stARS: Proposing an Adaptable Collaborative Learning Environment to Support Communication in the Classroom

Tommy Kubica, Ilja Shmelkin, Robert Peine, Lidia Roszko and Alexander Schill
Faculty of Computer Science, Technische Universität Dresden, Dresden, Germany
{*firstname.lastname*}*@tu-dresden.de*

Keywords: Learning Environment, Audience Response System, Backchannel System, stARS.

Abstract: The usage of technology provides a powerful opportunity to support classic classroom scenarios. In addition to improve the presentation of a lecturer's content, technical tools are able to increase the communication to the students or between students. Although many approaches exist that are able to support such interactions, the lecturer has to adjust his/her teaching strategy to the corresponding system. To overcome this problem, our goal is to allow lecturers to create their personal scenarios in an intuitive manner. As a solution, we propose an approach called stARS[a] that builds on top of a uniform (meta-)model. It provides a graphical editor as a user interface to create customized application models that represent teaching scenarios. In addition to classic *Audience Response* functions such as learning or survey questions, collaborative functionality is provided – specifically, group formations with associated interactions within these groups (e.g., discussion or voting functionalities) are examined. In order to evaluate our approach, in the first step, a user study was conducted to reason about the average user's modeling abilities with the graphical editor. Next, we target to evaluate both the functionality and the opportunities of our created prototype in real-life scenarios.

[a]scenario-tailored Audience Response System.

1 INTRODUCTION

In recent years, technology has increasingly found its way into teaching. While technical tools that allow to present content to the students are omnipresent, the potential of technology to improve communication in the classroom is still a subject of research. Although a lot of investigations have been conducted in the past, e.g., (Lingnau et al., 2003) or (Dragon et al., 2013), they were limited to small, non-anonymous scenarios.

Current approaches such as *Audience Response Systems*, *Classroom Response Systems*, or *Backchannel Systems* overcome those limitations by targeting to involve students anonymously using their personal mobile devices. They allow students to answer prepared questions or to ask their own questions during the ongoing lecture that can be discussed with other students. A variety of systems exist, e.g., as listed by (Hara, 2016), (Meyer et al., 2018) or (Kubica et al., 2019a). As we do not need to distinguish different types of systems in this paper, we will use the generic term *learning environments* from now on.

Although the usage of these systems provides a lot of promising opportunities to improve classroom teaching (Nikou and Economides, 2018), they suffer from heterogeneity. Instead of implementing their teaching strategy in mind, lecturers have to adapt their strategy to the system's limited functional scope and it's predefined settings, e.g., the number of repetitions a student got to answer a question correctly.

Accordingly, our goal is to allow lecturers to configure the system's functionality to their personal teaching strategy. As a solution, we present the prototype of an approach called *stARS*[1], which gives lecturers the opportunity to (1) adapt function blocks by different parameters, (2) build sequences of function blocks and add conditions in order to define different learning paths, (3) link function blocks to cooperate and (4) support collaboration between students by introducing novel function blocks for group formations and associated interactions. A user study will be discussed, showing that users with different modeling abilities are able to create customized scenarios. In addition, the usage within realistic scenarios is motivated to show the applicability and opportunities of our presented approach.

[1]The running prototype is provided on: https://stars-project.com/ (accessed 3/19/20).

The remainder of this paper is structured as follows. Section 2 presents related work while specifically targeting the customization of functionality in these systems. In section 3, we present preliminary work on the (meta-)model and the base on the system's concept. Next, section 4 introduces an infrastructure and describes each part of it in more detail. In addition, the opportunities provided for the lecturer are motivated. Section 5 presents the results retrieved from an initial user study and discusses the evaluation within realistic scenarios. Finally, section 6 concludes this paper and discusses the limitations and future work to be done.

2 RELATED WORK

Current literature presents many systems that target the support communication in classic classroom teaching, e.g., as listed by (Hara, 2016), (Meyer et al., 2018), or (Kubica et al., 2019a). In order to adjust the functional scope to the lecturer's personal teaching strategy, different approaches exist, which will be presented in the following.

The most straightforward way to customize the range of functions is to select specific functions. E.g., in Tweedback[2], the lecturer can enable or disable the features called "Chatwall", "Quiz" and "PanicButtons". Furthermore, GoSoapBox[3] allows a more fine-granular selection of the system's functionality by extending the selection of its core features called "Barometer", "Quizzes", "Polls", "Instant Polling", "Discussion" and "Social Q&A" by additional features (or filters), namely a "Profanity Filter", "Math Formatting" and "Names Required". In order to help the lecturer to choose the system's functional scope, ARSnova[4] allows to select predefined use cases. E.g., by selecting "Clicker Questions", all simple question types such as *Multiple Choice*, *Single Choice*, *Yes – No*, *Likert Scale* and *Grading* are enabled. Furthermore, (Kubica et al., 2017) investigates a proposal-based function selection that is able to establish a connection between the considered scenario and the system's functional scope. The scenario is characterized by entering values for predefined influence factors, e.g., the amount of students participating in the lecture. Afterwards, a suitable functional scope is proposed that can be adjusted manually to match the lecturer's personal preferences.

Although the mentioned function selections help at targeting the function scope at a certain degree, they lack at their limited functional scope as well as at predefined limitations, e.g., the number of repetitions to answer a question or the feedback whether a possible correct answer is displayed to the students or not.

(Schön, 2016) presents an approach that goes beyond a static functional scope by proposing a new generic model that allows to define customized teaching scenarios. The model consists of *objects* with *attributes*, and *rules* with *conditions* and *actions*. Due to the high generics of the model, almost any scenario can be created. In MobileQuiz2[5], the model was implemented and evaluated in several realistic scenarios. During the evaluation, the modeling task turned out to be very complex. Although a scenario editor did help the users to model valid scenarios, it could not make the modeling process easier to understand. For this reason, a didactic expert is required to define the application model of a custom scenario when MobileQuiz2 is used. Another issue got obvious during execution. Due to the problem that the generic model produces deep-nesting objects, it lacks in performance as soon as the participation count raises.

In summary, two directions can be recognized: On the one side, approaches exist targeting the system's functional scope by function selections. On the other side, a generic model was proposed that is able to express scenarios without predefined elements. Nevertheless, both groups have their individual limitations, which motivated us to create an approach that overcomes those and surpasses existing approaches. In order to give any lecturer the opportunity to target the function scope to his/her teaching strategy in mind, collaborative functional blocks will to be provided, which have so far only been investigated in smaller, non-anonymous scenarios, e.g., as presented by (Lingnau et al., 2003) or (Dragon et al., 2013).

3 PRELIMINARY WORK

This section presents preliminary work that was done in advance of the prototype creation. First, the concept of an adaptable learning environment is described. Second, the fundament of our concept is presented, namely a (meta-)model for defining elements, parameters, and rules.

3.1 Concept of an Adaptable Learning Environment

Our main concept combines the strength of both approaches, the application models with static func-

[2]https://tweedback.de/?l=en (accessed 3/19/20).

[3]https://www.gosoapbox.com/ (accessed 3/19/20).

[4]https://arsnova.eu/mobile/ (accessed 3/19/20).

[5]http://www.mobilequiz.org (accessed 3/19/20).

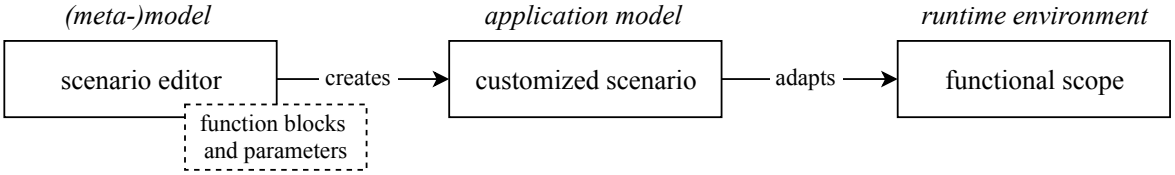

Figure 1: The concept of an adaptable learning environment that allows to create and execute customized scenarios in order to support lecturers' personal teaching strategies. (Kubica, 2019)

tional scopes and the flexible generic models allowing for highly customizable scenarios. In addition, it focuses on solving the respective limitations of these approaches. Therefore, it builds on ideas derived from two concepts, namely *Model-Driven Software Development* (MDSD) and *End User Development* (EUD).

MDSD is described as the generation of software from models, whereas the complexity of the model is significantly easier to understand than the generated code. The syntax of such models and the interrelationships between elements are typically defined by (meta-)models. (Stahl et al., 2007)

According to (Sendall and Kozaczynski, 2003), different types of methods to transform the model to a running software can be recognized, whereby the *intermediate representation*, meaning that the model is exported in a standardized form (e.g., XML or JSON) that can be used by external tools, is the most promising option for our concept.

Motivated by the results on modeling from (Schön, 2016), one major goal of our concept will be to provide an intuitive opportunity for end-users (i.e., lecturers with different abilities in computer science) to customize their scenarios. This is strongly related to the discipline of EUD, which is defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact" (Lieberman et al., 2006). E.g., instead of entering code as it is done in classic programming languages, the user will be able to compose visual elements, e.g., blocks, and connect or link them, resulting in a reduced complexity. Our concept adopts this idea in a graphical editor that gives lecturers the opportunity to customize their teaching scenarios (resulting in *application models*).

In order to execute these application models, our concept includes a runtime environment that needs to interpret those and target its functional scope and settings accordingly. Since performance is key in learning environments, especially during real-time functionality (e.g., learning or survey questions), it has to be designed in a way that can handle sudden changes in resource demand; hence we propose a scalable infrastructure.

To summarize, the overall concept consists of different components. First, a (meta-)model has to be created that defines the system's elements with their parameters and rules for relationships. Next, a graphical editor, which builds on top of the (meta-model) will support the lecturer in creating customized scenarios (i.e., application models). Finally, a scalable infrastructure has to be created, which is able to interpret these application models. Figure 1 summarizes the concept in a graphical manner.

3.2 stARS (Meta-)model

The processes which take place during a lecture are similar to the concept of workflows. As the lecture moves on, events happen (e.g., students give answers to questions, or a group session takes place) that allow to decide how the lecture is continued.

The previously presented concept is centered around a (meta-)model, which allows to create workflow models. Each workflow (i.e., the derived application model) represents one specific lecture and can be created by the lecturer with the help of a graphical editor. Which workflow elements exist and how they connect to each other is defined explicitly by the (meta-)model. Furthermore, each element has a set of parameters that describe it more concretely. As workflows have similar structural elements across multiple known modeling languages, our (meta-)model incorporates those conventions. Therefore each derived workflow uses exactly one *start node* and an arbitrary amount of *end nodes*. In between, an arbitrary amount of *function blocks* can be used to describe a specific lecture. All elements (i.e., *start nodes*, *end nodes* and *function blocks*) are connected by *transitions*, which determine the order of the activities of a lecture.

During design time, a lecturer cannot foresee how a lecture will proceed, hence different types of transitions exist:

- The *OR-fork* is used, when multiple paths in the workflow can be taken based on the outcome of a connected function block,
- *AND-forks* are used to design parallel activities, by splitting the control-flow into several sub-flows

- and a *join* connects several sub-flows again.

Each function block represents a unique functionality of a learning environment. In general, we distinguish seven groups of functions, each having several sub-functions and a number of parameters for configuration:

- *learning questions* have one or more correct answer(s) and can be solved by the students,
- *survey questions* are used to poll an opinion of the students,
- an *open discussion* allows students to ask own questions and discuss them with other students,
- *closed feedback* gives students the opportunity to provide instant feedback on predefined feedback dimensions,
- *group interactions* form groups of students and provide interactions within these groups,
- *result presentation* displays a result on students' devices
- and *media presentation* is used to display a media content on those.

Finally, a *pause block* exists, which allows lecturers to build breaks into their workflows and enable the modeling of a complete 90 minute lecture. A more detailed description of the (meta-)model's structure and which (sub-)functions and parameters it provides is presented in (Kubica et al., 2019b).

4 stARS PROTOTYPE

As motivated by (Bruff, 2019), each lecturer coming to class has a teaching strategy in mind. This results in lecturers that want to be able to use the learning environment in a way so that it supports this strategy. This section introduces our stARS prototype that targets to accomplish this task by implementing the previously described concept. First, a scalable infrastructure is presented, which can handle sudden changes in resource demand by design. Afterward, each part of this infrastructure is described in more detail. Last, the options for the lecturer are summarized.

4.1 Infrastructure

System performance is key to provide a good user experience. Although teaching scenarios can range from simple classroom teaching (i.e., 10 to 30 students) to crowded lectures (i.e., 1000 students or more), a system has to be able to deliver constant performance, even when multiple sessions take place simultaneously. Performance from monolithic applications can suffer during heavy load (i.e., "slash-dot effect"). Therefore, to provide a good user experience constantly, it is necessary to create a scalable distributed infrastructure that is able to run multiple application models simultaneously without interference. The infrastructure was proposed in (Kubica et al., 2019b) and consists out of:

- A *backend server* to provide access to the (meta-)model, the database and administrative functions,
- a *graphical editor frontend* for the lecturer to create application models and start or stop scenarios,
- an arbitrary amount of *cloud servers* which run the individual application models based on a dockerized runtime
- and a *user interface* for the students to participate in scenarios.

4.2 Backend Server

The backend represents the main entry point of administrative components. It allows administrators to manage the registration of cloud servers, i.e., new cloud servers can be added, or existing ones can be removed. Furthermore, it allows lecturers to create instances of their customized application models (the result of the modeling task using the graphical editor frontend). The retrieved model is checked against the (meta-)model to ensure valid sequences. During startup, the created instance is executed as a container on a cloud server. An automatic selection for an appropriate server is performed to avoid an overload of individual servers. In addition to administrative functions, the backend handles user authentification, i.e., the retrieval of user tokens for both backend and running instances. Last, it connects to a scalable database that stores data which is generated during the execution of instances. This ensures that instances can be resumed on failure, e.g., during server crashes.

4.3 Editor Frontend

The graphical editor frontend serves as a user interface for lecturers. It allows to customize their scenarios by composing different visual elements that represent the function blocks introduced by the (meta-)model. The main focus is the creation of an intuitive solution that lecturers with varying modeling abilities are able to use. For this reason, the concept uses ideas derived from the *User-Centered Design*

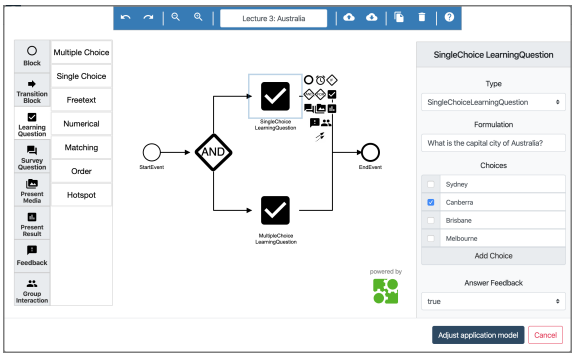

Figure 2: The graphical editor to create custom scenarios.

(UCD) approach that describes "design processes in which end-users influence how a design takes shape" (Abras et al., 2004). During an initial survey, the opinion of users for basic components of the editor (e.g., the position of the main menu or the strategy for inserting elements) was requested. Based on the results, a first conceptual prototype was developed. Open questions were discussed in user interviews, e.g., how the representation of elements should look like. The results retrieved from these interviews were combined in a final concept and implemented using *bpmn-js*[6] (a rendering toolkit and web modeler for BPMN 2.0[7]), as described by (Roszko, 2019) in more detail. A screenshot of an extended version of the editor is displayed in Figure 2.

4.4 Cloud Servers

The application infrastructure supports the integration of an arbitrary amount of cloud servers whereby the number of cloud servers increments as the resource demand increases. Each cloud server is able to run multiple instances of a runtime as a docker container, which, in turn, executes an application model specified by a lecturer. Each runtime allows a set of specified people (i.e., administrator, lecturer, specified students) to manage answers, finish currently active function blocks or retrieve results for a set of questions. Each runtime maintains a WebSocket connection to the backend as well as to each connected user (e.g., lecturer, student) to inform users in real-time about changes in the currently executed application models. While executing an application model, the runtime processes each function block as well as each transition individually and decides, based on the rules provided by the (meta-)model, which functions have to be executed and provided to the user.

[6]https://bpmn.io/toolkit/bpmn-js/ (accessed 3/19/20).

[7]Business Process Model and Notation – A modeling language for workflows.

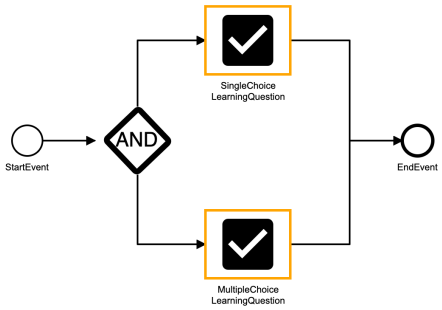

Figure 3: The display of the current scenario with its active function blocks.

4.5 User Interface

The user interface is the entry point for users to communicate with the system. According to the set user roles, different views are to be differentiated: The *lecturer* and *student* view.

The lecturer view displays a list of created instances with their respective application models. In addition, it allows to create new instances using the editor presented in subsection 4.3. Clicking on a specific instance will open a dashboard for managing it. The created application model is displayed on the top, and the currently active function blocks are highlighted in color, as depicted in Figure 3.

In the student view, these currently active function blocks are displayed. According to the set parameters, their respective functionality is adapted. E.g., a learning question with the parameter *answerFeedback* set to *true* will display immediate feedback after an answer has been given, as displayed in Figure 4. If the parameter is set to *false* (which is the default value of it), students do not receive feedback on the correctness of their answers until the lecturer evaluates the question. Setting these parameters properly enables the implementation of different teaching strategies.

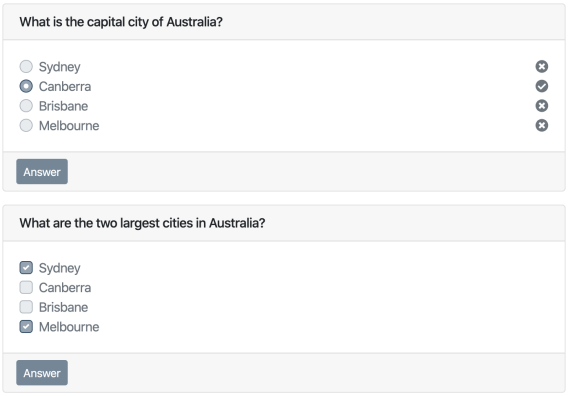

Figure 4: The answering of two questions with different *answerFeedback* parameter set.

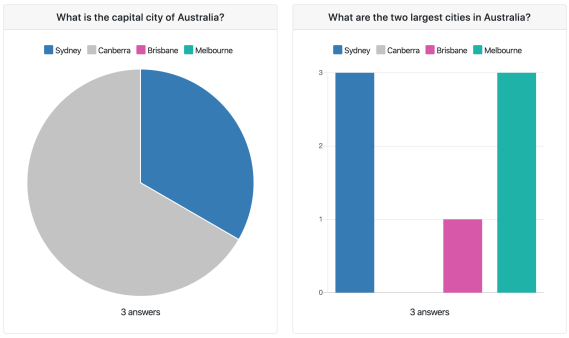

Figure 5: The result view of the questions displayed within Figure 4. Clicking on a specific container will open an extended view for evaluation.

In the lecturer view, the real-time evaluation of the currently active function blocks will be presented below the application model. For function blocks that allow the answering of students, e.g., learning or survey questions, the results are presented using charts, as depicted in Figure 5. Clicking on those overview items will open a modal for the presentation and discussion of these questions. For other function blocks, this view varies, e.g., for *closed feedback*, a modified student view with additional buttons for managing the discussions is displayed.

4.6 Options for the Lecturer

In order to configure the learning environment in a way that it supports the lecturer's personal teaching strategy, different options exist and are presented in the following.

First, lecturers are able to adapt each function block to their special needs by setting different parameters. E.g., lecturers that will discuss the results of a question with their students will disable the *answerFeedback*, while a lecturer using those questions as a self-test for students will not.

Second, lecturers can build sequences of function blocks and define conditions that allow for the realization of different learning paths. E.g., if the results of a question are not satisfying, another question could be displayed that will uncover the reason for the answers.

Third and as already motivated in the last example, function blocks can be linked to cooperate. E.g., the results of one or a set of questions can serve as the condition for different learning paths, or as the input of *result blocks* that allow to display results on the students' devices.

The fourth and final described option is the addition of novel collaborative functionality, motivated by the problem that peer or small group discussions are hard to accomplish in large lectures. We present an approach to move those discussions to an online learning environment. In addition to the support of a class-wide *open discussion* functionality, a group builder is introduced that allows the formation of groups based on different algorithms. Within these groups, different group interactions can be defined, e.g., group discussions or voting. An example of a group interaction is depicted in Figure 6 and visualizes the anonymous discussion between two students that answered a previous question differently.

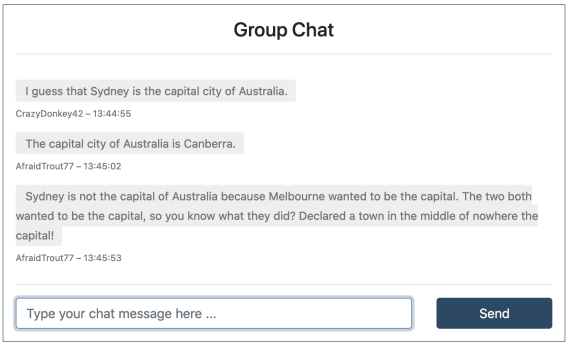

Figure 6: An example for a group discussion with participants having randomly generated pseudonyms.

5 EVALUATION

Since this work consolidates multiple different types of contributions based on several submissions, also different evaluation strategies apply for each of them. The following section summarizes those strategies.

5.1 Paper-based Evaluation of the (Meta-)model

As the (meta-)model was no fundamentally new idea but incorporated ideas of the according related work, the evaluation was primarily conducted to test if the weak points of prior approaches (i.e., very complicated modeling process, insufficient modeling capabilities) were still present. As the prototypical implementation of the system's backend and frontend as well as the graphical editor were still in development, a paper-based approach was chosen to simulate the system's capabilities. The evaluation was executed by 20 participants who predominantly agreed that the presented (meta-)model was easy to use and intuitive and that they therefore would use it in future teaching scenarios (Kubica et al., 2019b). During the evaluation, it was not feasible to test all functions which the (meta-)model provides. Therefore, a representative subset was chosen that was accepted very well by the participants of the study, although they did not use similar functions before.

5.2 Evaluation of the Prototypical Graphical Editor

In (Roszko, 2019), the prototype of the graphical editor was evaluated. The evaluation included five parts and was executed by 19 participants as follows: First, the participants were asked to fill in general information about their prior knowledge on graphical editors and the concept of workflows. Second, three different tasks were presented to the participants, which should be solved using the prototype. Each task checked specific abilities: While the participants were asked to insert, connect and parameterize function blocks in the first task, they should use abstract function blocks in a second more complex task and delete and adjust function blocks in the final task. Third, each component of the prototype was rated using a Likert scale. Fourth, a *System Usability Scale* (SUS; cf. (Brooke et al., 1996)) was determined before the fifth part concluded the evaluation with qualitative feedback from both positive and negative perspectives.

Although participants with different prior knowledge were part of this evaluation, we could not observe a significant difference in the results between those groups. Both groups of participants with prior knowledge and without were able to solve the tasks of the second part without major difficulties. This is also reflected in the SUS score, which ranges between 70 and 95. In particular, only 2 out of 19 participants rated a score under 85. The average score of 88 indicates a good to excellent usability. Moreover, the third part of our evaluation was able to verify our proposed concept. Only minor changes were necessary to implement the feedback of the participants, e.g., removing an alternative theme or additional buttons to duplicate or delete an element. Finally, the feedback retrieved from the qualitative questions was also added to the editor before it was integrated within the overall system.

5.3 Implementation in Realistic Scenarios

In the next step, we plan to extend these user studies by implementing our prototype in real scenarios, i.e., giving lecturers the prototype on hand and let them model and execute their customized scenarios. In addition to monitoring the system's behavior, interviews will be conducted to receive feedback from the lecturers. Our goal is to evaluate both the functionality and the opportunities provided by our prototype. Our created prototype is open to use for every lecturer.

6 CONCLUSIONS

In this paper, the prototype of an adaptable learning environment was presented. It gives lecturers the opportunity to target the system's functionality to their personal teaching strategy. They can adapt function blocks by different parameters, build conditional sequences of those function blocks, link function blocks to cooperate, and choose from novel collaborative functions as an addition to known *Audience Response*, or *Backchannel* functions. The results retrieved from user studies for the (meta-)model and the graphical editor were presented. Future applications within real-life scenarios were proposed to evaluate the overall concept and check the interplay of our individual components, namely the *backend server*, *cloud servers*, *graphical editor frontend* and *user interface*.

Due to design, our approach is targeted to lecturer-paced scenarios, i.e., the lecturer has to unlock the specific functionality before it can be used by the students. Our prototype does not investigate scenarios, in which the students can iterate the created application models by themselves. Nevertheless, an extension to such scenarios will be investigated in the future.

Having in mind that the presented prototype is in an early stage, improvements regarding the usability have to be investigated. In order to improve the user experience for lecturers even more, a proposal-based function is planned that will suggest the implementation of appropriate methods during the ongoing execution within a lecture.

ACKNOWLEDGEMENTS

This work is funded by the German Research Foundation (DFG) within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907). Special thanks to Tenshi Hara for his feedback and proofreading.

REFERENCES

Abras, C., Maloney-Krichmar, D., Preece, J., et al. (2004). User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, 37(4):445–456.

Brooke, J. et al. (1996). Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.

Bruff, D. (2019). *Intentional Tech: Principles to Guide the Use of Educational Technology in College Teaching*. West Virginia University Press.

Dragon, T., Mavrikis, M., McLaren, B. M., Harrer, A., Kynigos, C., Wegerif, R., and Yang, Y. (2013). Metafora: A web-based platform for learning to learn together in science and mathematics. *IEEE Transactions on Learning Technologies*, 6(3):197–207.

Hara, T. C. (2016). *Analyses on tech-enhanced and anonymous Peer Discussion as well as anonymous Control Facilities for tech-enhanced Learning*. PhD thesis, Technische Universität Dresden.

Kubica, T. (2019). Adaptable Collaborative Learning Environments. In *DCECTEL 2019 – Proceedings of the 14th EC-TEL Doctoral Consortium*.

Kubica, T., Hara, T., Braun, I., Kapp, F., and Schill, A. (2017). Guided selection of IT-based education tools. In *FIE 2017 – Proceedings of the 47h Frontiers in Education Conference*.

Kubica, T., Hara, T., Braun, I., Kapp, F., and Schill, A. (2019a). Choosing the appropriate Audience Response System in different Use Cases. In *ICETI 2019 – Proceedings of the 10th International Conference on Education, Training and Informatics*.

Kubica, T., Shmelkin, I., and Schill, A. (2019b). Towards a Development Methodology for adaptable collaborative Audience Response Systems. In *ITHET 2019 – Proceedings of the 18th International Conference on Information Technology Based Higher Education and Training*. IEEE.

Lieberman, H., Paternò, F., Klann, M., and Wulf, V. (2006). End-user development: An emerging paradigm. In *End user development*, pages 1–8. Springer.

Lingnau, A., Kuhn, M., Harrer, A., Hofmann, D., Fendrich, M., and Hoppe, H. U. (2003). Enriching traditional classroom scenarios by seamless integration of interactive media. In *Proceedings 3rd IEEE International Conference on Advanced Technologies*, pages 135–139. IEEE.

Meyer, M., Müller, T., and Niemann, A. (2018). Serious Lecture vs. Entertaining Game Show – Why we need a Combination for improving Teaching Performance and how Technology can help. In *EDULEARN 2018 – Proceedings of the 10th International Conference on Education and New Learning Technologies*.

Nikou, S. A. and Economides, A. A. (2018). Mobile-based assessment: A literature review of publications in major referred journals from 2009 to 2018. *Computers & Education*, 125:101–119.

Roszko, L. (2019). Entwicklung eines graphischen Editors zur Erstellung von beliebigen Lernszenarien in Audience Response Systemen.

Schön, D. (2016). *Customizable Teaching on Mobile Devices in Higher Education*. PhD thesis, Universität Mannheim.

Sendall, S. and Kozaczynski, W. (2003). Model transformation: The heart and soul of model-driven software development. *IEEE software*, 20(5):42–45.

Stahl, T., Völter, M., Efftinge, S., and Haase, A. (2007). Modellgetriebene Softwareentwicklung: Techniken. *Engineering, Management*, 2:64–71.