessed was changed. As mentioned before, bpmn-js has great support for subprocesses and automatically opens a Properties Panel when clicking on the subprocess container. Adapting the panel's behavior by adding custom UI elements was possible. Therefore, expanding and retracting subprocesses is now instead made possible via the Properties Panel. Additionally, a button is provided that allows to integrate the subprocess into the rest of the model.

The "Edit" option was another feature that the Context Menu was intended to provide. It should allow users to open an embedded scenario in a separate tab in which it can be modified. This feature was instead implemented differently. Now, every time a model is embedded, the implementation automatically detects all subprocesses and creates a tab for each at the bottom of the modeling canvas. The user can choose one of the tabs at the bottom of the canvas and modify the corresponding subprocess.

Export Dialog

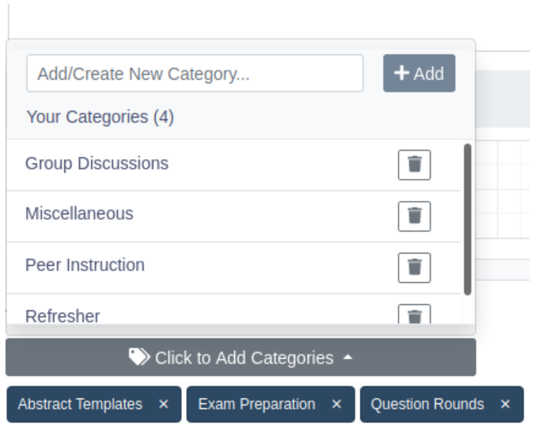


Figure 5.7.: The improved Category Selector.

Most of the features related to the export dialog were implemented without any problems. However, some minor changes to the concept of private templates and template categories were required to ensure that these functions could be used in a workable way. For example, it was not possible to remove a template or category once it had been created. The concept did not provide any form of mechanism to delete categories or private templates. The lack of such a feature was discovered during the implementation of Template Repository and Template Preview Cards, where a *Delete Button* was introduced for debug purposes. It was soon realized that end users would definitely need this feature, so the buttons were kept and integrated into the rest of the UI. Furthermore, the appearance of the category selector has changed iteratively during development. The version of the category selector that was envisioned in the mock-ups consisted of a simple box in which categories were visualized as tags. However, it was soon discovered that vital features like auto-completion and suggestions for categories were missing. With the original design, users would have no idea which categories already exist. The improved design (see Figure 5.7) therefore offers a scrolling list of categories that can be filtered by entering a search term. Entering a category name that does not exist and clicking the *Add* button will create the category and automatically add it to the template.

The backend server code responsible for persisting templates also needed to be modified to support deleting template categories. A bug caused associations between categories and templates to not be persisted correctly. When a category was deleted, the backend code did not remove the category from the template. Retrieving a template with a missing category caused the server to crash. Fortunately, this problem was detected early and fixed with the support of Robert Peine, a student that currently maintains the backend of *stARS*.

Import Dialog

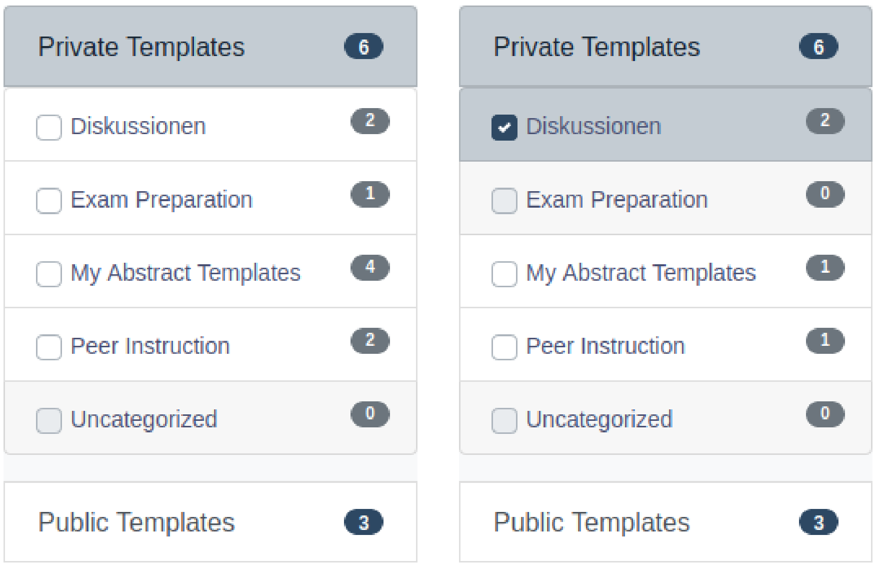


Figure 5.8.: The updated *Sidebar* behavior: Selecting a category dynamically updates the number of available templates in the other categories. *left*: before selection, *right*: after selection.

A dialog has been proposed in Section 4.3 which serves as a template repository. The *Sidebar* in the dialog displays public and private categories. In the actual implementation, this feature has been improved. When multiple categories are selected, the number of templates next to the category name changes accordingly. The same applies if a search term is entered in the search bar. This change was introduced because static numbers next to categories could potentially confuse users. When selecting multiple categories, the number of displayed templates is *reduced*, i.e. selecting multiple categories follows an *and-wise* logic. Only templates that belong to all selected categories will be displayed to the user. Consequently, the number of templates that apply to the user's current selection now changes dynamically.

5.7. Open Issues

Creating an embeddable solution that fits into the existing codebase has been one of the highest priorities. However, not all features could be implemented in a seamless way. Due to either technical limitations or time constraints, several issues remained open until the end of the implementation phase. However, with more time available, all of these problems can be solved in future efforts.

Integrate Function, Tabs and Undo History

The introduction of the integrate function function did create problems regarding the command stack and the undo and redo functions. The history of commands is managed by observing one main XML structure that represents the model. However, the introduction of tabs led to the need to manage multiple XML structures in parallel (one for each tab and one root XML structure for the main tab). This causes problems for the undo and redo system, which has therefore lost some of its functionality. The same is true for the ATM function, as adding a model can not be undone in the current implementation. After a model is added, the command history is lost because the parent model is completely re-created after the import operation. Future work therefore should concentrate on restoring the undo and redo function properly, for example by introducing multiple undo- and redo stacks (one for each XML).

Back-end support for subprocesses

As described previously, additional implementation efforts were necessary to make models containing subprocesses compatible with the back-end of *stARS*. An open issue therefore is to make the back-end compatible with subprocesses in general by introducing the subprocess into the meta-model. This solutions would reduce the complexity introduced to the codebase of the web editor greatly.

Performance and Responsiveness

Many of the implemented features require recalculating and redrawing models in the editor, which can be costly on the client-side hardware. One area of the application where this is especially noticeable is switching between tabs inside the canvas. On slower devices, an unpleasant delay could potentially occur before the model is fully drawn in the canvas. Here, performance improvements could be achieved by managing multiple instances of the canvas at the same time (one for each embedded model), therefore avoiding complete redraws of models. However, implementing this feature would be much more difficult, as syncing changes to diagrams in tabs is still an issue.

Another solution could be to listen on any changes that are made to embedded models. There exist several events that are triggered whenever the user edits the model (regardless of whether it is an embedded model or the main model). Listening to these events and recording the changes that were applied to the model could allow to use the API of bpmn-js to create instructions that represent these changes. The recorded "delta instructions" would then be re-applied to the embedded model that resides in the main tab once the user switches back to the main tab. This technique would likewise avoid the complete redraw of models. However, a feasible strategy that allows to record and translate model modifications into code has to be developed.

Furthermore, displaying many preview renderings of templates simultaneously in the template repository can cause performance issues. Especially loading times can suffer on lower-end devices, because model rendering is done sequentially and in some cases blocks the user interface until the rendering is complete. One solution for this problem would be to persist a pre-rendered SVG image of the model on the back end server, which is then used to render the preview of the template in the template repository. This would drastically improve loading times.

Wizard for Connected Components

A wizard that eases the creation of connected components was initially proposed and part of the concept. Due to time constraints and the additional implementation efforts needed to introduce tabs and to make subprocesses compatible with the back-end, considerations had to be made about the remaining time to implement the wizard and its benefit to the users and the evaluation. The wizard is an isolated feature, as no other features of the concept depend on its existence. More precisely, the wizard only represents an extension that should speed up the modeling process and no functional or non-functional requirements that were defined by this thesis depend on it. Therefore, the assumption was made that the evaluation phase would not suffer in terms of quality and meaningfulness if the wizard was not present, and its implementation was postponed indefinitely. However, as the wizard was rated positively in the pre-evaluation, the concept is still interesting and should be investigated further in the future.

5.8. Summary

This chapter described the implementation phase of this work that consisted of realizing the proposed UI components (*Save, Export, Import and Load*-Dialog), support for embedded models, and tabs for visualizing only parts of a more complex model. First, the tech stack and initial conditions were described, before outlining the general approach that was followed to implement the UI components. Then, insights to the data format of BPMN and necessary data transformations were provided to illustrate, how model data has to be manipulated to reach certain goals. This was additionally complemented by outlining algorithms for model composition and model flattening. Then, necessary changes to the UI concept were described, because not all features were implemented as initially proposed. Finally, this chapter concluded with a summary of issues that remained open until the end of the implementation phase.

In conclusion, the result of the implementation phase meets the majority of functional requirements that were earlier defined in Table 3.6, because most of the required functional features were implemented completely. Furthermore, the non-functional requirement of an embeddable solution (**NFR1**) was mostly met, (with a few exceptions), because the implementation represents a seamless extension of the already existing codebase. For now, the implementation has to stand the test against the remaining two non-functional requirements (**NFR2** and **NFR3** that concern an increase of usability and intuitiveness of new features respectively). Therefore, the upcoming chapter will elaborate on the evaluation of the implementation by real end-users.

6. Evaluation

Chapter 6: Evaluation

The previous chapter illustrated the methodology, difficulties, changes to the concept and results of the implementation phase. In the next phase of this thesis, the implementation had to be evaluated by real end-users. This chapter covers the evaluation process that took place in form of a user study. First, the evaluation methodology is described before details on the modeling task that participants of the study had to solve are discussed. Then, the results of the evaluation and quantitative feedback are presented. This chapter then concludes with a review of open feedback and suggestions for improvement that was provided by the participants of the evaluation.

6.1. Methodology

The implementation phase resulted in a prototype version of the *stARS* editor that was enhanced with many new features. This version provided the vast majority of the functions and capabilities originally envisioned by the mock-ups. In order to draw conclusions between the proposed mock-ups and the version of the *stARS* web editor implemented in this work, the same participants from the pre-evaluation were asked to test the implementation. It was assumed that participants from the previous evaluation would be able to make an informed comparison between the proposed mock-ups and the final implementation results. The overall goal of the evaluation was therefore to find out whether the user's expectations were met by the actual implementation.

Extensive pre-testing has been done to ensure that the evaluation takes place without any technical issues caused by specifics of the user's operating system or browser. The browsers considered include *Firefox 81.0* and *Google Chrome 87.0*, both of which were tested on Windows 10 and Ubuntu 20.04 LTS[1]. As testing progressed, some problems were encountered when using *stARS* with the latest version of the macOS browser *Safari*, which were quickly resolved with the assistance of Tommy Kubica, the initiator and one of the main maintainers of *stARS*. With this, all major operating systems and all major browser vendors were pre-tested and no further problems were found.

In total, **12** individuals tested the prototype by modeling a complex learning scenario. The evaluation was conducted in a very similar way to the pre-evaluation using video conferencing and a Google Forms questionnaire (see Appendix C). The goal was to evaluate all implemented features in terms of functional and non-functional requirements and user acceptance. As described previously, the solution should theoretically increase the usability of existing features (**NFR2**) and newly added features should be intuitive to use (**NFR3**). If the implementation is evaluated positively with respect to these two requirements, the solution can be considered a success. However, it is always possible that the proposed solution or parts of it do not appeal to the end-users after all. In this case, the exact reasons for user criticism and rejection must be identified and documented so that future work can address these weaknesses.

[1]These were the latest versions of Firefox and Google Chrome available at the time of writing

6.2. Task Design

The task that was given to the participants had to be designed carefully to allow for a meaningful evaluation of all features that were realized. The participants were asked to create a specific model on their own with focus on re-using parts of their solutions on the way. The task followed a structure that incentivized participants to seek out for new functions like the *Save Dialog*, the *Export Dialog*, the *Template Repository* and the possibilities to compose bigger models from smaller parts.

More concretely, the main objective for the participants was to create a model of *three consecutive question rounds* that are each separated by a lecture block. The sample solution of the overall task can be seen in Figure B.2. The model to be created deliberately contained repeating but not identical structures. This put participants in a situation where importing a template or having tabs for nested structures can be useful. In the following sections, a preliminary step and the four sub tasks that the participants had to solve will be explained.

Questions about previous experience with stARS	
Question	**Score**
How familiar are you with the stARS Editor user interface? *very unfamiliar = 1, very familiar = 5*	3.67
How familiar are you with the operation of the stARS editor? *very unfamiliar = 1, very familiar = 5*	3.67
How confident are you in your ability to model the ”three rounds of questions” structure shown above using stARS? *very insecure = 1, very confident = 5*	4.25

Table 6.1.: Results regarding statements to the familiarity of participants with *stARS*.

In a preliminary step, participants were asked to rate their familiarity and confidence with *stARS* and the web editor. A summary of statements regarding this is shown in Table 6.1. It can be said that participants were moderately familiar with both the user interface and the operation of the *stARS* web editor. Furthermore, most participants felt confident that they would be able to create a model of three consecutive question rounds.

Task 1: Creation of a template for a question round

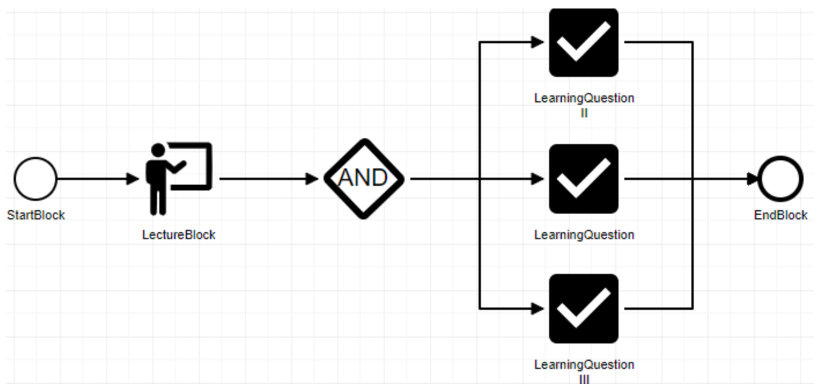


Figure 6.1.: The question round that participants had to model in Task 1.

The first task was a short introduction to the *stARS* web editor. Participants had to attempt to model a simple structure consisting of three questions that are preceded by a lecture block (see Figure 6.1). This reacquainted participants with the canvas and the tools that allow for model manipulation. Once participants were satisfied with their solution, they were asked to create a template with a custom name and an optional description. This meant that participants had to find and use the Save/Export dialog box. Participants were also asked to create a new category and assign it to their fresh new template. Once they saved the template, Task 1 was completed. Figure 6.2 shows the responses received to the questions posed in the first task about UI visualization, the suitability of the tools provided to solve Task 1 and participant satisfaction regarding the end result. Most participants *fully*

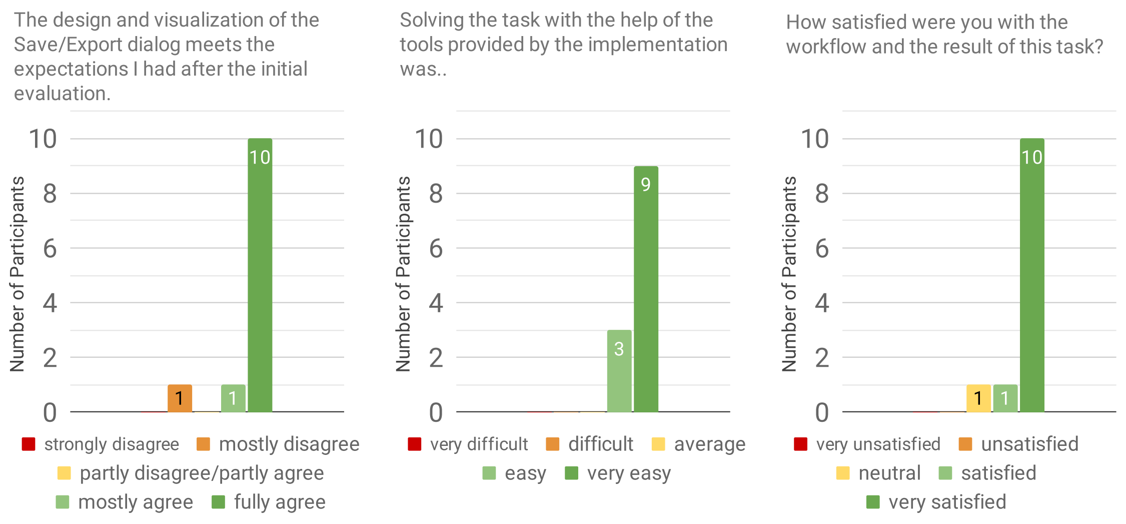


Figure 6.2.: Summary of the evaluation results of Task 1.

agree with the design and visualization of the Save/Export dialog, which means that user expectations were met. In summary, this task was the easiest of the four for all participants, as no one had any problems creating the model and saving it as a template. 9 Participants stated that it was *very easy* to solve the task with the tools that the implementation provided, while 3 other participants felt that it was *easy* to do so. Overall, the majority of participants was content with the overall workflow and result of the first task (10 were very satisfied, one satisfied and one had neutral feelings). An interesting question here was how the improved category selector would perform, since its design is significantly different from the proposed mock-up. Participants had differing opinions on how newly created categories were displayed. Some participants seemed to have overlooked an confirmation toast at the top right of the screen, which serves as visual feedback to the user about the successful creation of a category. In addition, some participants did not notice that their newly created category was immediately selected and visualized as a tag below the category selector. One participant noted that they would have expected checkboxes next to each category in the category list to indicate whether or not a category was selected. Another participant stated that they would not have expected selected categories to disappear from the drop-down list. Instead, they expected selected categories to remain in the list and to be visually highlighted instead. In summary, all participants were satisfied with the way the first task had to be solved. The majority of participants was pleased with the way templates can be created. Furthermore, the category selector (as a component that was heavily modified in comparison to its original design) was still intuitive to most participants with only minor remarks regarding the current selection of categories. A summary of participant statements regarding Task 1 and the creation of templates is given by Table 6.2.

Task 1: Creation of a template for a question round		
Question	**Score**	**Rating**
The design and visualization of the Save/Export dialog meets the expectations I had after the initial evaluation. *strongly disagree = 1, strongly agree = 5*	4.67	☆☆☆☆☆
The possibilities to enrich templates with additional information (name, description and categories) meet my expectations. *strongly disagree = 1, strongly agree = 5*	4.67	☆☆☆☆☆
To me, the operation of the UI elements to be used for creating categories was... *very unintuitive = 1, very intuitive = 5*	4.75	☆☆☆☆☆
The way the options for resetting function block attributes and converting concrete function blocks to abstract function blocks are provided meets my expectations. *strongly disagree = 1, strongly agree = 5*	4.67	☆☆☆☆☆
Solving the task using the tools provided by the implementation was ... *strongly disagree = 1, strongly agree = 5*	4.75	☆☆☆☆☆
How satisfied were you with the workflow and the end result of this task? *very unsatisfied = 1, very satisfied = 5*	4.75	☆☆☆☆☆

Table 6.2.: Rating of statements regarding the usability of the implementation for Task 1. (☆= very bad, ☆☆= mostly bad, ☆☆☆= neutral, ☆☆☆☆= good, ☆☆☆☆☆= very good)

Task 2: Model Composition

In the second task, the participants had to compose a model by reusing the template that they created in Task 1. The intermediary structure of the model to be built can be seen in Figure B.1. The structure consists of the same question round three times in a row. For this, the participants were asked to add the template *twice* into the modeling canvas. Participants then had the task to try and create the remaining question round via duplication (e.g. by copying and pasting the question round into the canvas). This task had participants explore the *Import/Load* dialog and its *Add to Model* function that allows for model composition. One of the challenges for some participants was finding how to access the *Im-*

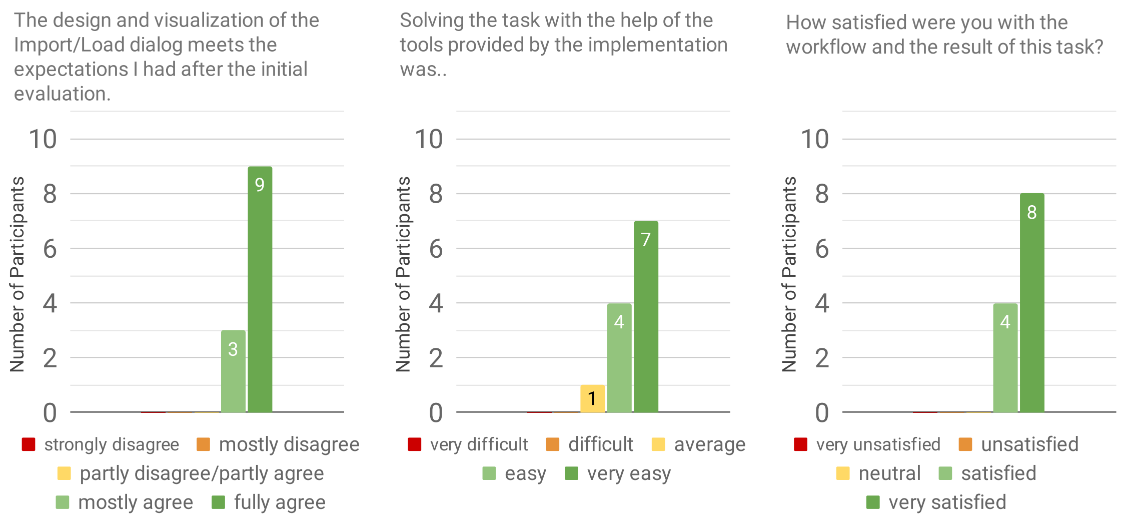


Figure 6.3.: Summary of the evaluation results of Task 2.

port/Load dialog. As a result, several participants stated that the icon in the Main Menu was not intuitive enough and that the tooltip below it gave them the information they needed instead. Once the *Import/Load* dialog was found, 8 participants used the Sidebar on the left to select the category they created in Task 1 to filter the available templates. Overall, participants indicated that they were able to locate the template they were looking for very quickly. In summary, participant's expectations regarding the design and visualization of the *Import/Load* dialog (see Figure 6.3) were mostly met. One participant had only *average* feelings about the tools provided to solve this task, and justified this by not being able to undo the addition of a question round to the model. As mentioned in Section 5.7, the undo/redo function should also work seamlessly when embedding models. However, this is still an open issue that needs to be resolved in the future.

At the beginning of the evaluation, doubts were expressed about the comprehensibility and intuitiveness of the ATM function. It was suspected that the explanatory text within the confirmation dialog alone would not be sufficient to explain the function to inexperienced users. Therefore, another question was added to the questionnaire after the first three individuals had participated in the evaluation. The question asked the remaining 9 participants if an explanatory illustration of the ATM feature would help them better understand the purpose of the feature. Participant feedback on this question was very controversial, with some participants fully agreeing with the suggestion, while others disagreed with the suggestion altogether (see Table 6.3). Further analysis of the collected data showed that some participants who were very familiar with both the UI and the operation of *stARS* would dismiss this suggestion. Some participants who expressed that they were not as familiar with the user interface and operation would agree to the suggestion. However, no clear cut can be made between inexperienced users that need support in understanding and expert users who do not need the extra assistance.

P# 4	**P# 5**	**P# 6**	**P# 7**	**P# 8**	**P# 9**	**P# 10**	**P# 11**	**P# 12**
UI Familiarity: *very unfamiliar = 1, very familiar = 5*								
5	4	4	3	2	3	4	2	4
Operation Familiarity: *very unfamiliar = 1, very familiar = 5*								
5	4	4	4	2	3	4	2	4
Modeling Confidence: *very insecure = 1, very confident = 5*								
5	5	5	5	4	3	4	4	3
Feelings towards explanatory illustration: *strongly disagree = 1, strongly agree = 5*								
1	1	3	1	3	4	3	4	3

Table 6.3.: Comparison between user confidence and experience with *stARS* vs. feelings towards an explanatory illustration for the ATM function.

One possible interpretation of the data could lead to the idea of treating the additional visualization for the ATM function as *optional* information. An idea to further investigate the usefulness of this feature could be to show the visualization on the first use of the function and to hide it in subsequent uses while providing a button that allows to show the visualization when needed. Another idea would be to integrate an explanation of the ATM function into the already existing explanatory overlay that is shown on the first launch of the editor.

After having added a question round to the model, most participants were quick to note that an embedded model can be interacted with just like any other element in the canvas. Participants attempted to select an embedded question round and reposition it in the canvas. Here, many participants indicated that they were not satisfied with the way the selection of the parent container for the question round worked. Participants noted that selection was only possible when they clicked in an area near the boundaries of the Parent Container. Clicking on another area within the container did not result in feedback to participants. At first, several efforts were made to solve this problem alone before a viable solution was provided by individuals at the bpmn-js community forum[2]. With this help and the support of Tommy Kubica, this problem was fixed during the evaluation phase by implementing a custom event handler that responded appropriately to clicks. The final 3 participants then benefited from this solution and did not notice any difficulty clicking and moving Parent Containers (i.e. embedded question rounds) in the canvas. When the second question round was added, some participants struggled with the remaining space on the canvas. Some managed to find the zoom function to create more space for the third question round while others were pointed at the zoom function by the investigator.

[2]The forum thread can be found here: `https://forum.bpmn.io/t/select-subprocess-when-clicking-inside-of-it/5825`, last successful access: 2021-02-23

Overall, it can be stated that participants were happy with the implementation during Task 2. The *Import/Load* dialog was praised in terms of its visualization and how fast desired information can be found. Participants only had conflicting opinions about the aforementioned explanatory illustrations for certain functions. Further investigations have to be made in order to determine whether such a feature would make it easier to new users to understand the ATM function. Participant statements regarding Task 2 are summarized in Table 6.4 below.

Task 2: Model Composition		
Question	**Score**	**Rating**
The design and visualization of the Import/Load dialog meets the expectations I had after the initial evaluation. *strongly disagree = 1, strongly agree = 5*	4.75	☆☆☆☆☆
The behavior of the controls in the Import/Load Dialog meets my expectations. *strongly disagree = 1, strongly agree = 5*	4.83	☆☆☆☆☆
I was able to find the template I was looking for very quickly using the Import/Load Dialog. *strongly disagree = 1, strongly agree = 5*	4.83	☆☆☆☆☆
I find the behavior of the "Add To Model" function to be... *very unintuitive = 1, very intuitive = 5*	4.75	☆☆☆☆☆
The visualization of embedded templates is well done and meets my expectations. *strongly disagree = 1, strongly agree = 5*	4.58	☆☆☆☆☆
An explanatory visualization for the "Add To Model" function would have helped me understand this function. *strongly disagree = 1, strongly agree = 5*	2.56	☆☆☆
Dealing with embedded elements (move, connect, duplicate) turned out to be easy for me. *strongly disagree = 1, strongly agree = 5*	4.33	☆☆☆☆☆
Solving the task using the tools provided by the implementation was ... *strongly disagree = 1, strongly agree = 5*	4.50	☆☆☆☆☆
How satisfied were you with the workflow and the final result of this task? *very unsatisfied = 1, very satisfied = 5*	4.67	☆☆☆☆☆

Table 6.4.: Rating of statements regarding the usability of the implementation for Task 2. (☆= very bad, ☆☆= mostly bad, ☆☆☆= neutral, ☆☆☆☆= good, ☆☆☆☆☆= very good)

Task 3: Variation of embedded model structures

In the third task, participants should try to modify the embedded question in different ways. This task was designed to encourage participants to further explore the embedded models by modifying both their structure and their properties. First, participants were asked to add question and answer text to two to three different questions of their choice. In addition, participants were asked to try to change the type of two to three functional blocks of their choice. Participants were then asked to modify one of the question rounds by adding function blocks for a single-choice and a multiple-choice learning question. Here, participants were explicitly asked to try to find an isolated view in which the question round could be modified. Not all participants used the tabs at the bottom of the canvas. 3 participants added the two additional learning questions to an embedded question round from the main tab with relative ease. One of the first participants stated that it would be a useful feature to be able to double-click on the parent container of an embedded question round to switch to a specific tab. This change was then quickly incorporated. Subsequent participants stumbled upon this feature rather accidentally, indicating that participants had an expectation of being able to open a question round in its own tab with a double-click.

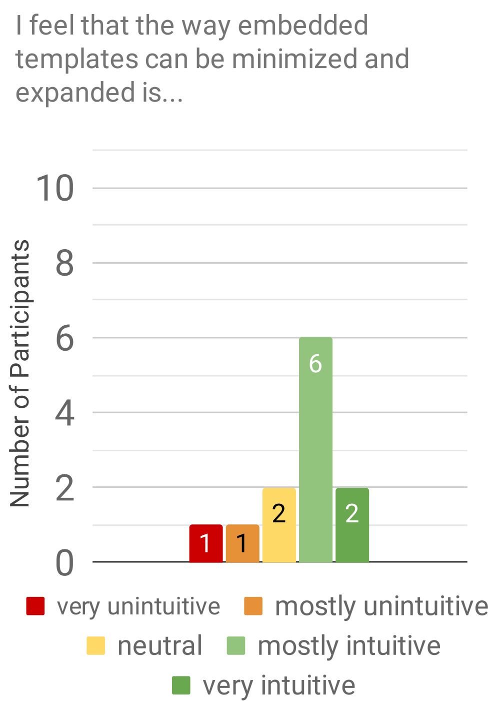


Figure 6.4.: Divided opinions regarding minimizing/expanding Parent Containers.

The last part of Task 3 revolved around the functionality that allows minimizing and expanding Parent Containers. As described in the previous chapter, the Context Menu was not implemented and its functionality was moved to the Properties Panel. Because of this change, it was anticipated that participants may find it difficult to access this functionality and may not like the placement of the button. This was reflected in the statements that participants made about this functionality (see Figure 6.4). Not all participants were happy with the way Parent Containers can be minimized and expanded. They instead expected to be able to do so by pressing a button on the top left side of the Parent Container. Furthermore, some participants noted that they expected the plus sign on a collapsed Parent Container to be clickable which unfortunately was not the case. However, one participant stated that *"[...] since the property window opened during selection, it was clear that the template could be expanded via it."* In summary, participants rated the implementation with regards to Task 3 mostly positively. User expectations were met on the majority of occasions expect for the interaction with Parent Containers. Most participants had no problem to modify the question rounds to their liking and to find and use tabs to edit embedded models in a separate view. Participant statements regarding Task 3 are summarized in Table 6.5 below.

Task 3: Variation of embedded model structures		
Question	**Score**	**Rating**
The tools provided for modifying embedded templates meet my expectations. *strongly disagree = 1, strongly agree = 5*	4.83	☆☆☆☆☆
The design and visualization of the tabs for embedded templates meet the expectations I had after the initial evaluation. *strongly disagree = 1, strongly agree = 5*	4.58	☆☆☆☆☆
I had no problems switching to an isolated model view for any of the question rounds. *very unintuitive = 1, very intuitive = 5*	4.58	☆☆☆☆☆
I find the way embedded templates can be minimized and expanded to be... *very unintuitive = 1, very intuitive = 5*	3.58	☆☆☆☆
Solving the task using the tools provided by the implementation was ... *strongly disagree = 1, strongly agree = 5*	4.50	☆☆☆☆☆
How satisfied were you with the workflow and the final result of this task? *very unsatisfied = 1, very satisfied = 5*	4.58	☆☆☆☆☆

Table 6.5.: Rating of statements regarding the usability of the implementation. (☆= very bad, ☆☆= mostly bad, ☆☆☆= neutral, ☆☆☆☆= good, ☆☆☆☆☆= very good)

Task 4: Finalization of a template and Template Sharing

The last task consisted of a combination of different activities. In the first part of Task 4, participants were asked to complete the modeling process by integrating the three rounds of questions into the rest of the model. Participants became familiar with the integration function and learned more about its purpose. Some participants expressed that they missed the planned Context Menu and its functionality. Nonetheless, all participants found the *Integrate Scenario* button in the properties bar and the general consensus was that the button was easy to find, contrary to the feelings about the button for expanding and retracting the Parent Container. In the second part, participants were asked to export their model as a template to their file system. Again, all participants were successful in this task and found the functionality in the *Export/Save* dialog box fairly quickly. 10 out of 12 participants indicated that they liked the steps required to export the template to their file system and that the export functionality met the expectations they had after the initial evaluation. Participants were then asked to import the previously downloaded model back into the canvas. All participants found the button in the *Import/Load* dialog with relative ease. However, there was some confusion as to the effect of pressing either the *Add to Model* button or the *Replace Model* button. Some participants pressed the *Add to Model* button only because of its blue color. When asked why they chose the button, participants noted that the *Replace Model* button looked to them like it was disabled and therefore not clickable. Hence, changing the color of the *Replace Model* button might be a consideration to avoid further confusion. Other than that, participants had no problem with both the export to and import from their local file system. Participants also rated the preview visualization of a model to be uploaded very positively. In the final task of the evaluation participants were asked to delete both the template and the category that they created earlier. Deleting templates was easy to the majority of participants, as most participants had already seen the *Delete Button* attached to the Template Preview Cards while browsing their templates. However, deleting categories was not as straightforward to the majority of participants, because it was difficult to locate the *Delete Button*. The category selector does allow for the deletion of categories. However it was not clear before the evaluation whether providing access to the delete functionality from within the category selector would contribute to an intuitive design. To answer this question, participants were asked whether they liked the way categories can be deleted. Participant feedback was very controversial as several different opinions about the category selector (see Figure 6.5) were stated. One participant noted that they would have expected some mechanism or dialog outside of both the export and import dialog that is responsible for category management. Another participant stated that they would instead have expected the category selector to appear in the *Import/Load dialog* as well. The participant stated that "*[...] then I do not have to remember exactly where the category selector is*". In the end it can be said that user expectations regarding the integration and the export and import of models were met. Participants rated the implementation positively regarding their expectations after the first evaluation. Some controversy was caused by

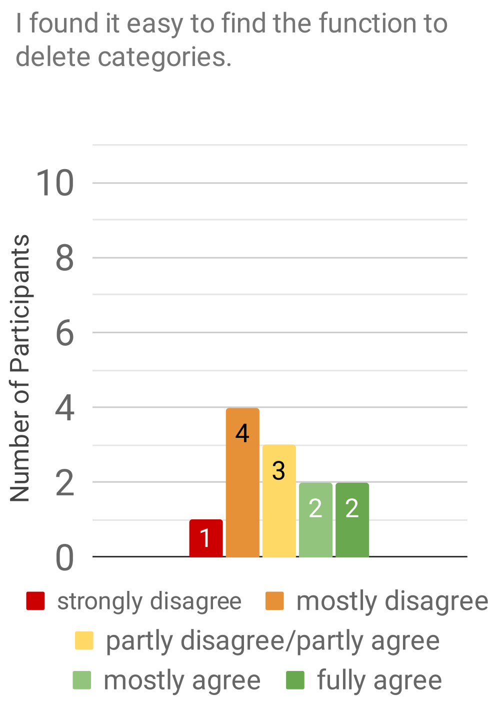


Figure 6.5.: Divided opinions regarding the deletion of categories.

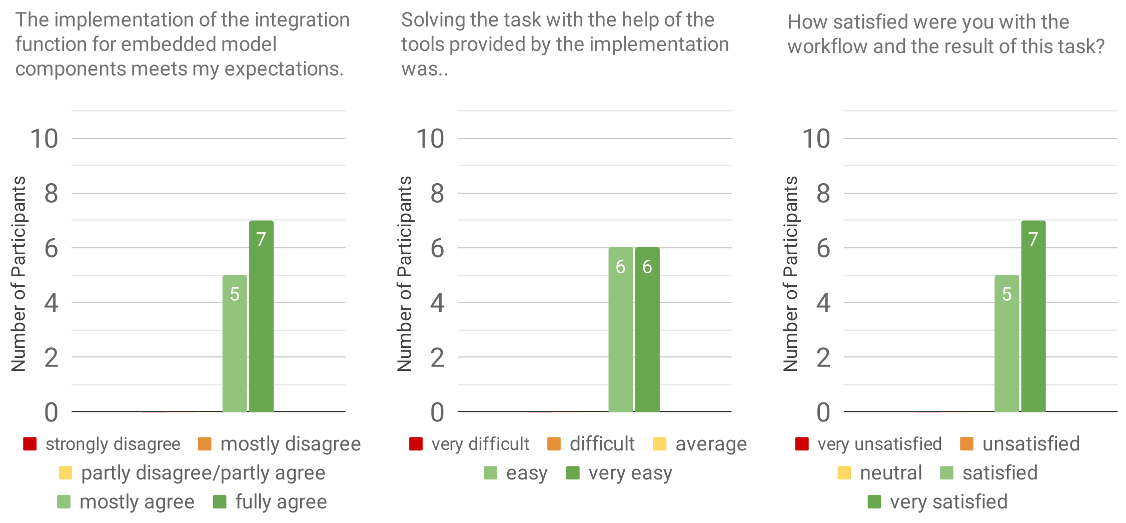


Figure 6.6.: Summary of the evaluation results of Task 4.

the category selector and the way that categories can be deleted. The majority of participants did visibly struggle to find the category selector and the *Delete Buttons* attached to each category. However, as the function to delete categories was added as one of the last features without any user feedback beforehand, it was anticipated that the way this feature was implemented will cause problems. The statements of participants regarding Task 4 are summarized in Table 6.6 below.

Task 4: Finalization of a template and Template Sharing		
Question	**Score**	**Rating**
The implementation of the integration function for embedded model components meets my expectations. *strongly disagree = 1, strongly agree = 5*	4.58	☆☆☆☆☆
I was able to easily find the button to integrate embedded model components. *strongly disagree = 1, strongly agree = 5*	4.67	☆☆☆☆☆
The "Integrate Scenario" function behaves as I would expect. *strongly disagree = 1, strongly agree = 5*	4.58	☆☆☆☆☆
I find the workflow for exporting a model as a template to the local file system to be... *very unintuitive = 1, very intuitive = 5*	4.83	☆☆☆☆☆
The implementation of the export function meets the expectations I had after the initial evaluation.*strongly disagree = 1, strongly agree = 5*	4.83	☆☆☆☆☆
I found it easy to find the function to delete templates. *strongly disagree = 1, strongly agree = 5*	4.58	☆☆☆☆☆
I found it easy to find the function to delete categories. *strongly disagree = 1, strongly agree = 5*	3.00	☆☆☆
I found it difficult to find the function for importing a template from the local file system. *strongly disagree = 1, strongly agree = 5*	1.17	☆☆☆☆☆
The preview for a selected template to be imported from the file system meets my expectations. *strongly disagree = 1, strongly agree = 5*	4.75	☆☆☆☆☆
The implementation of the import-from-file-system function meets the expectations I had after the initial evaluation. *strongly disagree = 1, strongly agree = 5*	4.67	☆☆☆☆☆
Solving the task using the tools provided by the implementation was ... *very difficult = 1, very easy = 5*	4.50	☆☆☆☆☆
How satisfied were you with the workflow and the final result of this task? *very unsatisfied = 1, very satisfied = 5*	4.58	☆☆☆☆☆

Table 6.6.: Rating of statements regarding the usability of the implementation. (☆= very bad, ☆☆= mostly bad, ☆☆☆= neutral, ☆☆☆☆= good, ☆☆☆☆☆= very good)

6.3. Open Feedback and Improvement Suggestions

During the evaluation, participants provided a lot of open feedback and improvement suggestions either verbally or via the questionnaire. This section will showcase the most important suggestions that were made by participants.

Sorting Templates inside Import/Load dialog

One participant noted that given enough time, a user could amass many templates during the usage of *stARS*. Every time a new template is created, the user has to scroll to the end of the list of private templates to select it which would take longer the more private templates exist. Therefore, a way for the user to sort templates has to be provided. The suggestion of the participant led to changing the implementation so that templates are sorted by their creation date, with the most recently created template appearing first in the list. However, other metrics for sorting such as *most used*, *least used* or *last modified* could be interesting to explore.

Interaction with Elements in the Canvas

Some participants criticized the way of interacting with model elements located inside the canvas. In particular, the majority of participants criticized the interaction with embedded model elements. The Context Menu was missed by one participant stating that the "*Expand/Hide button could be better implemented overall via extra context menu accessible via right click or via extra visual element*". Especially the plus button that is visible if a Parent Container is collapsed caused many participants to try and click it, expecting their embedded question rounds to expand. Therefore, future efforts should concentrate on implementing the Context Menu as it was originally proposed and evaluating its usefulness.

Furthermore, the selection of embedded question rounds was mentioned by several participants as it was difficult to select the Parent Container. After several participants commented on this, these difficulties were resolved by troubleshooting between the evaluation. However, during the discussion of these difficulties, one participant suggested an additional change in the interaction with embedded model elements. The participant thought it would be useful to assign different meanings to a double-click on an Parent Container depending on whether the edge of the container or somewhere else within the box was clicked. Double-clicking on the border should either expand or hide the Parent Container, while double-clicking anywhere within the Parent Container should bring up the embedded model's tab. A different participant had a similar idea to this, stating that a double click should expand a subprocess and a single click on the expanded subprocess should open the respective tab for the embedded model. This approach would avoid possible ambiguity as to where the border of the Parent Container starts and ends and where exactly a click has to be placed to achieve the desired effect. Future work could implement and compare both approaches in terms of intuitiveness. Another idea from different participant was to allow duplicating elements with a single click. When duplicating one of the question rounds in Task 2, the participant used the main menu and the "Copy" button. In doing so, the participant expected that pressing this button would immediately create a copy of the selected element and automatically place it in the model canvas. The participant reasoned that he did not know that he had to press the key combination (Ctrl+V) to paste a copied model into the canvas. This idea is worthwhile to be investigated in future efforts.

Category Management

As results of the evaluation showed, many participants criticized the way categories can be deleted. For most participants, it was not intuitive that deleting categories is only possible within the *Export/Save* dialog. One participant noted that a central location for managing categories needs to be provided elsewhere in the user interface, while another suggested adding the category selector and its functions to the *Import/Load* dialog. Another participant had the idea of asking users if a category should be deleted once the last template that belonged to that category was deleted. This could help users manage their categories more efficiently and could avoid "dead" categories. Furthermore, instead of only allowing the deletion of categories, editing (renaming) categories should also be possible.

Main Menu Icons

The majority of participants was confused by the icons that are used in the Main Menu for the import and export functions. One participant stated that "*Import/Export/Save/Load buttons I had to find by trial and error. Based on the icons, it was not immediately clear to me that these are the right buttons*". In addition, several participants stated that the icons of the buttons for resetting and centering the canvas were not intuitive enough in each case. Future work could therefore focus on either identifying and using existing icons that are more suitable for the use case in question, or designing icons specifically adapted for this purpose.

Tabs

Most participants praised the tabs that can be used to view and edit embedded model structures. One participant however noted, that besides labeling the tabs according to the embedded model's name, it could be of use to use different colors for each tab. The Parent Container and its respective tab would share the same background color, further strengthening the idea that both belong together. However, as this would represent a more visible change to the user interface, this idea should be further explored in another user study along with some of the other suggestions.

6.4. Summary

In this chapter, the extension of the *stARS* web editor that was developed during this master thesis was evaluated. For this purpose, a user study was conducted, which served on the one hand to test the usability and intuitiveness of the newly added functions and to identify problems that still need to be solved before the extension can become part of the system and the functions can be made available to other users. The user study showed that most of the features were very positively received by the participants and that the expectations for the mock-ups from the initial evaluation were largely met. Some testers rated aspects of the implementation negatively (e.g., the lack of a Context Menu or several aspects regarding category management), but the overall rating given by these testers was still positive. The implemented extension of the *stARS* web editor thus fulfills the requirement for intuitiveness (or comprehensibility and usability), which was formulated at the beginning of this master thesis. Due to the overall positive feedback, the final version of the implemented extension for the *stARS* web editor was merged into the development branch of the *stARS* frontend project and already received several smaller improvements in the meantime.

Furthermore, the user study has shown that there still remain some problems at the moment that need to be solved in the future. Here, the main focus lies on the further refinement of the proposed user interface components, the interaction with embedded model structures as well as the development of appropriate introductions and assistance. Overall, however, some of these problems are due to the immense implementation effort and time pressure within this master thesis. These issues can be fully addressed with more time and by further investigation of the main library bpmn-js regarding the realization of additional suitable extensions to the existing implementation. To achieve these goals, extensive investigations will have to be carried out in the future.

7. Conclusion

Chapter 7: Conclusion

The previous chapter described how the extension of the *stARS* web editor was successfully evaluated. This evaluation formed the conclusion of the development phase of this thesis. The final chapter briefly summarizes the research and results of this thesis before going on to discuss topics that could not be explored in depth within the scope of the master's thesis, but are nevertheless important for the continuation of the project.

7.1. Summary and Results

stARS is a novel approach to an ARS that focuses on giving lecturers more flexibility in the way typical ARS features are used during lectures. This is made possible by the *stARS* web editor, an application that allows the creation of workflows that map real-world learning scenarios and didactic strategies. The task of this thesis was to design a suitable extension for the *stARS* web editor that facilitates the modeling process of more complex learning scenarios. For this purpose, the current state of the art in the context of ARS with modeling capabilities and GMTs was analyzed to gain knowledge about suitable and relevant features that are missing in the *stARS* web editor. Based on the identified features, a requirements list for a solution to the shortcomings of the editor was first developed. Then, using a UCD-based process that incorporated feedback from real end users, a concept was designed and iteratively refined.

The concept included suggestions for new features that would enable easier reuse of components, including the introduction of templates, a template repository, enhanced save and export functionality, enhanced import and load functionality, and template sharing based on file exchange. In addition, several proposals were made to enable and facilitate the composition of complex models from multiple existing models or templates, including a feature that enables model composition in the first place, concepts for visualizing embedded model structures, concepts for interacting with containers of embedded structures, an integration feature that flattens nested models, and a design for dedicated tabs that enable the modification of embedded structures in an isolated manner. The majority of these features were then implemented by appropriately extending the existing codebase of the *stARS* web editor. During the implementation phase, a lot of knowledge about BPMN, the data format in which models are managed, was gathered and documented. These insights were appropriately exploited in the implementation of more complex model data manipulation routines.

After most of the proposals were successfully implemented, a final evaluation was conducted to determine if user expectations were met. Participants in the evaluation were given a carefully designed modeling task that prioritized the newly introduced features. The results of the evaluation indicated that the implementation can be considered a success for the most part. Participants were predominantly positive in their assessment of the newly added features, while also providing additional feedback suitable for further improving the current state of the *stARS* web editor. Table 7.1 illustrates the extent to which non-functional and functional requirements were met by the implementation in light of feedback provided by evaluation participants. In summary, the vast majority of features proposed by the concept were successfully implemented in terms of user expectation and intuitiveness. Many important features have been added to the *stARS* web editor to facilitate the creation of complex learning scenarios.

Non-Functional Requirement	**Notes & Open Issues**	**Status**
NFR1 Embeddable Solution		**fulfilled**
The implementation is embeddable in the sense of requirement **NFR1**, because it represents an extension of the existing code base of *stARS*.	–	
NFR2 Increase Usability of Existing Features		**fulfilled**
A user study confirmed, that the usability of existing features was increased when it comes to modeling complex learning scenarios that contain the same component multiple times.	–	
NFR3 Intuitiveness of New Features		**fulfilled**
The intuitiveness of all newly added features was confirmed by a user study.	–	

Functional Requirement	**Notes & Open Issues**	**Status**
FR1 Save & Export		**fulfilled**
The implementation supports saving any given model as a template. For this, a fully functional dialog with several abstraction options was realized. Concrete function blocks can be converted into their abstract counterparts and function block attributes can be reset. To manage and sort templates, template categories were introduced. Templates can be re-opened, edited and saved again as often as needed. Furthermore, any given template can be exported as a .bpmn file and downloaded to the local file system.	• The possibilities to edit and delete categories have to be enhanced. • The preview of the model that should be saved as a template does not provide any information regarding the validity of the model. • Currently, no categories for public templates exist. The creation of public categories by an administrator is not supported yet.	
FR2 Load & Import		**fulfilled**
The implementation provides a way to load a previously created model into the canvas for further refinement. It is possible to load and edit already created instances or templates. Furthermore, importing models in form of .bpmn files is possible. A preview for the model to be imported provides users with more information about a given .bpmn file.	• Adding a model to the canvas currently resets the undo/redo history. The same is true when the canvas content is replaced by a new model.	

Functional Requirement	Notes & Open Issues	Status
FR3 Templates & Template Repository		**fulfilled**
The implementation provides a way to create templates from either newly created models or already existing instances. Furthermore, a *Template Repository* is provided that allows to browse private and public templates. Available templates are visualized in an appropriate manner and a preview window allows to inspect a template in full size before import. A sidebar was implemented that allows for coarse grain template filtering on the basis selecting (potentially multiple) categories. A Search Bar was implemented to further facilitate the quick location of desired templates.	• Other ways to sort templates are missing. • Rendering performance can suffer on low end devices if many templates are shown at the same time.	
FR4 Scenario Sharing		**fulfilled**
The implementation provides a way to share templates via file exchange. Both an export and import feature were successfully implemented, supporting nested model structures.	• A cloud-based approach of scenario/template/model via *stARS* as a platform would be a reasonable extension.	
FR5 Model Composition & Embeddable Models		**fulfilled**
The implementation provides a way to compose bigger models by reusing smaller models. The Add To Model function allows to combine preexisting templates into more complex structures. Embedded model structures are visualized inside a container that can be moved around, connected, expanded, retracted, duplicated and deleted like any other model element.	• The current implementation breaks the undo / redo history, as import operations can not be undone.	
FR6 Adequate Visualization of Embedded Model Structures		**fulfilled**
By leveraging the BPMN subprocess, embedded model structures are visualized adequately. The implementation realized the hybrid visualization concept consisting of an in-place visualization of embedded structures paired with tabs that offer an isolated view on a given embedded structure.	• The current implementation breaks the undo / redo history, as modifications that were made inside a tab of an embedded model can not be undone.	

Functional Requirement	Notes & Open Issues	Status
FR7 On-demand Abstraction of Embedded Models Structures		**partially fulfilled**
Embedded models can be expanded and retracted on demand. The implementation leveraged the pre-existing mechanisms provided by the library bpmn-js to visualize parent containers in both retracted and expanded state.	• Expanding and retracting embedded models is only possible via the Properties Panel. The Context Menu that was intended for this purpose has yet to be implemented. • A button attached to Parent Containers of embedded structures that allows to minimize expanded containers still has to be implemented. • The plus sign shown on a collapsed Parent Container does not allow to expand the container as it was originally envisioned. • Transitions and surrounding nearby modeling elements are not properly aligned / displaced if an embedded structure is expanded.	
FR8 Modifiable Embedded Models Structures		**fulfilled**
Support for embedded model structures was implemented in such a way that modification of embedded structures is straightforward. Furthermore, tabs were implemented that allow to edit embedded structures in an isolated view. A function that allows to integrate embedded model structures into the rest of the model was realized.	• The integration function sometimes fails to align transitions to neighbouring elements properly. This is especially the case when models are created that do not follow the left-to-right metaphor[1].	
FR9 Exchangeable & Removable Embedded Models		**partially fulfilled**
The implementation provides a basic mechanism that allows to exchange embedded models. Embedded models can be selected and deleted like any other model element. A removed embedded element can subsequently be replaced by importing a different template.	• Due to time constraints, the proposed swap option that should have been accessible via the Context Menu was not implemented.	

Table 7.1.: Status of Non-Functional and Functional Requirement fulfillment: The implementation fulfilled the majority of requirements (shown in green). The requirements that were only partially met are highlighted in yellow.

[1]This means the progressive placement of model elements from left to right in the canvas.

7.2. Future Work

During the processing of this master thesis, some aspects and topics were identified that proved to be important to consider, but at the same time did not directly fit into the context of the task, which is why a more detailed consideration of these topics was refrained from. In the following, these topics, which may be relevant for future work on *stARS*, are briefly presented.

Back-end Compatibility

In this work, the concept of embedded model structures was introduced by using the BPMN subprocess, a model element that is exactly suited for this use case. However, the introduction of embedded model structures resulted in the loss of compatibility with the *stARS* back-end execution engine. The *stARS* workflow execution engine is not able to handle models that contain subprocess elements. To restore compatibility with the existing back-end infrastructure, an algorithm was developed to convert any given model containing subprocesses into an equivalent flat model. While the algorithm works perfectly fine, it is more of a workaround than a necessary feature in the front-end codebase. Therefore, future work should focus on making the back-end of *stARS* compatible with sub-processes (e.g. by introducing the subprocess to the meta-model). This would significantly reduce the complexity of parts of the codebase responsible for the web editor.

Front-end Unit testing

By implementing various data manipulation routines that enable model composition, a lot of complex code was introduced into an already extensive codebase. The new functionality has only been tested to a limited extent and the current implementation must be considered more of a prototype application. Future development of *stARS* and its front-end will lead to even more and more functionality being added. To make the application robust and end-user ready and to ensure software quality standards, the introduction of unit tests for the front-end module is inevitable. Automated frontend testing, unit testing and regression testing will provide future developers with the necessary tools to assess the overall quality of the software. The front-end module of *stARS* already includes some unit tests, but no conclusive statements about test coverage and code quality can be made at this time. Future work should therefore focus on implementing more unit and regression testing for the front-end module as the complexity of the codebase increases and several complicated data manipulation routines are executed on the client side that need to be tested against possible edge cases. Implementing a continuous integration environment such as **Jenkins** [2] or **TravisCI** [3] is highly recommended.

[2] `https://jenkins.io`, last successful access: 2021-02-14
[3] `https://travis-ci.org/`, last successful access: 2021-02-16

Performance Optimization

With the extension of the *stARS* web editor, many places were introduced where previews for models are displayed as **scalable vector graphics** (SVG). Some examples of this are the *Save/Export* dialog where a preview of a template is displayed, or the preview displayed when importing a model from the local file system. The most notable example, however, is the Template Repository, where the user is shown a preview of each private template. The preview images are a convenient way to visualize a model, allowing the user to select the desired template from many. However, rendering the models is a time-consuming process that is run on the client, possibly hundreds of times, depending on the set of templates and how the user interacts with the web editor. On low-end hardware or laptops, significant speed penalty and reduced responsiveness is the result. The effect can also be observed in the web editor's dashboard, where the user is shown a preview for each model instance. In extreme cases, it can take up to 30 seconds before any meaningful interaction with the system is possible. Figure 7.1 shows the time it takes to render 50 model previews in the *stARS* dashboard. Most of the time is spent evaluating JavaScript on the client side, which is responsible for creating the model preview SVGs. Future work should therefore focus on improving the performance and responsiveness of the *stARS* web editor to ensure the best possible user experience. To this end, two suggestions are made below.

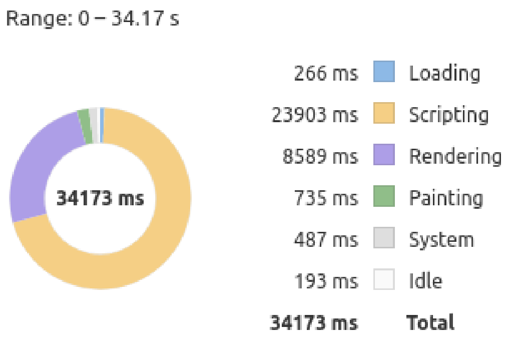


Figure 7.1.: Time required to render 50 model previews in the dashboard of *stARS*.

Thumbnail image files To address the delay caused by the extensive rendering of model previews, one solution would be to use thumbnail image files for model previews. As long as a full size model preview is not requested by the user, a jpeg or png file could be used instead. However, the backend server would then need to persist and manage these preview images. Also, additional bandwidth would be required to transmit the preview images to the clients.

Server-side rendering A more comprehensive approach that could eliminate most causes of delay is server-side rendering. Currently, the *stARS* frontend is configured as a SPA, which means that client-side JavaScript is responsible for rendering the application in the browser. One approach that would avoid slow rendering on the client would be to move the rendering to the server side. Fortunately, nuxt-js, part of the frontend module tech stack, allows switching to SSR in a very simple way. This would offload the client-side hardware, assuming that a powerful web server handles the rendering instead. Several libraries such as **jsdom**[4] or **svgdom**[5] allow for the rendering of svg on headless systems (e.g. a node-js

[4]jsdom documentation: `https://github.com/jsdom/jsdom`, last successful access: 2021-02-14
[5]svgdom documentation: `https://www.npmjs.com/package/svgdom`, last successful access: 2021-02-16

backend server). A guide on implementing server-side rendering of SVG can be found in [66]. Furthermore, several guides exist that explore the combination of SPA and SSR rendering techniques, as illustrated in [67]. Future work could apply these techniques to the codebase of *stARS* and investigate, whether a notable performance gain can be achieved with SSR.

Transition Management and Model Displacement

A known problem of the *stARS* web editor is the way transitions are placed by the editor. Sometimes, transitions have too many way points and too many corners. Other times, transition placement simply ignores neighboring elements, resulting in transitions that pass through function blocks. While this thesis was written, a student effort by Niclas Zellerhoff addressed these issues and developed an algorithm that optimizes transition layouts. This algorithm works most of the time preferably and produces desirable results. However, it was not tested with the new collapsed or expanded subprocesses. Future work should therefore continue the efforts of Niclas Zellerhoff and investigate on how transitions could be placed in a more optimal way, especially around subprocesses (i.e. Parent Containers). Furthermore, an unanswered question is whether surrounding transitions and model elements should be displaced whenever a subprocess is expanded, and whether surrounding elements should be put back to their original position once the subprocess is collapsed. This problem is especially noticeable when a collapsed subprocess is moved around and then expanded, which results in "jagged" and very indirect transitions.

In addition to that, the integrate function for embedded structures causes similar problems that were observed during the evaluation. Some participants did not create their models with the left-to-right metaphor in mind. Instead, they placed model elements from top to bottom. Upon integration of the question rounds, the resulting transitions did not look very pleasing. Transitions had sharp turns and went straight through other modeling elements. Most of the participants then manually corrected the layout of the transitions to the best of their abilities. The unpleasant layout is caused by the rather simple code that calculates the new transitions during the integration of an embedded model, which only incorporates the first and last way point of the original transition. Future work should therefore concentrate on developing a dedicated algorithm that calculates a pleasing and good looking layout for the resulting transitions automatically.

Cloud-based Template Sharing

This work introduced templates and a basic way of sharing templates via file exchange. A template can be exported and downloaded as a file, which can be sent to others via e-mail and re-imported via the *Import/Load* dialog. However, this way of sharing templates is cumbersome. A more complete solution would allow users to provide their templates to other users directly via the Template Repository. Future work could concentrate on mechanisms that allow to publicize templates in a way that makes them re-usable to other users. Users of *stARS* would then be able to create additional re-usable content from which all users could benefit. A realistic extension of this feature could include recommendations for templates based on popularity or use case.

Bibliography

[1] *Computer im Hörsaal - Fluch oder Segen?* **[online]** `https://media1.faz.net/ppmedia/aktuell/wirtschaft/2840271525/1.4065051/format_top1_breit/elektronische-stoerenfriede-in.jpg`. **[accessed: 2020-09-06]**.

[2] *Number of smartphone users worldwide from 2016 to 2021*. **[online]** `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`. **[accessed: 2020-09-26]**.

[3] Ruth Wood and Shaista Shirazi. "**A systematic review of audience response systems for teaching and learning in higher education: The student experience**". In: *Computers & Education* 153 (2020). ISSN: 0360-1315. DOI: `https://doi.org/10.1016/j.compedu.2020.103896`. URL: `http://www.sciencedirect.com/science/article/pii/S0360131520300956`.

[4] Joshua Freeman and Alison Dobbie. "**Use of an audience response system to augment interactive learning**". In: vol. 37. 1. 2005, pp. 12–4.

[5] Jeffrey R Stowell and Jason M Nelson. "**Benefits of electronic audience response systems on student participation, learning, and emotion**". In: *Teaching of psychology* 34.4 (2007), pp. 253–258.

[6] Tommy Kubica et al. "**stARS: Proposing an Adaptable Collaborative Learning Environment to Support Communication in the Classroom.**" In: *CSEDU (2)*. 2020, pp. 390–397.

[7] Ilja Shmelkin. "**Entwicklung eines Metamodells zur Beschreibung kontext-sensitiver Audience Response Systeme**". Technische Universität Dresden, 2019.

[8] Lidia Roszko. "**Entwicklung eines graphischen Editors zur Erstellung von beliebigen Lernszenarien in Audience Response Systemen**". Technische Universität Dresden, 2019.

[9] Eric Mazur. ***Peer Instruction: A User's Manual***. Upper Saddle River, N.J. : Prentice Hall, 1997. ISBN: 0135654416 9780135654415.

[10] Aliya Nusrath et al. “**Jigsaw Classroom: Is it an Effective Method of Teaching and Learning? Student’s Opinions and Experience.**” In: vol. 13. 2. 2019, p. 3.

[11] Martin Ebner et al. “**Technologiegestützte Echtzeit-Interaktion in Massenvorlesungen im Hörsaal. Entwicklung und Erprobung eines digitalen Backchannels während der Vorlesung**”. In: *Lernräume gestalten - Bildungskontexte vielfältig denken* (2014), pp. 567–578.

[12] Kathy Kotiadis and Stewart Robinson. “**Conceptual modelling: Knowledge acquisition and model abstraction**”. In: *2008 Winter Simulation Conference*. IEEE. 2008, pp. 951–958.

[13] Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske. “**Process model abstraction: A slider approach**”. In: *2008 12th International IEEE Enterprise Distributed Object Computing Conference*. IEEE. 2008, pp. 326–331.

[14] Michael Redhead. “**Models in physics**”. In: *The British Journal for the Philosophy of Science* 31.2 (1980), pp. 145–163.

[15] Alexander Osterwalder, Yves Pigneur, and Christopher L Tucci. “**Clarifying business models: Origins, present, and future of the concept**”. In: *Communications of the association for Information Systems* 16.1 (2005), pp. 1–25.

[16] Susanne Strahringer. “**Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips.**” In: *CEUR Workshop Proceedings zur Modellierung ’98* January 1998 (1998), pp. 1–6.

[17] Tim Shaffer, Nathaniel Kremer-Herman, and Douglas Thain. “**Flexible partitioning of scientific workflows using the JX workflow language**”. In: *ACM International Conference Proceeding Series* (2019). DOI: `10.1145/3332186.3338100`.

[18] Böhm, Corrado and Jacopini, Giuseppe. “**Flow diagrams, turing machines and languages with only two formation rules**”. In: *Communications of the ACM* 9.5 (1966), pp. 366–371.

[19] ***BPMN 2.0 Specification***. **[online]**
`https://www.omg.org/spec/BPMN/2.0/`.
[accessed: 2020-09-02].

[20] Thomas Allweyer. ***BPMN 2.0: Introduction to the Standard for Business Process Modeling***. BoD–Books on Demand, 2016. ISBN: 9783837093315.

[21] ***BPMN Examples***. **[online]**
`https://camunda.com/de/bpmn/bpmn-examples/`.
[accessed: 2020-12-30].

[22] Ilja Shmelkin. ***Untersuchung der Adaptierbarkeit webbasierter Audience Response Systeme***. 2018.

[23] Thomas Köhler et al. “**Wissensgemeinschaften: Digitale Medien - Öffnung und Offenheit in Forschung und Lehre**”. In: *Wissensgemeinschaften 2011* (2011), pp. 178–187.

[24] Kathleen Marrs and Gregor Novak. "**Just-in-time teaching in biology: Creating an active learner classroom using the internet**". In: vol. 3. 1. Am Soc Cell Biol, 2004, pp. 49–61.

[25] ***The Think, Pair, Share Cooperative Learning Strategy***. **[online]**
`https://www.teachervision.com/group-work/think-pair-share-cooperative-learning-strategy`.
[accessed: 2020-08-30].

[26] Catherine H Crouch and Eric Mazur. "**Peer instruction: Ten years of experience and results**". In: *American journal of physics* 69.9 (2001), pp. 970–977.

[27] Catherine H. Crouch et al. "**Peer Instruction: Engaging Students One-on-One, All At Once**". In: *Research-Based Reform of University Physics* (2007), pp. 1–55. ISSN: 10476938. DOI: `10.1364/OPN.9.9.000037`.

[28] Elliot Aronson et al. ***The Jigsaw Classroom.*** Sage, 1978. ISBN: 978-0803909977.

[29] Chadia Abras, Diane Maloney-Krichmar, Jenny Preece, et al. "**User-centered Design**". In: *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications* 37.4 (2004), pp. 445–456.

[30] ***Usability & User Experience (UX) - A definition provided by usability.de***. **[online]**
`https://www.usability.de/en/usability-user-experience.html`.
[accessed: 2020-09-04].

[31] Jakob Nielsen. ***10 Usability Heuristics for User Interface Design***. **[online]**
`https://www.nngroup.com/articles/ten-usability-heuristics/`.
[accessed: 2020-09-05].

[32] ***ARSnova - An ARS developed by the Technische Hochschule Mittelhessen***. **[online]**
`https://arsnova.thm.de/`.
[accessed: 2021-01-11].

[33] ***PINGO - An ARS developed by the Universität Paderborn***. **[online]**
`https://pingo.coactum.de/`.
[accessed: 2021-01-11].

[34] ***Einsatz von Classroom-Response-Systemen und Peer Instruction in der Veranstaltung Grundlagen von Datenbanken***. **[online]**
`http://www.hochschullehre.org/wp-content/files/die_hochschullehre_Wolters_2018.pdf`.
[accessed: 2021-01-11].

[35] ***tweedback - A web-based live feedback system of the University of Rostock***. **[online]**
`https://tweedback.de`.
[accessed: 2021-01-11].

[36] ***FreeQuizDome - A free software for anonymous surveys and voting for large groups***. **[online]**
`https://tweedback.de`.
[accessed: 2021-01-11].

[37] *LucidChart - A web-based proprietary platform that allows users to collaborate on drawing, revising and sharing charts and diagrams.* **[online]**
`https://www.lucidchart.com/`.
[accessed: 2020-09-26].

[38] *Gliffy - Web-based diagram editor*. **[online]**
`https://gliffy.com`.
[accessed: 2020-09-26].

[39] *Microsoft Visio - A diagramming and vector graphics application that is part of the Microsoft Office family*. **[online]**
`https://www.microsoft.com/de-de/microsoft-365/visio/flowchart-software`.
[accessed: 2020-09-26].

[40] *bpmn.io*. **[online]**
`https://www.bpmn.io`.
[accessed: 2020-09-26].

[41] *Camunda - An open-source workflow and decision automation platform*. **[online]**
`https://camunda.com`.
[accessed: 2020-09-26].

[42] *Kissflow*. **[online]**
`https://kissflow.com`.
[accessed: 2020-09-26].

[43] *draw.io - A free online diagram software for making flowcharts, process diagrams, org charts, UML, ER and network diagrams*. **[online]**
`https://draw.io`.
[accessed: 2020-09-26].

[44] *Alfred App - A productivity application for macOS*. **[online]**
`https://www.alfredapp.com/workflows/`.
[accessed: 2020-09-26].

[45] *CPN Tools - A tool for editing, simulating, and analyzing high-level Petri nets*. **[online]**
`https://cpntools.org`.
[accessed: 2020-09-26].

[46] *Visual Paradigm - An all-in-one UML, SysML, BPMN Modeling Platform for Agile, EA TOGAF ADM Process Management.* **[online]**
`https://visual-paradigm.com`.
[accessed: 2020-09-26].

[47] *cardanit - A BPM editor with dynamic process and decision modeling*. **[online]**
`https://cardanit.com`.
[accessed: 2020-09-26].

[48] *Activiti Modeler - A lightweight, java-centric open-source BPMN engine*. [online]
https://www.activiti.org/.
[accessed: 2020-09-26].

[49] *creately - A visual collaboration tool with diagramming and design capabilities*. [online]
https://creately.com.
[accessed: 2020-09-26].

[50] *Flowable - An open-source workflow engine written in Java*. [online]
https://flowable.com.
[accessed: 2020-09-26].

[51] *HEFLO - A technology platform created for companies that want to control and scale their business processes*. [online]
https://www.heflo.com.
[accessed: 2020-09-26].

[52] *Bonita Studio - An open-source business process management and Low-code development platform*. [online]
https://www.bonitasoft.com/.
[accessed: 2020-09-26].

[53] *cacoo - An online diagram software to create flowcharts, wireframes, UML models and more*. [online]
https://cacoo.com.
[accessed: 2020-09-26].

[54] *smartdraw - A diagram tool used to make flowcharts, organization charts, mind maps, project charts, and other business visuals*. [online]
https://www.smartdraw.com.
[accessed: 2020-09-26].

[55] *mydraw - An advanced diagramming software for drawing flowcharts, org charts, mind maps, network diagrams and more*. [online]
https://mydraw.com.
[accessed: 2020-09-26].

[56] *Pingendo - A free app for Bootstrap prototyping*. [online]
https://pingendo.com.
[accessed: 2021-01-11].

[57] *What Is a Single Page Application?* [online]
https://www.excellentwebworld.com/what-is-a-single-page-application/.
[accessed: 2021-01-11].

[58] *vue.js - A progressive JavaScript Framework for building user interfaces.* [online]
https://vuejs.org/.
[accessed: 2020-12-30].

[59] *yarn - A package manager for the JavaScript programming language*. **[online]**
`https://yarnpkg.com/`.
[accessed: 2021-02-22].

[60] *nuxt.js - A framework for making web development simple and powerful.* **[online]**
`https://nuxtjs.org/`.
[accessed: 2020-12-30].

[61] *bpmnjs - BPMN 2.0 viewer and editor.* **[online]**
`https://bpmn.io/toolkit/bpmn-js/`.
[accessed: 2020-11-02].

[62] *node2xmljs - A simple XML to JavaScript object converter*. **[online]**
`https://www.npmjs.com/package/xml2js`.
[accessed: 2020-12-30].

[63] *Bootstrap - A responsive, mobile-first front-end framework*. **[online]**
`https://getbootstrap.com/`.
[accessed: 2020-12-30].

[64] *Font Awesome - The web's most popular icon set and toolkit*. **[online]**
`https://fontawesome.com/`.
[accessed: 2020-12-30].

[65] *patch-package - Lets app authors instantly make and keep fixes to npm dependencies.* **[online]**
`https://www.npmjs.com/package/patch-package`.
[accessed: 2021-01-17].

[66] *Love Generating SVG With JavaScript? Move It To The Server!* **[online]**
`https://www.smashingmagazine.com/2014/05/love-generating-svg-javascript-move-to-server/`.
[accessed: 2021-02-14].

[67] *SPAs, PWAs and SSR - Comparison and combination of rendering techniques for the web.* **[online]**
`https://simplabs.com/blog/2019/04/05/spas-pwas-and-ssr/`.
[accessed: 2021-02-15].

A. Pre-Evaluation Questionnaire

Umfrage für die Evaluation von stARS

Sehr geehrte Damen und Herren,
vielen Dank, dass Sie sich die Zeit nehmen, um an der Evaluation von stARS (scenario-tailored Audience Response System) teilzunehmen.

In den folgenden rund 60 Minuten sollen Sie den aktuellen Stand sowie Vorschläge zur Optimierung des stARS-Editors zur Modellierung von eigenen Lehrszenarien bewerten.

Wir möchten Sie darauf hinweisen, dass es bei der Evaluation keine richtigen oder falschen Antworten gibt und bitten Sie die einzelnen Fragen möglichst ehrlich zu beantworten. Sofern Unklarheiten bei der Evaluation auftreten, fragen Sie bitte jederzeit bei uns nach.

* Erforderlich

1) Fragen zur bisherigen Erfahrung

Bitte beantworten Sie die folgenden Fragen.

1. Wie lange betreiben Sie bereits Lehre bzw. haben Sie Lehre betrieben? *

Markieren Sie nur ein Oval.

- ◯ Bis zu 1 Jahr
- ◯ Zwischen 1 und 3 Jahre
- ◯ Zwischen 3 und 5 Jahre
- ◯ Zwischen 5 und 10 Jahre
- ◯ Über 10 Jahre
- ◯ Sonstiges: __________

2. Welche Arten von Lehrveranstaltungen haben Sie bereits gehalten? *

Wählen Sie alle zutreffenden Antworten aus.

- ☐ < 20 Teilnehmer (z.B. Übung, Seminar)
- ☐ 20 - 49 Teilnehmer (z.B. große Übung, kleine Vorlesung)
- ☐ 50 - 150 Teilnehmer (z.B. Vorlesung)
- ☐ > 150 Teilnehmer (z.B. große Vorlesung)

Sonstiges: ☐ __________

3. Halten Sie während der Corona-Krise Lehrveranstaltungen? Wenn ja, welche Strategie(n) benutzen Sie hierbei? *

4. Mit welchen technischen Werkzeugen unterstützen Sie allgemein Ihre Lehrveranstaltungen? *

Wählen Sie alle zutreffenden Antworten aus.

- ☐ Präsentations-Programm (z.B. PowerPoint oder Keynote)
- ☐ Handschriftliche Notizen (z.B. über Tablet)
- ☐ Werkzeuge zur Aktivierung von Studierenden (z.B. Audience Response Systeme, Backchannel Systeme)
- ☐ Vorlesungsvideos
- ☐ Aufgaben zur Vor- / Nachbereitung

Sonstiges: ☐

5. Sofern bereits benutzt: Wie oft haben Sie bereits Werkzeuge zur Aktivierung von Studierenden eingesetzt? Welche Werkzeuge haben Sie hierfür eingesetzt und welche Erfahrung haben Sie dabei gemacht? *

6. Haben Sie vor dieser Evaluation bereits schon einmal mit dem stARS-Editor gearbeitet oder das stARS-Metamodell evaluiert? *

Wählen Sie alle zutreffenden Antworten aus.

- ☐ Editor
- ☐ Metamodell
- ☐ Nein

2) Bewertung der Benutzerfreundlichkeit

Im diesem Teil der Evaluation sollen Sie den stARS-Editor benutzen und sowohl vorgegebene Szenarien als auch ein eigenes Szenario modellieren. Zur besseren Einschätzung der Benutzbarkeit bitten wir Sie Ihren Bildschirm mit uns zu teilen.

Bitte navigieren Sie zu nachfolgender URL und loggen Sie sich als neuer Lehrender ein. Öffnen Sie anschließend das Fenster zur Modellierung eines neuen Szenarios.
https://stars-project.com/

Aufgabe 1

In der ersten Aufgabe sollen Sie eine einfache Lehrveranstaltung mit drei Inhaltsblöcken modellieren, deren Präsentation durch zwei Fragerunden unterbrochen wird.

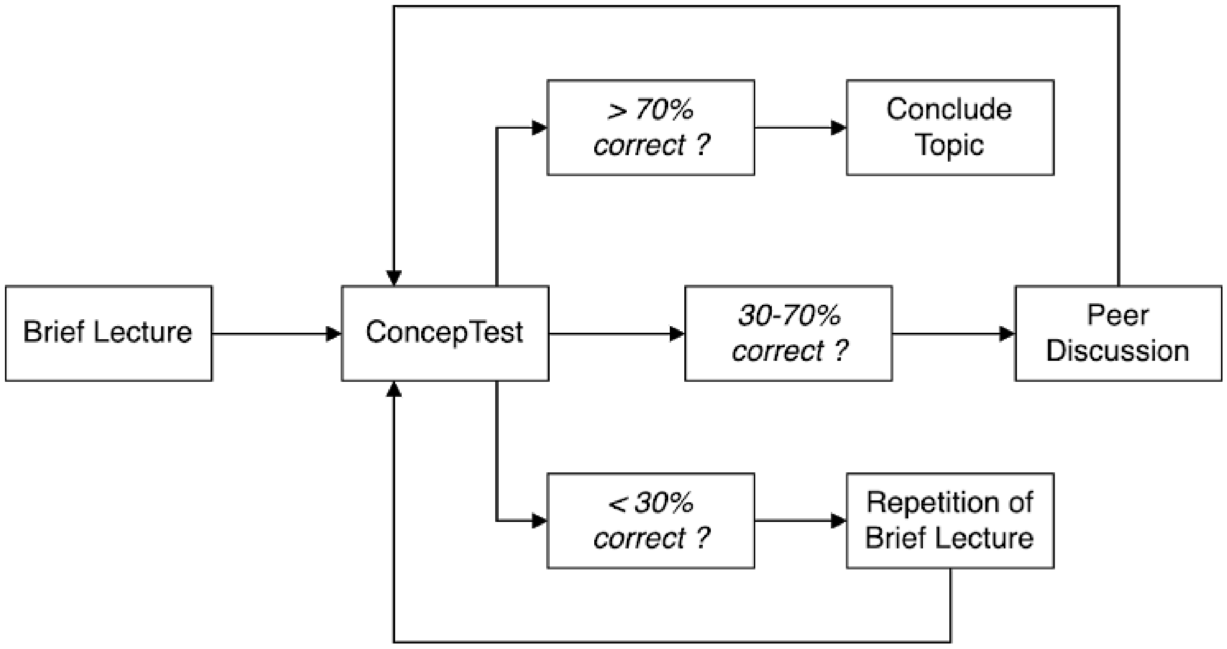
Überlegen Sie sich hierzu im ersten Schritt durch welche Funktionalität Ihre Inhaltsblöcke unterstützt werden sollen. Fügen Sie diesen Block ein. Anschließend erweitern Sie Ihr Szenario um eine Fragerunde mit zwei parallelen Lernfragen mit einem Sub-Typ Ihrer Wahl. Erweitern Sie das Szenario um einen zweiten Inhaltsblock, eine zweite Fragerunde mit zwei parallelen Umfragen beliebigen Sub-Typs sowie um einen abschließenden Inhaltsblock.

Aufgabe 2)

In der zweiten Aufgabe sollen Sie einen Workflow modellieren, um die Lehrmethode "Peer Instruction" (siehe unten) umzusetzen.

Versuchen Sie diese Lehrmethode mittels geeigneter Blöcke in stARS abzubilden. Der Versuchsleiter wird Sie hierbei durch das Szenario leiten und steht Ihnen bei Fragen jederzeit unterstützend zur Verfügung.

Beispielhafter Ablauf von Peer Instruction



Bewertung der Benutzerfreundlichkeit

Im letzten Abschnitt haben Sie den stARS-Editor kennengelernt. Nun bitten wir Sie diesen anhand einiger Fragen auf dessen Benutzerfreundlichkeit zu bewerten.

7. Bitte beantworten Sie folgende Fragen: *

Markieren Sie nur ein Oval pro Zeile.

	Stimme gar nicht zu	Stimme eher nicht zu	Weder noch	Stimme eher zu	Stimme voll zu
Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	⬭	⬭	⬭	⬭	⬭
Ich empfinde das System als unnötig komplex.	⬭	⬭	⬭	⬭	⬭
Ich empfinde das System als einfach zu nutzen.	⬭	⬭	⬭	⬭	⬭
Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	⬭	⬭	⬭	⬭	⬭
Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	⬭	⬭	⬭	⬭	⬭
Ich finde, dass es im System zu viele Inkonsistenzen gibt.	⬭	⬭	⬭	⬭	⬭
Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	⬭	⬭	⬭	⬭	⬭
Ich empfinde die Bedienung als sehr umständlich.	⬭	⬭	⬭	⬭	⬭
Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	⬭	⬭	⬭	⬭	⬭
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	⬭	⬭	⬭	⬭	⬭

8. Was hat Ihnen besonders gut gefallen?

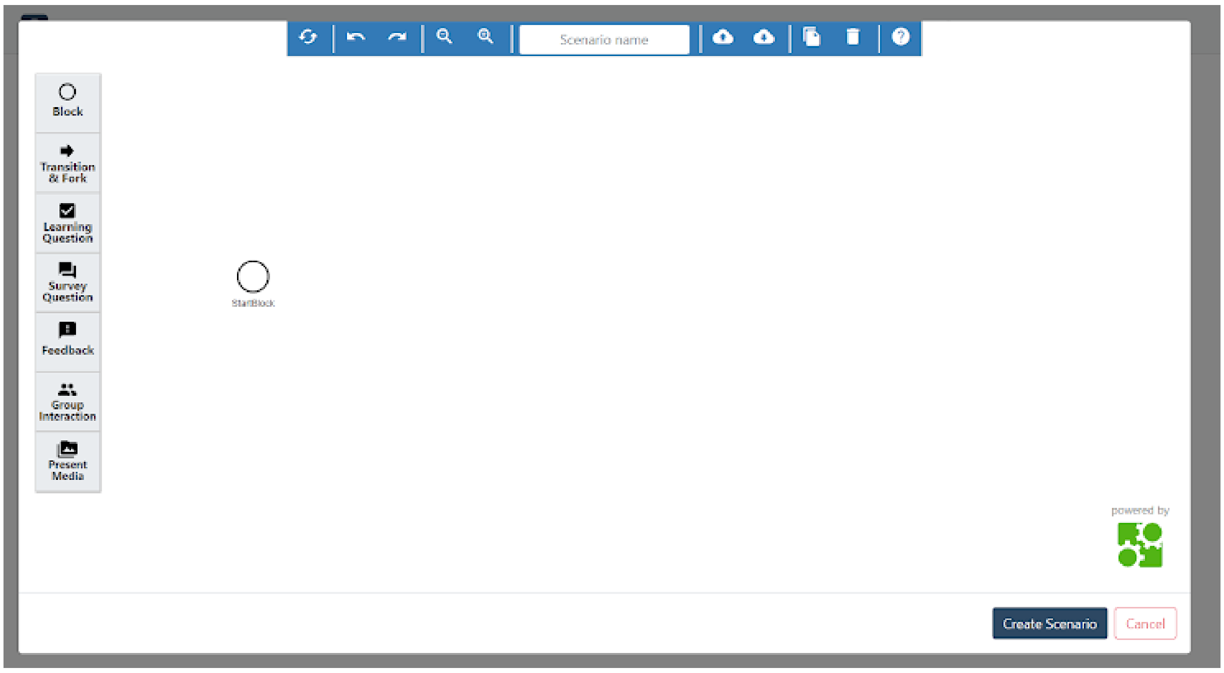
9. Was hat Ihnen nicht gefallen?

3) Bereitstellung einer initialen Hilfe

In diesem Teil der Umfrage präsentieren wir Ihnen Komponenten, die den Einstieg in den Editor erleichtern sollen.

Bitte beantworten Sie die jeweiligen Frage-Blöcke, sobald Sie auf den Präsentationsfolien darauf hingewiesen werden.

10. Welche Elemente oder Konzepte waren für Sie am Anfang unklar? *

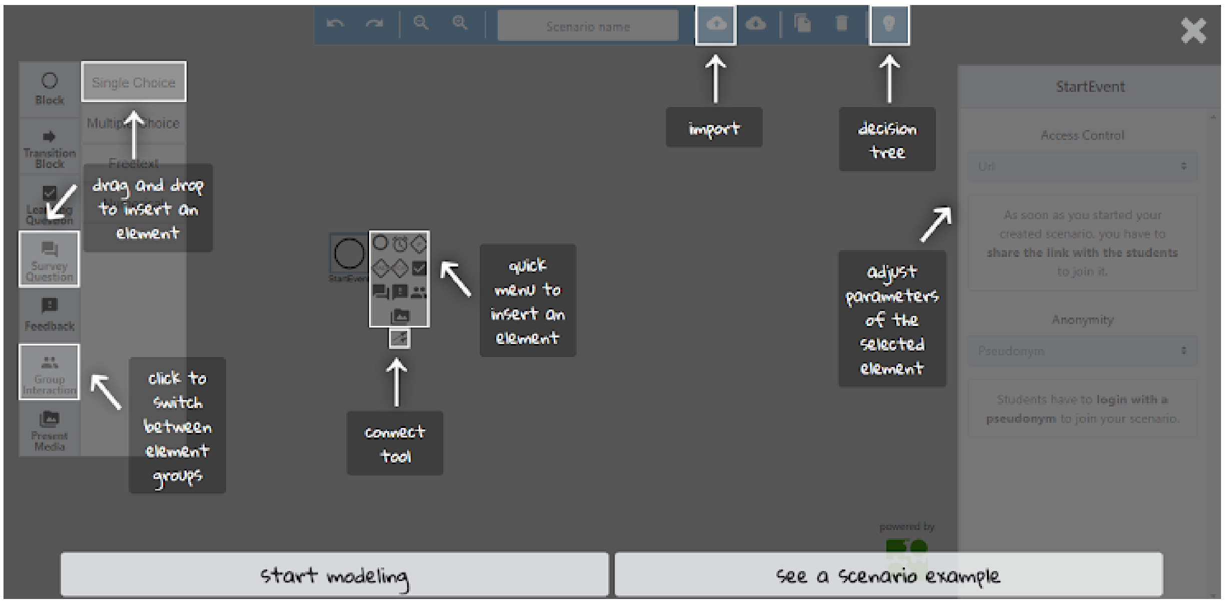


3.1) Initiale Hilfe: Overlay

Nach dem ersten Öffnen des Editors wird ein Overlay angezeigt, das helfen soll, die wichtigsten Elemente und Funktionen schnell zu erkennen.

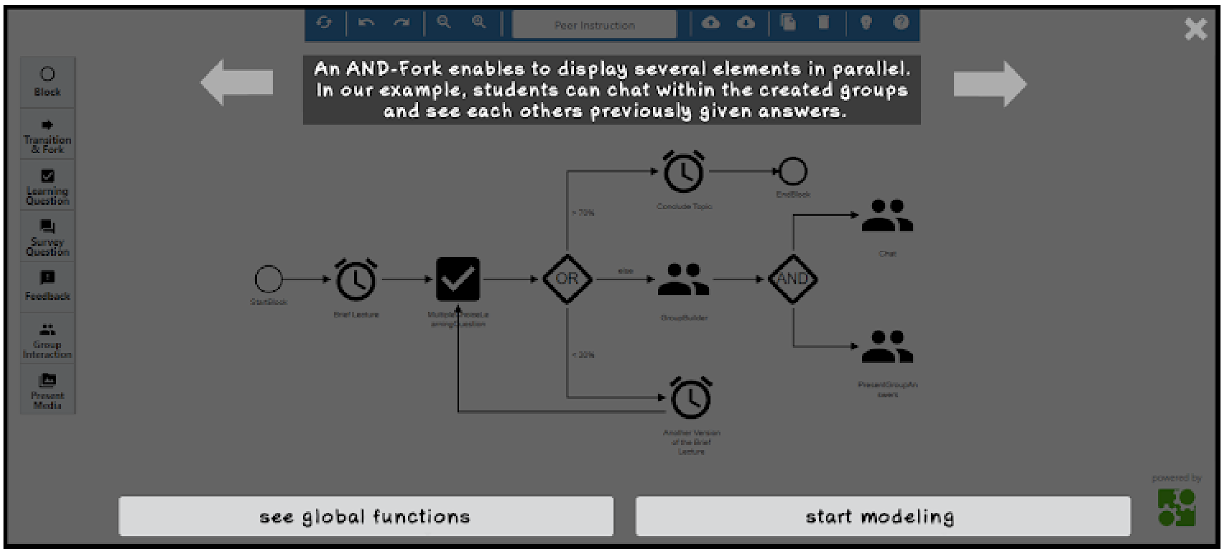
Die erste Evaluation hat ergeben, dass das Overlay sowohl die allgemeinen Elemente und Funktionen (Teil 1) als auch ein Beispielszenario (Teil 2) vorstellen sollte.

Overlay Teil 1


Scenario name
import
decision tree
StartEvent
Access Control
Url
As soon as you started your created scenario, you have to share the link with the students to join it.
Anonymity
Pseudonym
Students have to login with a pseudonym to join your scenario.
Block
Transition Block
Learning Question
Survey Question
Feedback
Group Interaction
Present Media
Single Choice
Multiple Choice
Freetext
drag and drop to insert an element
quick menu to insert an element
adjust parameters of the selected element
click to switch between element groups
connect tool
start modeling
see a scenario example

11. Würden Sie in der angezeigten Version vom Overlay (Teil 1) andere Elemente zum Beschreiben auswählen? *

Overlay Teil 2


An AND-Fork enables to display several elements in parallel. In our example, students can chat within the created groups and see each others previously given answers.

Overlay Teil 2 - Schritte

Schritt 1: “This example shows how one iteration of peer instruction (PI) is modeled.”

Schritt 2: “The first stage of PI is a brief lecture, in which a new topic is introduced. This is represented by a pause block that describes situation in which no system functionality is required.”

Schritt 3: “In the concepTest, students’ gained knowledge is checked. In our example, this is represented by a learning question of the subtype multiple choice.”

Schritt 4: “Next, the conditional execution of the lecture based on students’ previously given answers is modeled by an OR-Fork.”

Schritt 5: “If the minority of students answers correctly, another version of the brief lecture is held. The concepTest is afterwards repeated.”

Schritt 6: “If the majority of students answers correctly, the topic is concluded and the iteration of PI ends.”

Schritt 7: “If some students answered correctly while others did not, a peer discussion should be executed, in which students discuss their given answers. In order to form group of students, a group builder is needed.”

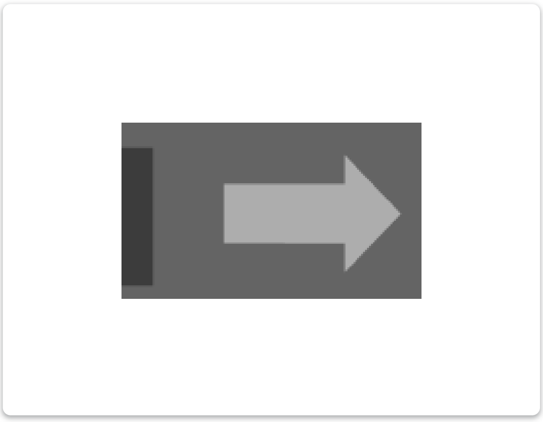
Schritt 8: “An AND-Fork enables to display several elements in parallel. In our example, students can chat within the created groups and see each others previously given answers.”

Schritt 9: “After completing the peer discussion, the concepTest is repeated.”

12. Würden Sie in der angezeigten Version vom Overlay (Teil 2) andere Elemente zum Beschreiben auswählen oder die Anzahl der Schritte verringern/erhöhen? *

13. Wie sollte man während der Präsentation des Beispielszenarios zum nächsten Schritt übergehen? (Mehrfachauswahl möglich) *

Wählen Sie alle zutreffenden Antworten aus.



☐ automatisch (Timeout)

☐ mittels des Pfeils auf der Oberfläche

Sonstiges: ☐ ____________

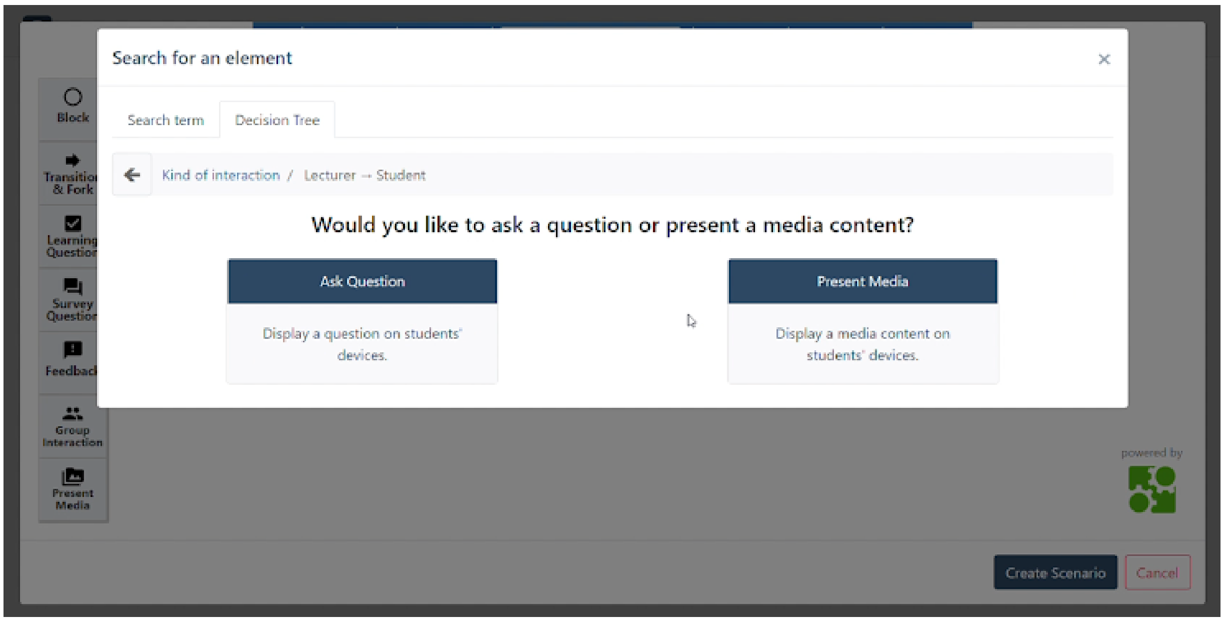
☐ mittels Pfeiltasten [→]

14. Ergänzendes Feedback zum Overlay

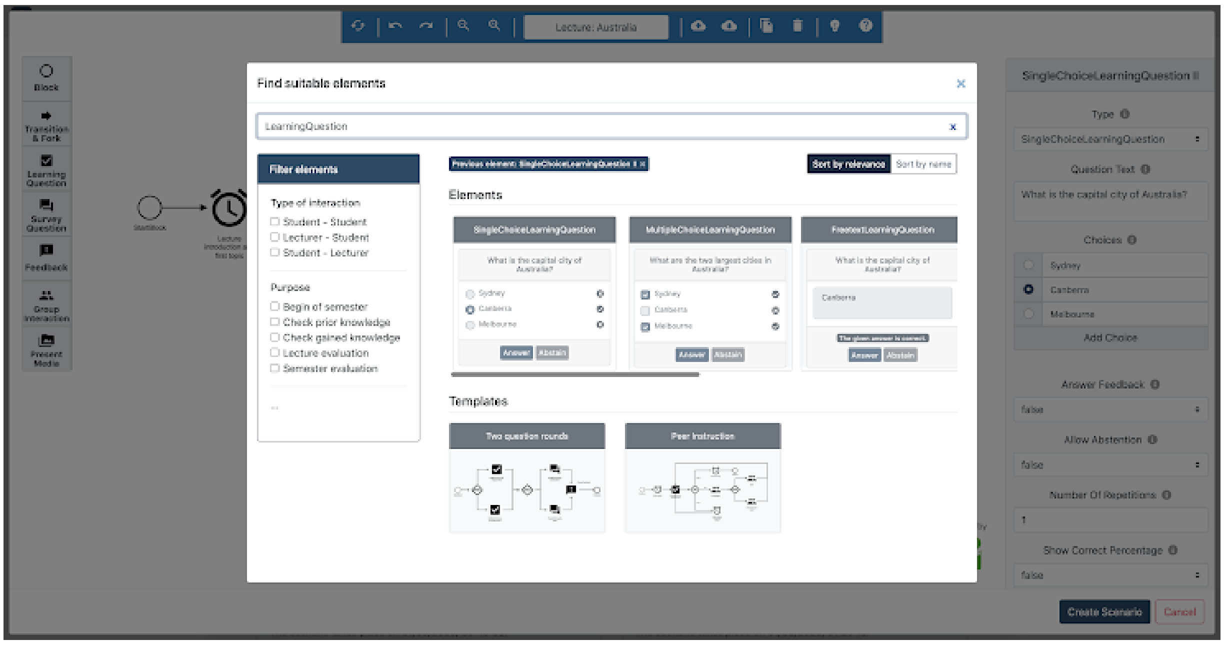
3.2) Initiale Hilfe: Decision Tree

Nach der Auswahl des Glühbirnen-Symbols öffnet sich ein Fenster, das bei der Auswahl von Elementen behilflich sein kann. Anhand von Fragen wird das gewünschte Element ermittelt und anschließend zum Canvas hinzugefügt.

15. Würden Sie in der angezeigten Version vom Decision Tree etwas ändern? *



16. Könnten Sie sich vorstellen, dass der Decision Tree in Zukunft durch einen vorschlagsbasierten, interaktiven Suchdialog ersetzt wird? Welche Vor- und Nachteile sehen Sie hierbei? *



Find suitable elements

LearningQuestion

Previous element: SingleChoiceLearningQuestion

Type of interaction
- ☐ Student - Student
- ☐ Lecturer - Student
- ☐ Student - Lecturer

Purpose
- ☐ Begin of semester
- ☐ Check prior knowledge
- ☐ Check gained knowledge
- ☐ Lecture evaluation
- ☐ Semester evaluation

...

SingleChoiceLearningQuestion: What is the capital city of Australia? Sydney, Canberra, Melbourne

MultipleChoiceLearningQuestion: What are the two largest cities in Australia? Sydney, Canberra, Melbourne

FreetextLearningQuestion: What is the capital city of Australia? Canberra — The given answer is correct.

SingleChoiceLearningQuestion

Type: SingleChoiceLearningQuestion

Question Text: What is the capital city of Australia?

Choices: Sydney, Canberra, Melbourne

Answer Feedback: false

Allow Abstention: false

Number Of Repetitions: 1

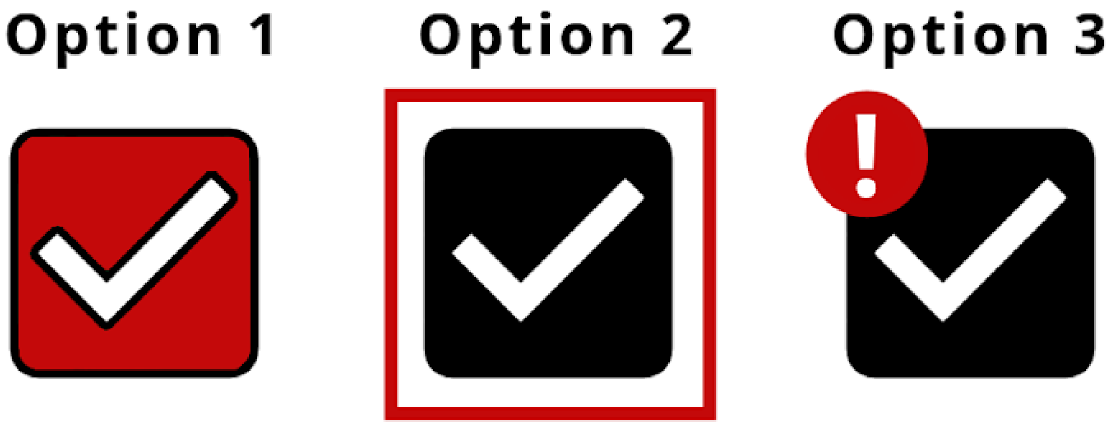
Show Correct Percentage: false

3.3) Initiale Hilfe: Reminder-Nachrichten

Um den Nutzer während der Erstellung eines Szenarios zu unterstützen, könnten kurze Nachrichten zu bestimmten Zeitpunkten eingeblendet werden. Das Klicken auf sie würde eine adäquate Aktion hervorrufen.

17. Haben Sie Änderungsvorschläge oder Ideen für andere Nachrichten? *

18. Wie sollte die Markierung von jeweiligen Blöcken aussehen? *



Markieren Sie nur ein Oval.

- ⬭ Option 1
- ⬭ Option 2
- ⬭ Option 3
- ⬭ Sonstiges: ______________________

4) Unterstützung bei der Erstellung komplexer Workflows

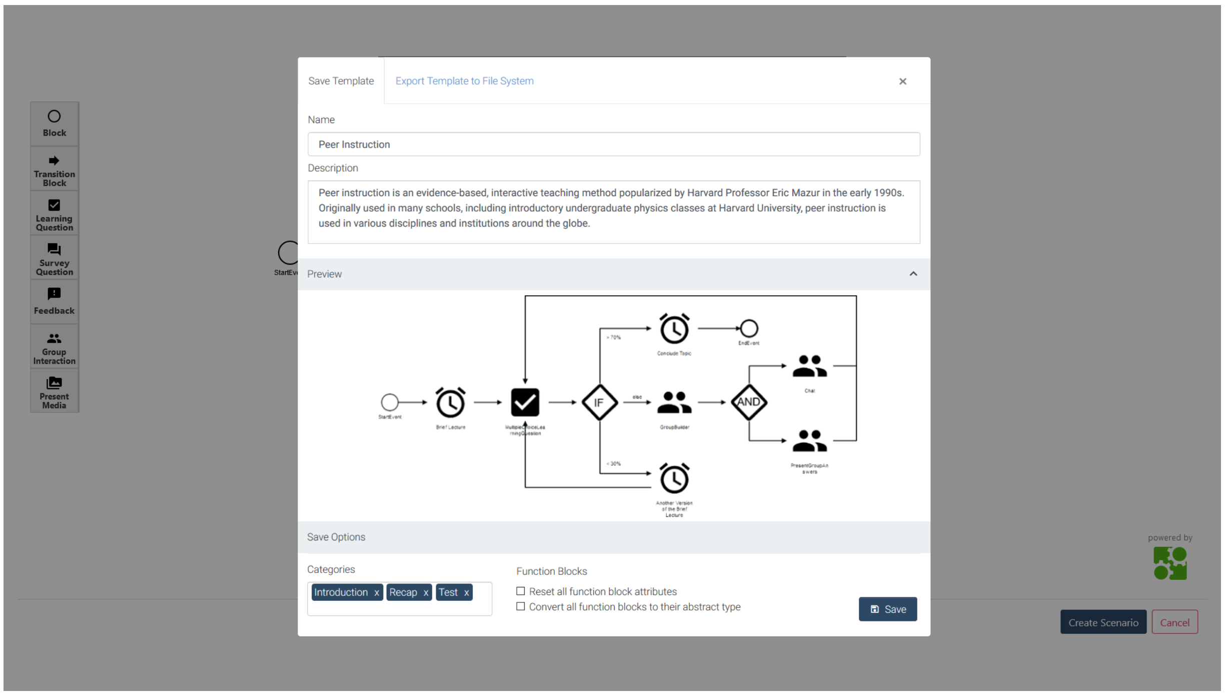
In diesem Abschnitt präsentieren wir Ihnen einige Vorschläge, die die Erstellung von und den Umgang mit komplexen Workflows vereinfachen sollen. Der Fokus liegt hierbei auf Mechanismen, die den Import und Export von Workflows erlauben sowie die Visualisierung und Einbettung importierter Workflow-Inhalte.

Bitte beantworten Sie die jeweiligen Frage-Blöcke, sobald Sie auf den Präsentationsfolien darauf hingewiesen werden.

4.1) Speichern / Exportieren von Templates

Um bestehende Workflows abzuspeichern und zur Wiederverwendung zur Verfügung stellen zu können, wird ein Speichern/Export-Feature benötigt, das im Folgenden skizziert wird.

Speichern



Name

Peer Instruction

Description

Peer instruction is an evidence-based, interactive teaching method popularized by Harvard Professor Eric Mazur in the early 1990s. Originally used in many schools, including introductory undergraduate physics classes at Harvard University, peer instruction is used in various disciplines and institutions around the globe.

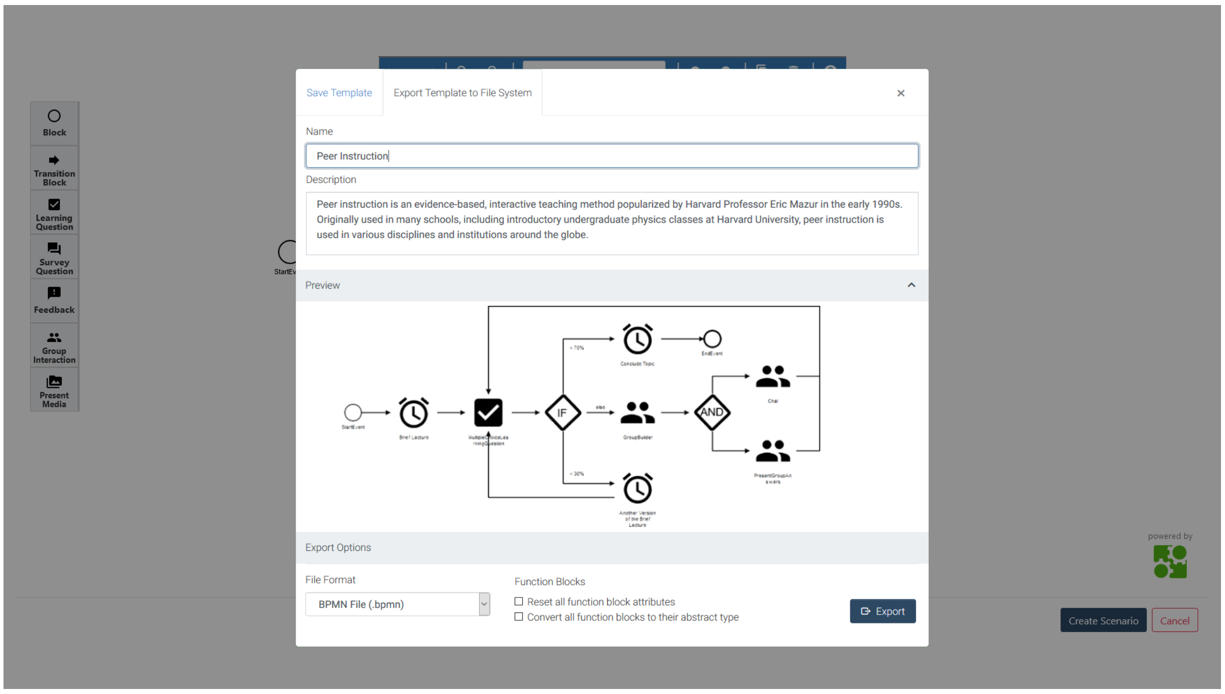
Categories

Introduction x Recap x Test x

Function Blocks

☐ Reset all function block attributes
☐ Convert all function blocks to their abstract type

Exportieren



Name

Peer Instruction

Description

Peer instruction is an evidence-based, interactive teaching method popularized by Harvard Professor Eric Mazur in the early 1990s. Originally used in many schools, including introductory undergraduate physics classes at Harvard University, peer instruction is used in various disciplines and institutions around the globe.

File Format

BPMN File (.bpmn)

Function Blocks

☐ Reset all function block attributes
☐ Convert all function blocks to their abstract type

19. Die Darstellung einer Vorschau des zu speichernden / exportierenden Workflows empfinde ich als sinnvoll. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stimme nicht zu	◯	◯	◯	◯	◯	Stimme voll zu

20. Bitte beantworten Sie die folgenden Fragen zum Category-Selector. *

Markieren Sie nur ein Oval pro Zeile.

	Stimme gar nicht zu.	Stimme eher nicht zu.	Weder noch.	Stimme eher zu.	Stimme voll zu.
Ich empfinde den Category-Selector als angemessenes Bedienelement um Workflow-Komponenten Kategorien zuzuordnen.	◯	◯	◯	◯	◯
Die Darstellung des Category-Selector erscheinen mir intuitiv.	◯	◯	◯	◯	◯
Die Platzierung des Category-Selector empfinde ich als sinnvoll.	◯	◯	◯	◯	◯

21. Das Speichern/Export-Feature bietet die Möglichkeit, alle Funktionsblöcke vor der Aktion in ihre abstrakte Form zu überführen. Halten Sie diese Option für sinnvoll? *

Markieren Sie nur ein Oval.

◯ Ja

◯ Nein

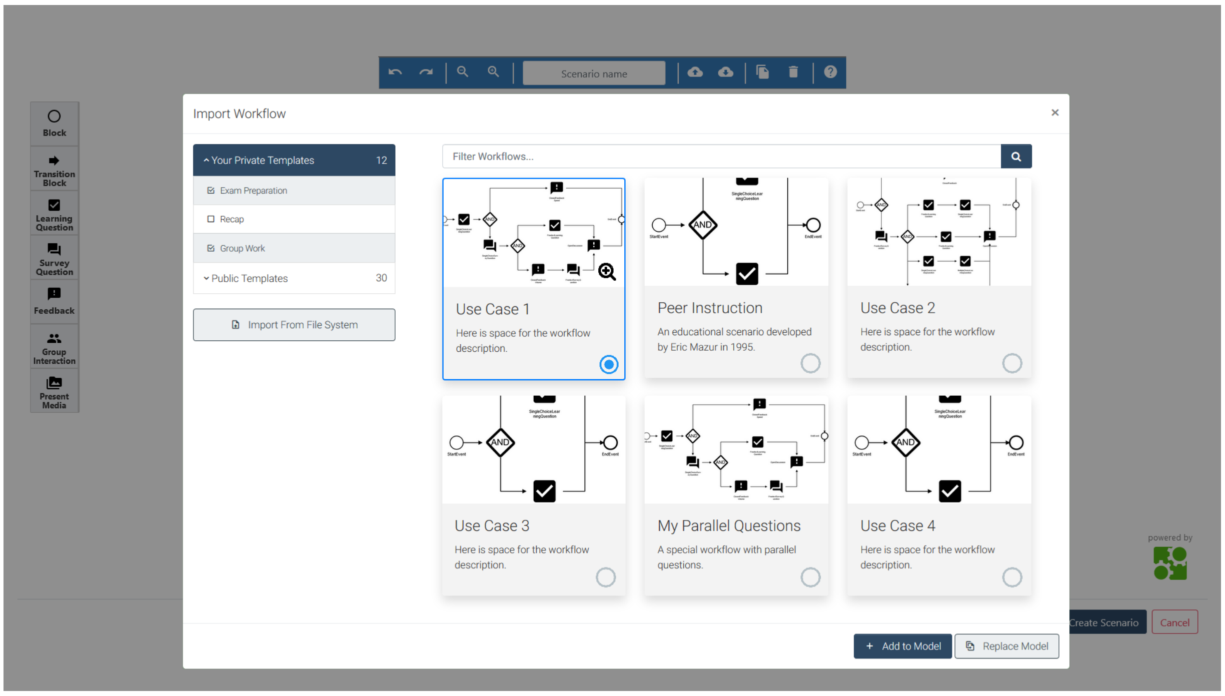
22. Begründen Sie bitte Ihre Antwort. *

23. Welche weiteren Funktionen würden Sie im Speichern/Export-Dialog erwarten?

4.2) Import von Templates

Es soll ein Import-Mechanismus zur Verfügung gestellt werden, der es erlaubt, komplexe Komponenten mit wenig Aufwand wiederverwenden zu können. Im Folgenden wird der nächste Verfeinerungsschritt einers bereits existierenden Ansatzes vorgestellt.

Workflow Import - Variante "Preview Cards"



24. Die Informationen im Import-Dialog werden geordnet und intuitiv dargestellt. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stimme nicht zu.	◯	◯	◯	◯	◯	Stimme voll zu.

25. Bitte beantworten Sie folgende die Sidebar betreffenden Fragen. *

Markieren Sie nur ein Oval pro Zeile.

	Stimme gar nicht zu.	Stimme eher nicht zu.	Weder noch.	Stimme eher zu.	Stimme voll zu.
Das Filtern nach Kategorien über die Sidebar durch Mehrfachauswahl empfinde ich als sinnvoll.	◯	◯	◯	◯	◯
Die farbliche Gestaltung und das Layout der Sidebar empfinde ich als intuitiv.	◯	◯	◯	◯	◯

26. Die Platzierung und farbliche Gestaltung der Aktions-Buttons (Import From File System, Add To Model, Replace Model) empfinde ich als angemessen und intuitiv. *

Markieren Sie nur ein Oval pro Zeile.

	Stimme gar nicht zu.	Stimme eher nicht zu.	Weder noch.	Stimme eher zu.	Stimme voll zu.
Import From File System	◯	◯	◯	◯	◯
Add To Model	◯	◯	◯	◯	◯
Replace Model	◯	◯	◯	◯	◯

27. Welche der vorgeschlagenen Bedienelemente empfinden Sie als unintuitiv oder störend?

28. Der geplante Arbeitsablauf zum Hinzufügen/Importieren von Workflow-Komponenten zu bestehenden Workflows ist intuitiv und einfach gestaltet. *

Markieren Sie nur ein Oval.

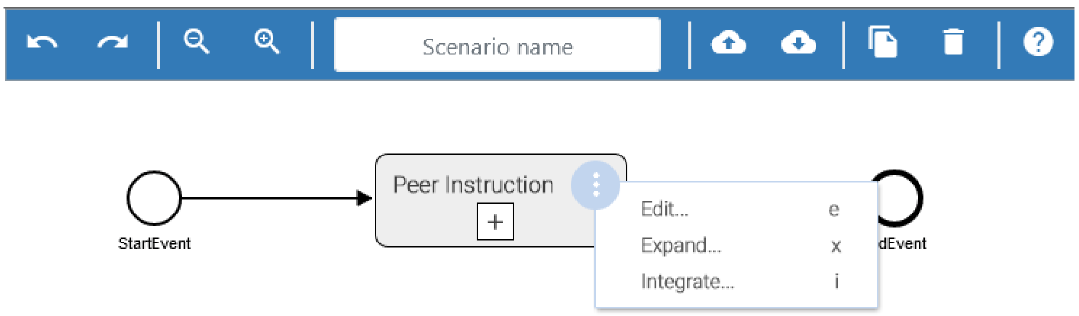
	1	2	3	4	5	
Stimme nicht zu.	◯	◯	◯	◯	◯	Stimme voll zu.

29. Welche Verbesserungsvorschläge haben Sie?

4.3) Visualisierung importierter Elemente

Im Folgenden werden drei verschiedene Ansätze zur Visualisierung und Einbettung importierter Inhalte skizziert.

Reduzierte Ansicht der importierten Komponente "Peer Instruction"

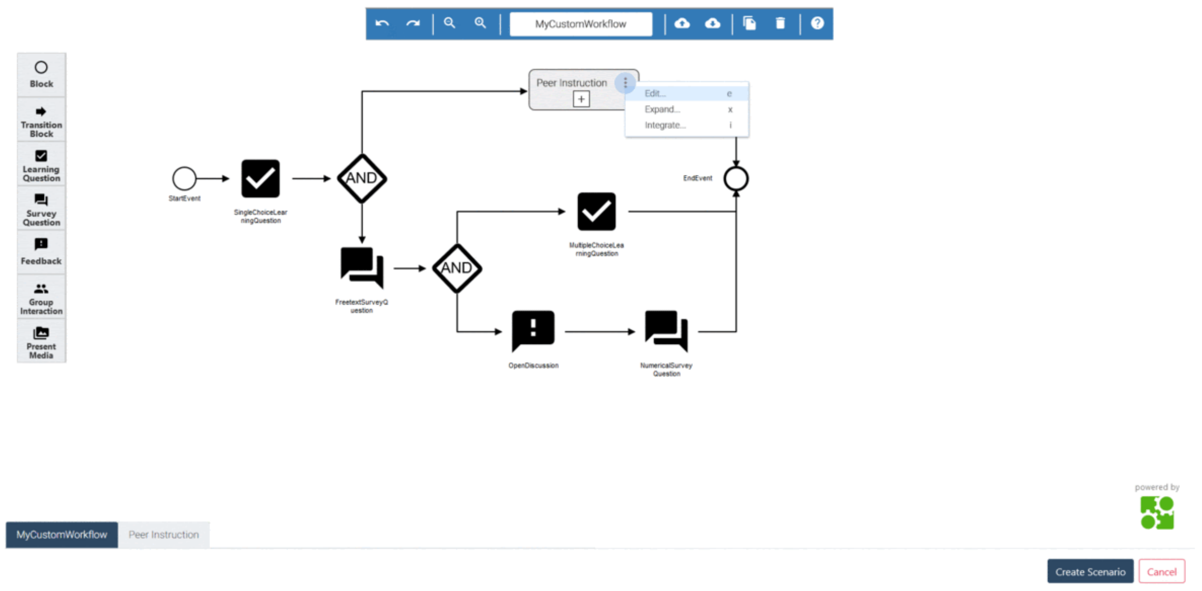


30. Die reduzierte Ansicht für Workflow-Elemente wird auf eine intuitive Art visualisiert. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stimme nicht zu.	⬭	⬭	⬭	⬭	⬭	Stimme voll zu.

Editieren eingebetteter Komponenten

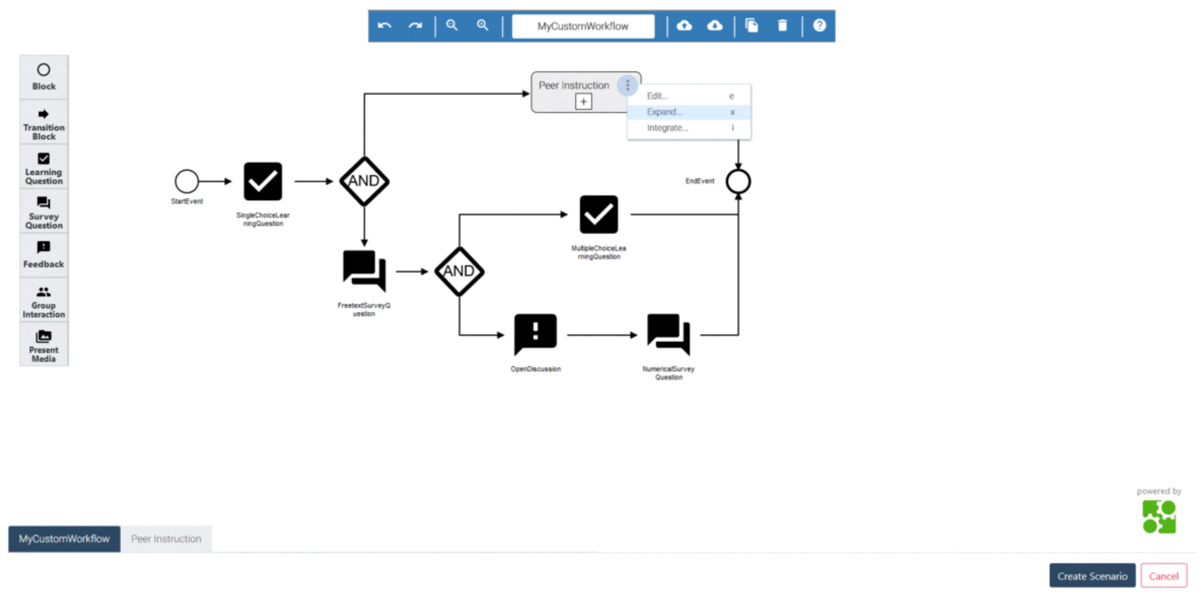


31. Die Art und Weise wie importierte, eingebettete Workflow-Elemente editiert werden können empfinde ich als intuitiv. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stimme nicht zu.	⬭	⬭	⬭	⬭	⬭	Stimme voll zu.

Expandieren eingebetteter Komponenten

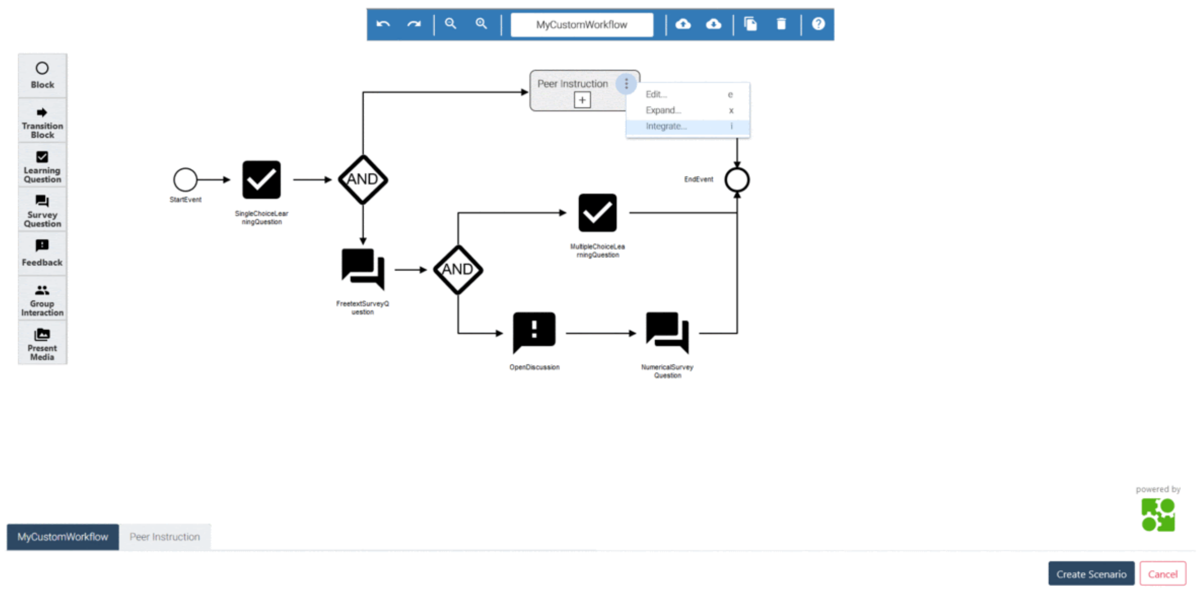


32. Die Art und Weise wie die reduzierte Ansicht von Workflow-Elementen expandiert werden können empfinde ich als intuitiv. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stimme nicht zu.	⬭	⬭	⬭	⬭	⬭	Stimme voll zu.

Integrieren eingebetteter Komponenten



33. Die Art und Weise wie Komponenten in bestehende Workflow-Diagramme integriert werden können empfinde ich als intuitiv. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stimme nicht zu.	◯	◯	◯	◯	◯	Stimme voll zu.

34. Bitte schätzen Sie die Nützlichkeit der im Context-Menu angebotenen Optionen ein.

Markieren Sie nur ein Oval pro Zeile.

	Sehr überflüssig.	Eher überflüssig.	Weder noch.	Eher nützlich.	Sehr nützlich.
Edit	⬭	⬭	⬭	⬭	⬭
Expand	⬭	⬭	⬭	⬭	⬭
Integrate	⬭	⬭	⬭	⬭	⬭

35. Welche Verbesserungsvorschläge haben Sie?

4.4) Konstruktion zusammenhängender Elemente

Die Konstruktion zusammenhängender Elemente wie paralleler Flows soll optimiert werden. Ein Ansatz hierzu wird im Folgenden skizziert.

36. Eine Funktion zur Erstellung mehrerer paralleler Funktionsblöcke, die über den AND-Block verknüpft sind, empfinde ich als sinnvoll. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stimme nicht zu.	⬭	⬭	⬭	⬭	⬭	Stimme voll zu.

37. Für welche weiteren Funktionsblöcke von stARS wünschen Sie sich einen Assistenten für die Erstellung zusammenhängender Komponenten? *

Wählen Sie alle zutreffenden Antworten aus.

- ☐ OR (Konditionaler Block)
- ☐ Zusammenängende Gruppenblöcke
- ☐ Sequentielle Anordnung mehrerer Fragen
- Sonstiges: ☐ ____________________

38. Die Anordnung der Informationen im Create-Connected-Components-Dialog empfinde ich als... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unintuitiv.	○	○	○	○	○	sehr intuitiv.

39. Welche Verbesserungsvorschläge haben Sie?

____

____

____

____

____

Vielen Dank für Ihre Teilnahme

Wir bedanken uns vielmals, dass Sie erfolgreich an dieser Evaluation teilgenommen haben. Bitte vergessen Sie nicht, den Evaluationsfragebogen anschließend abzusenden.

B. Evaluation: Modeling Task Sample Solutions

Task 2:
Three Question Rounds

Task 3:
Variation of embedded structures

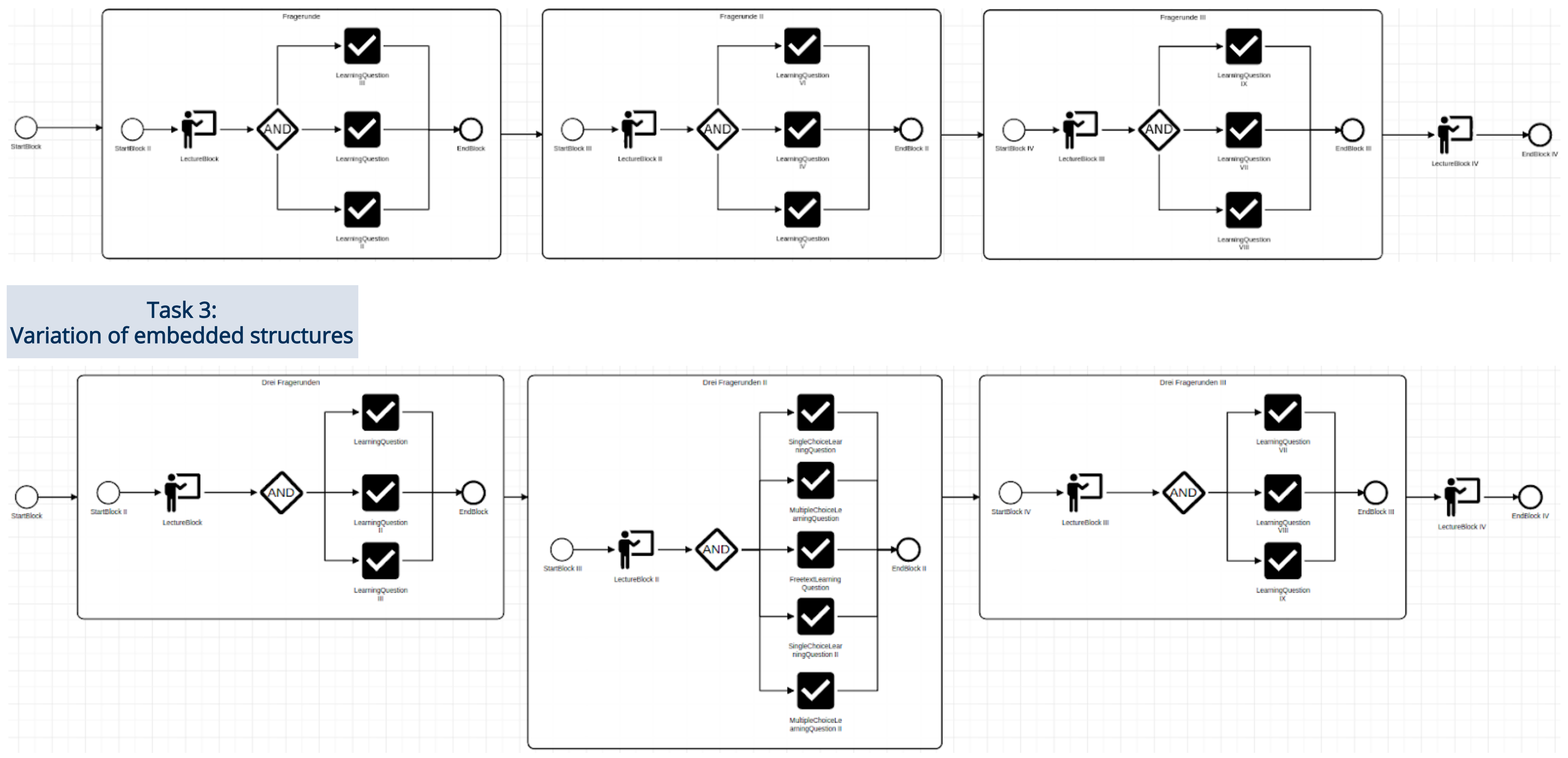


Figure B.1.: Sample solution models for Task 2 and Task 3. In Task 2, the question round template was re-used three times. In Task 3, the structure of the second question round was altered to contain two additional questions.

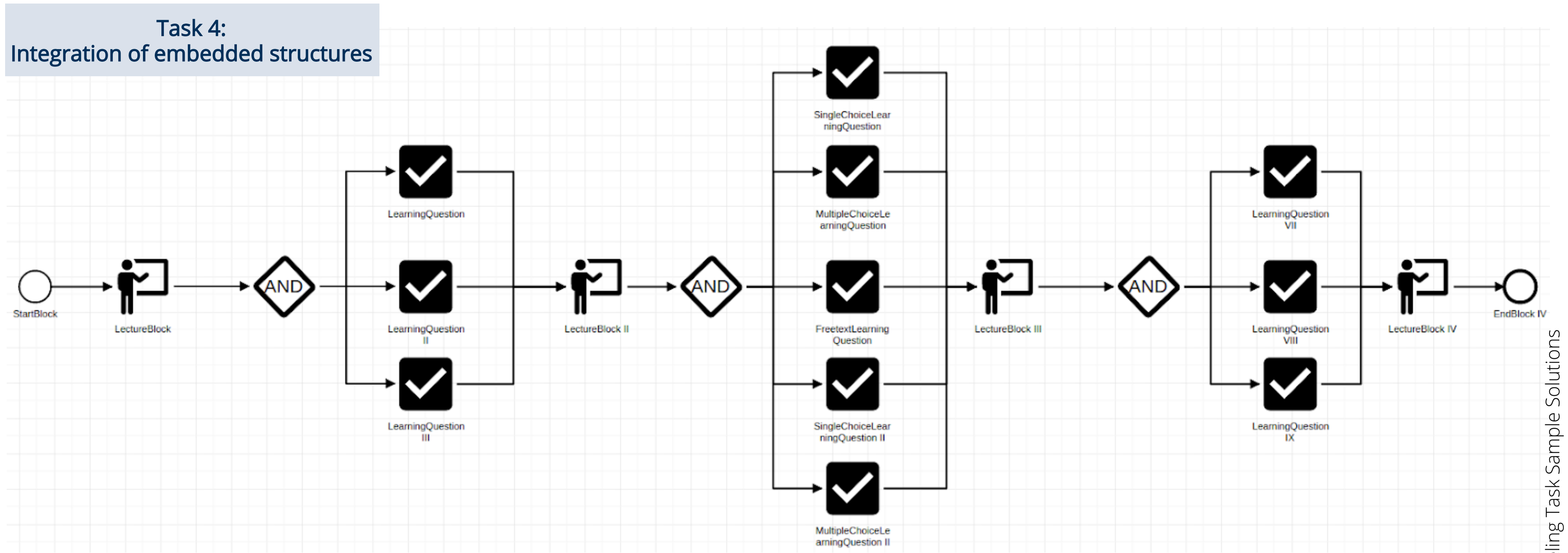


Figure B.2.: Sample solution model for Task 4. In Task 4, the three question rounds were integrated into the rest of the model. The result is a flat model of three consecutive question rounds.

C. Evaluation Questionnaire

Umfrage für die Abschlussevaluation der Masterarbeit „Unterstützung von Lehrenden bei der Erstellung komplexer Lernszenarien in Audience Response Systemen“

Sehr geehrte Damen und Herren,

vielen Dank, dass Sie sich die Zeit nehmen, um an der Evaluation von stARS (scenario-tailored Audience Response System) im Rahmen meiner Masterarbeit

„Unterstützung von Lehrenden bei der Erstellung komplexer Lernszenarien in Audience Response Systemen“

teilzunehmen.

Sie wurden ausgewählt, an dieser Evaluation teilzunehmen, da Sie in der Vergangenheit bereits an einer Evaluation zur Konzeptfindung für die Erweiterung von stARS wertvolles Feedback abgegeben haben.
Damals wurden durch mich neben anderen Vorschlägen folgende Features vorgestellt und durch Sie bewertet:

- Erstellung von Templates
- Import/Load Dialog und Template Repository
- Export/Save Dialog für Templates
- Support für eingebettete Elemente
- Darstellung eingebetteter Elemente durch Tabs

In den folgenden rund 30 - 45 Minuten werden Sie eine Erweiterung des stARS-Editors testen, die den oben genannten Funktionsumfang zur Verfügung stellt. Ich bitte Sie hierbei um Ihr Feedback und Ihre ehrliche Meinung zur Umsetzung der genannten Features.

Ich möchte Sie darauf hinweisen, dass es bei der Evaluation keine richtigen oder falschen Antworten gibt. Sofern Unklarheiten bei der Evaluation auftreten, fragen Sie bitte jederzeit nach.

* Erforderlich

Fragen zur bisherigen Erfahrung mit stARS

Bitte beantworten Sie zunächst einige Fragen bezüglich Ihrer Erfahrung im Umgang mit stARS.

1. Wie vertraut sind Sie mit der Benutzeroberfläche des stARS-Editors? *

Markieren Sie nur ein Oval.

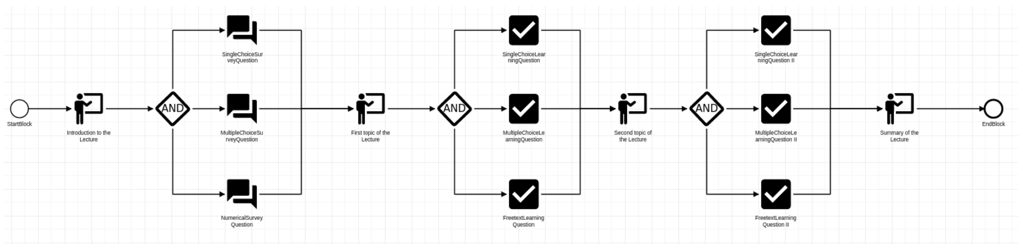
	1	2	3	4	5	
Die Benutzeroberfläche ist mir sehr fremd.	◯	◯	◯	◯	◯	Die Benutzeroberfläche ist mir sehr vertraut.

2. Wie vertraut sind Sie mit der Bedienung des stARS-Editors? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Die Anwendung ist mir sehr fremd.	◯	◯	◯	◯	◯	Die Anwendung ist mir sehr vertraut.

Drei Fragerunden, modelliert mithilfe von stARS


StartBlock
Introduction to the Lecture
AND
SingleChoiceSurveyQuestion
MultipleChoiceSurveyQuestion
NumericalSurvey Question
First topic of the Lecture
AND
SingleChoiceLearningQuestion
MultipleChoiceLearningQuestion
FreetextLearning Question
Second topic of the Lecture
AND
SingleChoiceLearningQuestion II
MultipleChoiceLearningQuestion II
FreetextLearning Question II
Summary of the Lecture
EndBlock

3. Wie zuversichtlich sind Sie, die oben abgebildete Struktur „Drei Fragerunden“ mithilfe von stARS modellieren zu können? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unsicher.	◯	◯	◯	◯	◯	sehr zuversichtlich.

Teil 2: Benutzung des stARS-Editors zur Erstellung eines komplexen Lernszenarios

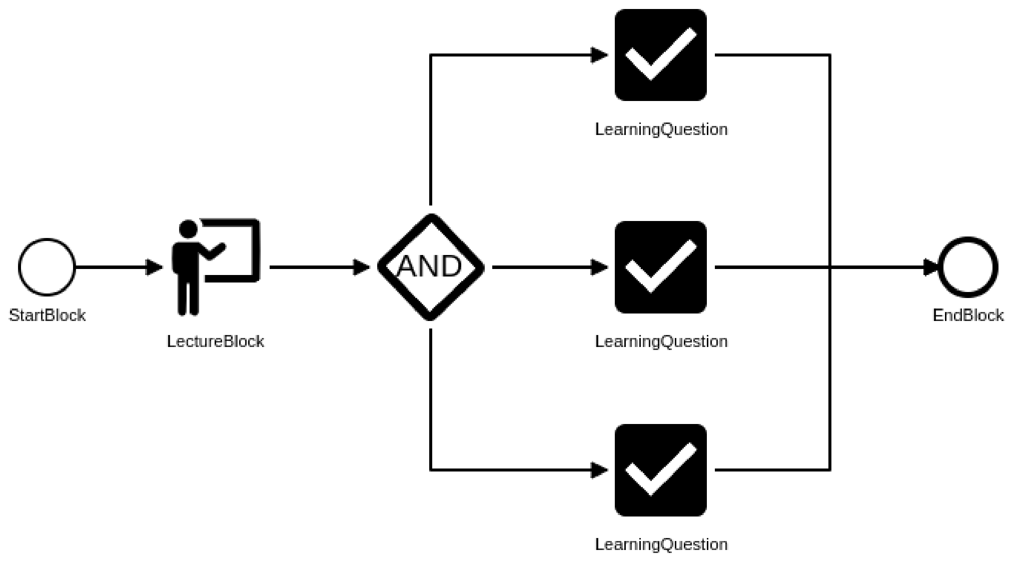
Im diesem Teil der Evaluation bitte ich Sie darum, den stARS-Editor zu benutzen um ein vorgegebenes Szenario zu modellieren. Zur besseren Einschätzung der Benutzbarkeit bitte ich Sie, Ihren Bildschirm mit mir zu teilen.

Bitte navigieren Sie zu nachfolgender URL und loggen Sie sich als Lehrender ein. Nutzen Sie dazu die unten stehenden Zugangsdaten. Öffnen Sie anschließend das Fenster zur Modellierung eines neuen Szenarios.

https://stars-project.com/
Benutzername: evaluation
Passwort: evaluation

Aufgabe 1: Erstellung des Templates „Fragerunde“	In dieser Aufgabe sollen die Möglichkeiten zur Erstellung eines Templates durch Sie getestet werden. Beurteilen Sie bitte anschließend, ob Ihre Erwartungen an das System erfüllt wurden.

Template „Fragerunde“



Aufgabe 1: Erstellung des Templates „Fragerunde”

1a) Erstellen Sie mithilfe des stARS-Editors die oben abgebildete Struktur. Versuchen Sie diese mittels geeigneter Blöcke in stARS abzubilden.

1b) Speichern Sie ihr erstelltes Modell als Template. Vergeben Sie einen Namen und optional eine Beschreibung und erstellen Sie ebenfalls eine neue Kategorie namens „Meine Fragerunden” für Ihr Template. Setzen Sie außerdem alle Funktionsblockeigenschaften zurück und konvertieren Sie alle Funktionsblöcke in ihre abstrakte Form.

4. Die Gestaltung und Visualisierung des Save/Export Dialog entspricht den Erwartungen, die ich nach der ersten Evaluation hatte. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	○	○	○	○	○	Trifft voll und ganz zu.

5. Die Möglichkeiten zur Anreicherung von Templates mit Zusatzinformationen (Name, Beschreibung und Kategorien) entsprechen meinen Erwartungen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

6. Die Bedienung der UI-Elemente, die für das Erstellen von Kategorien zu nutzen sind, war für mich... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unintuitiv.	◯	◯	◯	◯	◯	sehr intuitiv.

7. Die Art und Weise wie die Optionen für das Zurücksetzen von Funktionsblockattributen und das Überführen in abstrakte Funktionsblöcke zur Verfügung gestellt werden entspricht meinen Erwartungen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

8. Die Aufgabe mithilfe der von der Implementierung zur Verfügung gestellten Werkzeuge zu lösen fiel mir... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
besonders schwer.	◯	◯	◯	◯	◯	besonders leicht.

9. Wie zufrieden waren Sie mit dem Arbeitsablauf und dem Endergebnis dieser Aufgabe? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unzufrieden.	◯	◯	◯	◯	◯	sehr zufrieden.

10. Sonstige Ergänzungen:

____

____

____

____

____

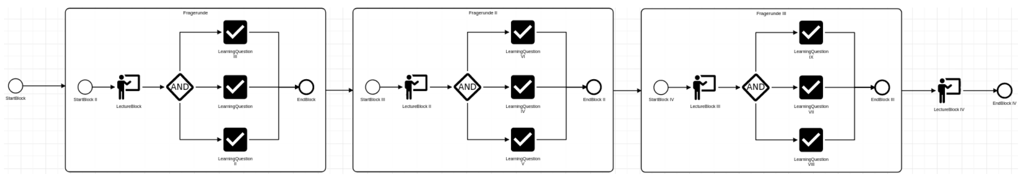
Aufgabe 2: Erstellung des Modells „Drei Fragerunden“

In dieser Aufgabe liegt der Fokus auf der Komposition komplexer Modelle aus existierenden Bausteinen.

Das Template „Fragerunde“, welches in der ersten Aufgabe erstellt wurde, wird nun wiederverwendet, um das Modell „Drei Fragerunden“ zu erstellen. Hierbei werden Sie neue Features nutzen, die es erlauben, bestehende Templates miteinander zu komplexeren Modellen zu verknüpfen.

Bitte lösen Sie die folgende Aufgabe und bewerten Sie anschließend die Umsetzung im Hinblick auf die vergangene Evaluation.

Aufgabe 2: Erstellung des Modells „Drei Fragerunden“



Aufgabe 2: Modellkomposition

Das Template „Fragerunde“, welches in der ersten Aufgabe erstellt wurde, wird nun wiederverwendet, um das Modell „Drei Fragerunden“ zu erstellen. Hierbei werden Sie neue Features nutzen, die es erlauben, bestehende Templates miteinander zu komplexeren Modellen zu verknüpfen.

2a) Setzen Sie den Modellierungscanvas zurück.

Ziel der nächsten Teilaufgabe ist es durch Nutzung des Templates, welches Sie in Aufgabe 1 erstellt haben, die oben dargestellte Modellstruktur zu erstellen. Bitte gehen Sie, um den Test verschiedener Features zu ermöglichen, bei der nächsten Aufgabe folgendermaßen vor:

2b) Importieren Sie zweimal das von Ihnen erstellte Template in die Zeichenfläche. Versuchen Sie, die dritte Fragerunde durch Duplizieren einer anderen Fragerunde zu erstellen. Versuchen Sie dann, die fehlenden Verbindungen und Elemente im Modell zu ergänzen.

11. Die Gestaltung und Visualisierung des Import/Load Dialog entspricht den Erwartungen, die ich nach der ersten Evaluation hatte. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	⬭	⬭	⬭	⬭	⬭	Trifft voll und ganz zu.

12. Das Verhalten der Bedienelemente im Import/Load Dialog entspricht meinen Erwartungen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	⬭	⬭	⬭	⬭	⬭	Trifft voll und ganz zu.

13. Ich konnte mithilfe des Import/Load Dialog sehr schnell das Template finden, nach dem ich gesucht habe. *

Markieren Sie nur ein Oval.

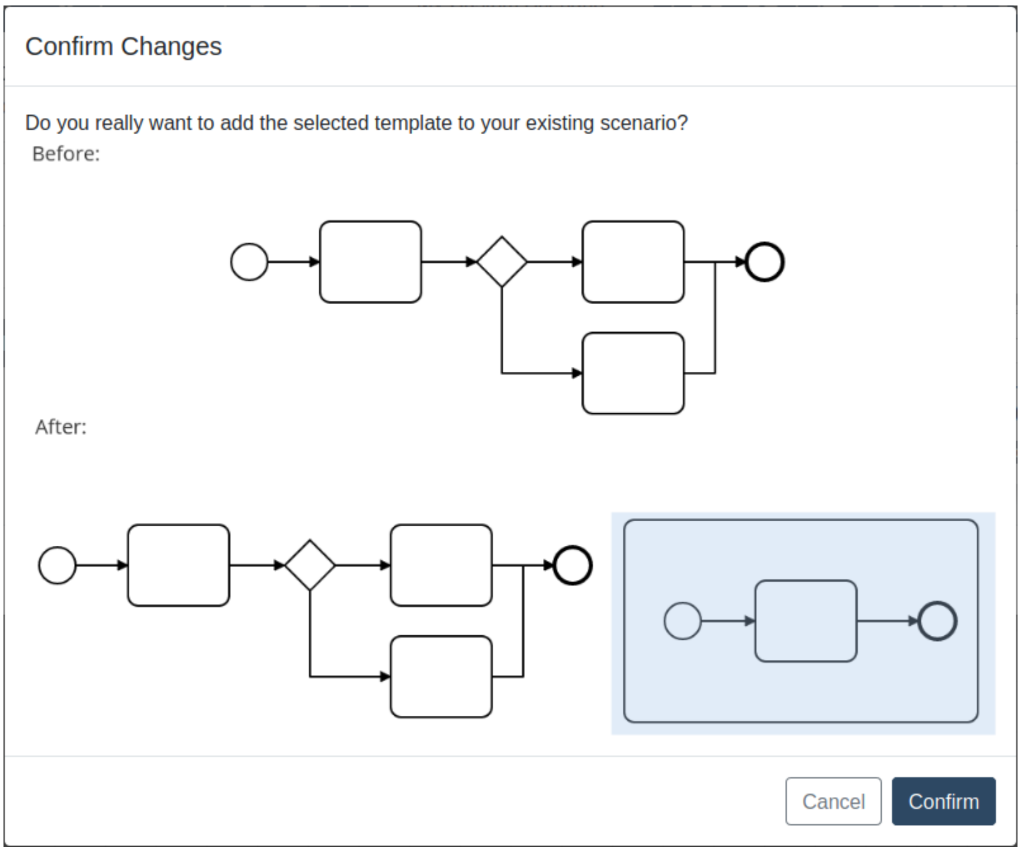
	1	2	3	4	5	
Trifft überhaupt nicht zu.	⬭	⬭	⬭	⬭	⬭	Trifft voll und ganz zu.

14. Das Verhalten der „Add To Model“ Funktion empfinde ich als... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unintuitiv.	⬭	⬭	⬭	⬭	⬭	sehr intuitiv.

Vorschlag für Illustration zur Erklärung der „Add To Model“ Funktion


Confirm Changes

Do you really want to add the selected template to your existing scenario?

Before:

After:

15. In Bezug auf obige Abbildung: Eine erklärende Visualisierung für die Add To Model Funktion hätte mir beim Verständnis dieser Funktion geholfen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	⬭	⬭	⬭	⬭	⬭	Trifft voll und ganz zu.

16. Die Visualisierung eingebetteter Templates ist gelungen und entspricht meinen Erwartungen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	⬭	⬭	⬭	⬭	⬭	Trifft voll und ganz zu.

17. Der Umgang mit eingebetteten Elementen (Verschieben, Verbinden, Duplizieren) gestaltete sich für mich als einfach. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

18. Die Aufgabe mithilfe der von der Implementierung zur Verfügung gestellten Werkzeuge zu lösen fiel mir... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
besonders schwer.	◯	◯	◯	◯	◯	besonders leicht.

19. Wie zufrieden waren Sie mit dem Arbeitsablauf und dem Endergebnis dieser Aufgabe? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unzufrieden.	◯	◯	◯	◯	◯	sehr zufrieden.

20. Sonstige Ergänzungen:

____

____

____

____

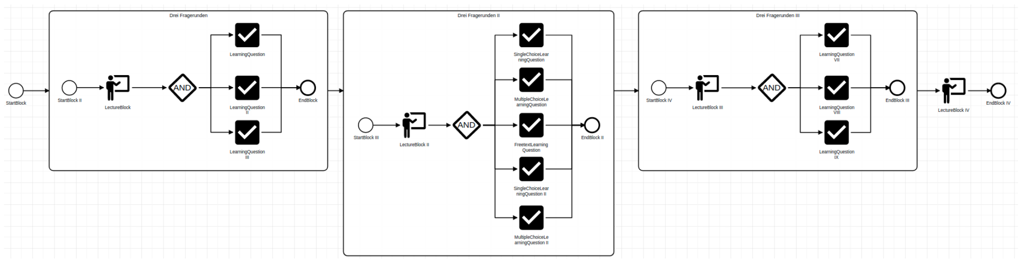
____

Aufgabe 3: Variation des Modells „Drei Fragerunden“

Im vorherigen Abschnitt haben Sie drei Fragerunden miteinander verbunden und ein valides Modell erstellt. Nun geht es darum, die einzelnen Fragerunden auf individuelle Bedürfnisse anzupassen.

Bitte lösen Sie die folgende Aufgabe und bewerten Sie anschließend die Umsetzung im Hinblick auf die vergangene Evaluation.

Aufgabe 3: Variation des Modells „Drei Fragerunden“



Aufgabe 3: Variation eingebetteter Modellbestandteile

3a) Formulieren Sie beispielhaft für zwei bis drei der Fragen in Ihrem Modell Frage- und Antwortmöglichkeiten. Ändern Sie außerdem den Fragetyp für zwei bis drei Fragen Ihrer Wahl.

3b) Versuchen Sie für die folgende Aufgabe mit einer isolierten Ansicht für die zweite Fragerunde zu arbeiten: Ändern Sie bitte die Struktur der mittleren Fragerunde: Fügen Sie eine Single-Choice- und eine Multiple-Choice-Lernfrage hinzu.

3c) Stellen Sie nun eine kompaktere Modellvisualisierung für die dritte Fragerunde her. Versuchen Sie, diese Fragerunde einzuklappen/zu minimieren.

21. Die angebotenen Tools zur Modifikation von eingebetteten Templates entsprechen meinen Erwartungen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	⬭	⬭	⬭	⬭	⬭	Trifft voll und ganz zu.

22. Die Gestaltung und Visualisierung der Tabs für eingebettete Templates entspricht den Erwartungen, die ich nach der ersten Evaluation hatte. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	⬭	⬭	⬭	⬭	⬭	Trifft voll und ganz zu.

23. Ich hatte keine Probleme damit, in eine isolierte Modellansicht für eine der Fragenrunden zu wechseln. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

24. Wie eingebettete Templates minimiert und expandiert werden können, empfinde ich als... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unintuitiv.	◯	◯	◯	◯	◯	sehr intuitiv.

25. Die Aufgabe mithilfe der von der Implementierung zur Verfügung gestellten Werkzeuge zu lösen fiel mir... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
besonders schwer.	◯	◯	◯	◯	◯	besonders leicht.

26. Wie zufrieden waren Sie mit dem Arbeitsablauf und dem Endergebnis dieser Aufgabe? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unzufrieden.	◯	◯	◯	◯	◯	sehr zufrieden.

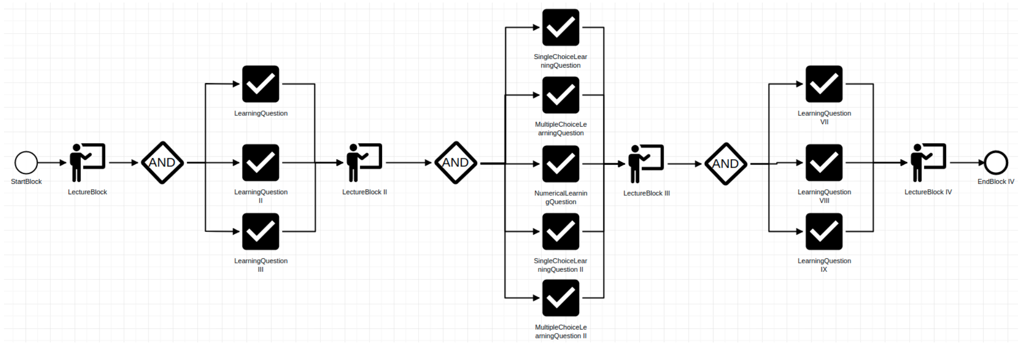
27. Sonstige Ergänzungen:

Aufgabe 4: Finalisierung des Templates „Drei Fragerunden“

Im vorherigen Abschnitt haben Sie die Anpassungsmöglichkeiten für eingebettete Templates genutzt, um die Fragerunden variieren zu lassen. In dieser Aufgabe geht es darum, das Modell endgültig in die geforderte Form zu überführen. Des Weiteren sollen die Funktionen zum Austausch von Templates zwischen Kollegen durch Sie getestet werden.

Bitte lösen Sie die folgende Aufgabe und bewerten Sie anschließend die Umsetzung im Hinblick auf die vergangene Evaluation.

Aufgabe 4: Finalisierung des Templates „Drei Fragerunden“



Aufgabe 4: Finalisierung des Templates „Drei Fragerunden“

4a) Integrieren Sie nun nacheinander alle drei Fragerunden in den Rest des Modells. Der Begriff „Integrieren“ meint hierbei das Auflösen der Eltern-Kind-Relation von eingebetteten Modellen.

Die folgenden Arbeitsschritte sollen nun beispielhaft einen denkbaren Ablauf illustrieren, der das Teilen von Templates mit Kollegen zum Thema hat.

4b) Versetzen Sie sich zunächst in die Lage einer Lehrperson, die ein Template für das Modell „Drei Fragerunden“ einem Kollegen zur Verfügung stellen möchte. Dieser Vorgang soll in Form eines Dateiaustausches stattfinden. Versuchen Sie zunächst, das Template herunterzuladen. Exportieren Sie hierzu das Template als .bpmn-Datei und speichern Sie es auf Ihrem Computer.

4c) Versetzen Sie sich nun bitte in die Lage einer Lehrperson, die von einem durch einen Kollegen zur Verfügung gestellten Template profitieren möchte. Sie haben dieses Template beispielsweise per Mail erhalten. Versuchen Sie, das im vorangegangenen Schritt exportierte Template nun in die Zeichenfläche von stARS zu importieren.

Die letzte Teilaufgabe soll abschließend einige Features zum Löschen von Templates und Kategorien evaluieren.

4d) Versuchen Sie, das Template „Drei Fragerunden“ wieder zu löschen. Versuchen Sie außerdem, die Kategorie „Meine Fragerunden“ zu löschen, die Sie in der ersten Aufgabe erstellt haben.

28. Die Umsetzung der Integrationsfunktion für eingebettete Modellbestandteile entspricht meinen Erwartungen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

29. Ich habe den Button zum Integrieren von eingebetteten Modellbestandteilen leicht finden können. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

30. Die „Integrate Scenario“ Funktion verhält sich so, wie ich es erwarten würde. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

31. Den Arbeitsablauf zum Exportieren eines Modells als Template ins lokale Dateisystem empfinde ich als... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unintuitiv.	◯	◯	◯	◯	◯	sehr intuitiv.

32. Die Implementierung der Export-Funktion entspricht den Erwartungen, die ich nach der ersten Evaluation hatte. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

33. Mir fiel es leicht, die Funktion zum Löschen von Templates zu finden, *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

34. Mir fiel es leicht, die Funktion zum Löschen von Kategorien zu finden, *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

35. Mir fiel es schwer, die Funktion für das Importieren eines Templates aus dem lokalen Dateisystem zu finden. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

36. Die Implementierung der Import-From-File-System Funktion entspricht den Erwartungen, die ich nach der ersten Evaluation hatte. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

37. Die Vorschau für ein ausgewähltes, vom Dateisystem zu importierendes Template entspricht meinen Erwartungen. *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Trifft überhaupt nicht zu.	◯	◯	◯	◯	◯	Trifft voll und ganz zu.

38. Die Aufgabe mithilfe der von der Implementierung zur Verfügung gestellten Werkzeuge zu lösen fiel mir... *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
besonders schwer.	◯	◯	◯	◯	◯	besonders leicht.

39. Wie zufrieden waren Sie mit dem Arbeitsablauf und dem Endergebnis dieser Aufgabe? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
sehr unzufrieden.	◯	◯	◯	◯	◯	sehr zufrieden.

40. Sonstige Ergänzungen:

Vielen Dank für Ihre Teilnahme!

Ich bedanke mich bei Ihnen für Ihre erfolgreiche Teilname an dieser Evaluation. Bitte vergessen Sie nicht, die Antworten Ihrer Evaluation abzuschicken.