



# Generalized Automatic Item Generation for Graphical Conceptual Modeling Tasks

Paul Christ<sup>1</sup><sup>a</sup>, Torsten Munkelt<sup>2</sup> and Jörg M. Haake<sup>1</sup><sup>b</sup>

<sup>1</sup>Department of Cooperative Systems, Distance University Hagen, Universitätsstraße 11, Hagen, Germany

<sup>2</sup>Faculty of Informatics and Mathematics, HTWD, Dresden, Germany

**Keywords:** AIG, Automatic Item Generation, Conceptual Modeling, Competency-Based Learning, E-Assessment, E-Learning, Bloom's Taxonomy, Business Process Modeling, Graph-Rewriting, Generative AI, Large Language Models.

**Abstract:** Graphical conceptual modeling is an important competency in various disciplines. Its mastery requires self-practice with tasks that address different cognitive processing dimensions. A large number of such tasks is needed to accommodate a large number of students with varying needs, and cannot be produced manually. Current automatic production methods such as Automatic Item Generation (AIG) either lack scalability or fail to address higher cognitive processing dimensions. To solve these problems, a generalized AIG process is proposed. Step 1 requires the creation of an item specification, which consists of a task instruction, a learner input, an expected learner output and a response format. Step 2 requires the definition of a generator for the controlled generation of items via a configurable generator composition. A case study shows that the approach can be used to generate graphical conceptual modeling tasks addressing the cognitive process dimensions *Analyze and Create*.

## 1 INTRODUCTION

Conceptual modeling is the process of abstracting a model from a real or proposed system (Robinson, 2008). The outcome of conceptual modeling is an abstraction or graphical representation of the modelled system in the chosen modeling language (He et al., 2007; Delcambre et al., 2018; Guarino et al., 2019). The modality of the conceptual model depends on the type of modeling language, e.g. textual or graphical.


As laid out by Striwe et al., conceptual modeling is a core component of the curriculum of Business Informatics and neighbouring disciplines (Striwe et al., 2021) (Soyka et al., 2022). Multiple organizations, such as the Association for Computing Machinery (ACM) and the Gesellschaft für Informatik (GI) recommend including conceptual modeling into the curricula of various IT-disciplines (ACM, 2021; GI, 2017).


In IT-related disciplines, graphical conceptual modeling may be applied to construct diagrams for designing databases and software or business pro-

cess models to abstract a planned or existing area of an organization. Beyond that, conceptual modeling is broadly applicable in different disciplines, e.g. mathematical modeling (Dunn and Marshman, 2019), molecular modeling (Taly et al., 2019) or modeling schemata in music theory (Neuwirth et al., 2023).

To teach, learn and assess graphical conceptual modeling, summative and formative assessments are required (Meike and Constantin, 2023). An assessment is a set of one or many assessment-items (items). An item refers to a statement, question, exercise, or task for which the test taker is to select or construct a response, or perform a task (American Educational Research Association and American Psychological Association, 2014). Assessments may address different cognitive processing dimensions. To model these cognitive processing dimensions, we utilize the revised Bloom's taxonomy (Anderson et al., 2001), due to its popularity for structuring modeling tasks (Soyka et al., 2022; Bork, 2019; Bogdanova and Snoeck, 2017).

Striwe et al. further argue, that there is a large unmet demand for a shift from lecturer-centered to student-centered teaching. This shift requires strengthening problem-solving and self-

<sup>a</sup> <https://orcid.org/0000-0002-6096-7403>

<sup>b</sup> <https://orcid.org/0000-0001-9720-3100>

learning skills of students. This, in turn, requires the creation of competence-oriented assessments, particularly assessments that address higher cognitive dimensions. Using traditional multiple choice (MC) questions or other closed task formats make it challenging to address these higher cognitive dimensions. Conceptual modeling is especially affected by this limitation, as it is the practical application of modeling methods and tools that is of importance (Striwe et al., 2021).

The construction of conceptual modeling tasks is demanding. This is especially true for conceptual modeling tasks that address higher cognitive processing dimensions, as they require a meaningful problem description, instructions and at least a sample solution model. Thus, only a small amount of tasks is available to students, which limits their opportunity to exercise their abilities and therefore hinders their learning process.

To solve the problem of insufficient amounts of conceptual modeling tasks, addressing higher cognitive processing dimensions, we identified 3 approaches:

1. Traditional item development (Rudolph et al., 2019), which represents the manual effort of creating graphical conceptual modeling tasks. This approach does not scale to a large amount of tasks to be created, since lecturing staff is a limited resource.
2. Automatic item generation (AIG) (Gierl et al., 2021), which utilizes human-made task-templates and computer technology, to create many tasks according to that template. This approach generally only applies to closed task formats, that make it challenging to address higher cognitive processing dimensions.
3. Domain-specific methods (Schüler and Alpers, 2024; Yirik et al., 2021; Ghosh and Bashir, 2018), which commonly utilize transformation rules, to transform a (often textual) representation into a valid model of a formal modeling language. This approach requires lecturing staff to provide separate input for each task to be generated, and thus does not scale.

Because all three approaches have serious disadvantages, it is an open research question how to provide a sufficient amount of graphical conceptual modeling tasks for each cognitive processing dimension of the revised Bloom's taxonomy.

Chapter 2 provides a deeper analysis of the research question and defines requirements for potential solutions. Chapter 3 gives an overview of the current state of the art and its shortcomings with respect to

the defined requirements. Chapter 4 presents the proposed solution to the defined question. Chapter 5 validates the proposed solution by utilizing a case study to show that the requirements are met. Chapter 6 provides a summary of the paper and an overview of open questions and future research.

## 2 ANALYSIS

To answer how to provide a sufficient amount of graphical conceptual modeling tasks for each cognitive processing dimension, one must first define 1.) what graphical conceptual modeling tasks are, 2.) how a specific graphical conceptual modeling task addresses a specific cognitive dimension and 3.) what a sufficient amount of each graphical conceptual modeling task is.

1.) and 2.) are best answered in conjunction, by defining a mapping of cognitive process dimensions to potential types of graphical conceptual modeling tasks. We utilize the work of (Bork, 2019), which maps the cognitive processing dimensions to potential conceptual modeling tasks, and specify a concrete conceptual modeling task with an item specification for each dimension. An item specification, akin to Gierl's definition of a cognitive model, refers to the concepts, assumptions and logic to create and the assumptions about how examinees are expected to solve a content-specific task (Gierl et al., 2021).

Table 1 shows the mapping of cognitive process dimension to each item specification. The cognitive process dimensions refer to the 6 dimensions of the revised Bloom's taxonomy. A placeholder is signified by angle brackets. An item specification holds the information required to perform the task, that is described by it. An item specification consists of an instruction template, an input placeholder, an output placeholder and a response format. The instruction template specifies what task the learner is expected to perform, the input placeholder contains the content that is required to perform the task, the output placeholder contains an expected solution, and the response format governs how the learner's response is retrieved.

The placeholders must be filled with a concrete instance of the placeholder type. This can be done manually or via an automated production process. Once the placeholders are filled, an item-instance is created.

The first row in Table 1 shows an example item specification of a task that is specified for the dimension *Remember*. In this case, it is a selected-response task, e.g. MC. The instruction guides the learner to select the correct name of an element of a modeling

Table 1: Cognitive Process Dimension Mapped to Item Specification of Conceptual Modeling Tasks.

Dimension	Instruction Template	Item Specification		
		Input Placeholder	Output Placeholder	Response Format
<b>Remember</b>	Select the correct name of the shown syntactical element in the notation of ⟨MODELING LANGUAGE⟩.	⟨SYNTACTICAL ELEMENT⟩	⟨SOLUTION⟩ ⟨DISTRACTOR <sub>n</sub> ⟩	Selected-Response
<b>Understand</b>	Select the correct semantic concept of the shown construct.	⟨MODEL CONSTRUCT⟩	⟨SOLUTION⟩ ⟨DISTRACTOR <sub>n</sub> ⟩	Selected-Response
<b>Apply</b>	Use ⟨SET OF OPERATIONS⟩ to transform ⟨PROCESS PART⟩ into ⟨TRANSFORMED PROCESS PART⟩.	⟨MODEL⟩	⟨TRANSFORMED MODEL⟩	Constructed-Response
<b>Analyze</b>	Find and mark syntactical errors in the shown model.	⟨ERRONEOUS MODEL⟩	⟨MARKED MODEL⟩	Constructed-Response
<b>Evaluate</b>	Find and mark the inconsistencies between the textual and graphical model representation.	⟨MODEL WITH LABELS⟩ ⟨TEXTUAL MODEL DESCRIPTION⟩	⟨MARKED MODEL⟩ ⟨MARKED MODEL DESCRIPTION⟩	Constructed-Response
<b>Create</b>	Create a model from the given textual description in the notation of ⟨MODELING LANGUAGE⟩.	⟨TEXTUAL MODEL DESCRIPTION⟩	⟨MODEL WITH LABELS⟩	Constructed-Response

language, that has yet to be specified. The learner is presented an input, more precisely a graphical syntactical element of a modeling language. The learner is then expected to select one of the presented options, of which one is the solution and the remainder being distractors.

Other tasks include the application of model transformations on a presented model, finding and marking errors in a erroneous model, finding inconsistencies between a textual and a graphical representation of a model and the manual creation of a model, given a textual description of the model.

The generation of the aforementioned conceptual modeling tasks require the generation of viable instances of the specified input and output placeholder, that fit the instruction (Requirement R1).

To answer 3.), what determines the sufficient total number of items  $A$ , we propose the following simplified model:

Let  $I_c$  be the number of items for a cognitive processing dimension  $c$ , and let  $I_C = \sum_{c=1}^{|C|} I_c$  be the number of items for all cognitive processing dimensions  $C$ , that we assume necessary for students exercising and the assessment of students. Let  $o_m$  be the number of learning objectives of a module  $m$ . The total num-

ber  $O_M$  of learning objectives for all modules  $M$  is then calculated as  $O_M = \sum_{m=1}^{|M|} o_m$ . Let  $T$  be the number of teachers. As teachers may prefer the formulation and style of the items to be conforming to the rest of their learning material, they each might need separate items. Thus, we arrive at

$$A = T * O_M * I_C \tag{1}$$

for computing the number of items needed.  $I_C$  may be broken down further as the sum of the number of items required in a set  $S$  of different scenarios  $I_C = \sum_{c=1}^{|C|} \sum_{s=1}^{|S|} I_{cs}$ . Different scenarios may include (I) the number of items required in the coursework  $I_{cw}$ , (II) the number of items a learner requires for individual practice  $I_{cp}$  or (III) the number of items for exams  $I_{ce}$ . In the case of individual practice, it may be required to provide a number of personalized items for a number of learners  $L$ , resulting in  $I_{cp_L} = \sum_{l=1}^L I_{cl}$ . In the case of exams, it may be required to provide a.) new items for every exam in order to avoid memorization and b.) different items per examinee in order to minimize cheating, which introduces the factor of the number of examinees  $X$  and breaks  $I_{ce}$  down further into the sum of required items per exam  $I_{ce} = \sum_{e=1}^E I_{ce} * X$ .

In the example of a university course teaching business process modeling, which deals with 2 distinct conceptual modeling languages, no item personalization, a cohort of 50 students, 2 potential exam offerings with an initial failure quote of 20% and a respective participation quote of 100%, we arrive at  $T = 1$ ,  $M = 1$ ,  $O = 2$ ,  $I_{CW} = 3$ ,  $I_{CP} = 10$  and  $I_{CE} = \sum_{c=1}^{|C|=3} 2 * 50 + 2 * 10 = 460$ , yielding a required number of items  $A = 1 * 2 * 460 = 920$ .

The ATOMIC-formula gives the reader a way to determine an appropriate number for their use case.

To potentially cover all scenarios described above, a production method for items must address each factor of the ATOMIC-formula individually and in a scalable manner (Requirement R2).

### 3 STATE OF THE ART

#### 3.1 Traditional Item Development

Traditional item development relies on a method in which a subject matter expert (SME) creates items individually (Gierl et al., 2021). Under the best condition, traditional item development is an iterative process where highly trained groups of SMEs use their experience and expertise to produce new items. Then, after these new items are created, they are edited, reviewed, and revised by another group of highly trained SMEs until they meet the appropriate standard of quality (Haladyna, 2015).

This approach has two major limitations: 1.) Item development is time-consuming and expensive because it relies on the item as the unit of analysis (Stark et al., 2006). Each item in the process is unique and therefore, each item must be individually written, and ideally, edited, reviewed, and revised (Gierl et al., 2021). 2.) The traditional approach to item development is challenging to scale efficiently and economically. When one item is required, one item is written by the SME because each item is unique. Hence, large numbers of SMEs who can write unique items are needed to scale the process. (Gierl et al., 2021).

As a result, traditional item development does not meet requirement R2 and can not be considered a feasible method to generate a sufficient amount of conceptual modeling tasks.

#### 3.2 Automatic Item Generation

AIG is the process of using models to generate items using computer technology (Gierl et al., 2021). Gierl et al. describe AIG as the three-step process for generating items depicted in figure 1 (Gierl and Lai, 2016).

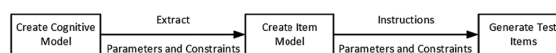


Figure 1: Three-step process of AIG (Gierl et al., 2021).

In step 1, the content for item generation is created in the form of a cognitive model which highlights the knowledge, skills and problem-solving processes required to solve the generated task (Gierl et al., 2012). In step 2, an item model is created, that specifies which parts and which content in the task can be manipulated to create new items (Laduca et al., 1986). The parts include the stem, the options and the auxiliary information to generate selected-response items (e.g. MC) (Gierl et al., 2021). In step 3, all possible combinations of the created content are placed into the item model, for which different computer-based assembly systems are used (Kucharski et al., 2023; Gierl et al., 2008).

One of the main limitations of existing AIG-methods is the simplicity of generated questions, as most generated questions use closed response-formats (Kurdi et al., 2020; Falcão et al., 2023), which make it challenging to target higher cognitive dimensions (beyond *remember* and *understand*) (Kurdi et al., 2020). While closed target formats may address the higher processing dimensions, *apply* and *analyze*, even proponents of these formats agree, that they can't address the highest processing dimensions *evaluate* and *create* (Haataja et al., 2023).

Thus, current AIG-methods fail to meet requirement R1 and therefore are not a feasible method to produce conceptual modeling tasks that address the highest cognitive processing dimensions.

#### 3.3 Subject Specific Methods

To overcome the limitations of traditional item development and AIG-methods, we consider the utilization of subject-specific construction methods for producing the required inputs and outputs for conceptual modeling tasks described in section 2. Different subject matter domains provide methods for transforming a potentially fuzzy model representation, such as textual descriptions in natural language into potentially many valid representations in the respective formal modeling language. Examples include the transformation of business process descriptions into a graphical modeling language (Schüler and Alpers, 2024), the generation of constitutional isomer chemical spaces from molecular structural formula (Yirik et al., 2021) or the generation of entity-relationship diagrams from a set of textual requirements in natural language. (Ghosh and Bashar, 2018).

Two major limitation of utilizing subject-specific

production methods are 1.) their lack of generalization across subject matter domains and 2.) the need for lecturing staff to provide separate input, in order to create item variants.

Thus, subject-specific methods fail to meet requirement R2 and are therefore not a feasible method to produce sufficient amounts of conceptual modeling tasks across different subject matter domains.

### 3.4 Delta to Existing Methods for Generating Graphical Conceptual Modeling Tasks

Traditional item development and subject-specific methods fulfill requirement R1 but fail to address requirement R2. Conversely, current AIG-methods fulfill requirement R2 but fail to address requirement R1 fully.

Thus, a solution to the posed research question of how to generate a sufficient amount of graphical conceptual modeling tasks, for each cognitive dimension of the revised Bloom's taxonomy, must fulfill the requirements R1 and R2 simultaneously.

## 4 APPROACH

Our proposed solution for the AIG for graphical conceptual modeling tasks consists of a two-step process. The first requires the item-developer to create an item specification. As shown in table 1, an item specification consists of an instruction template, an input placeholder, an output placeholder and a response format. The item specification specifies what task the learner is expected to perform via the instruction template, what content is required to perform the task via the input placeholder, what the expected solution is via the output placeholder, and how the learner's response is selected via the response format.

The second step requires the item-developer to specify a generator, that generates instances for the input and output placeholder of the item specification specified in the first step. A generator is a function  $G(I) = O$ , that takes an input  $I$  and produces an output  $O$ . A generator may consist of multiple other generators.

The proposed two-step process is a generalization of AIG as described in subsection 3.2, which will from here on be called generalized AIG (gAIG). This generalization uses AIG for generating items addressing the first two cognitive dimensions (see item specification shown in table 2).

To generate a generic single choice question, a

Table 2: Item specification for a generic single choice question.

<b>Instruction Template</b>	Select the correct answer for the given statement.
<b>Input Placeholder</b>	$\langle \text{STEM} \rangle \langle \text{OPTIONS} \rangle$
<b>Output Placeholder</b>	$\langle \text{CORRECT OPTION} \rangle$
<b>Response Format</b>	Selected Response

generator must generate a  $\langle \text{STEM} \rangle$  that formulates a question, corresponding  $\langle \text{OPTIONS} \rangle$  which consist of multiple distractors and a  $\langle \text{CORRECT OPTION} \rangle$ .

As described by Gierl et al., AIG generates such  $\langle \text{STEM} \rangle$ , which contains the content or question the examinee is required to answer and the  $\langle \text{OPTIONS} \rangle$ , a set of alternative answers with one correct option and one or more incorrect options (Gierl et al., 2021). Thus, AIG can be applied as a generator in the second step of the gAIG-process, that produces the necessary input and output placeholders of the item specification described in table 2.

But, as already discussed in section 2, AIG is not able to produce the specified graphical conceptual modeling tasks, that address cognitive levels above the cognitive processing dimension of *Apply*, as they require different input and output placeholders. As a consequence, different generators are needed.

### 4.1 Generators for Graphical Conceptual Modeling Tasks Addressing Higher Cognitive Dimensions

In the following we introduce 3 generators, that are able to produce instances of the required input and output placeholders, using the following 3 technologies: 1.) a graph-rewriting system (GRS), 2.) a large language model (LLM) and 3.) a text-template engine (TTE).

These technologies were chosen because: 1.) they can produce the required output modalities, 2.) they are subject matter domain independent, and 3.) their generation behavior is configurable.

Figure 2 depicts a simplified model of the generator-elements. The abstract class GENERATORELEMENT has a generate-method which takes a generic input  $I$  and produces a generic output  $O$ . The generate-method of the GRS receives a GRSINPUT and produces a GRAPH as output, which consists of an array of nodes, an array of edges and an array of subgraphs. The GRSINPUT consists of:

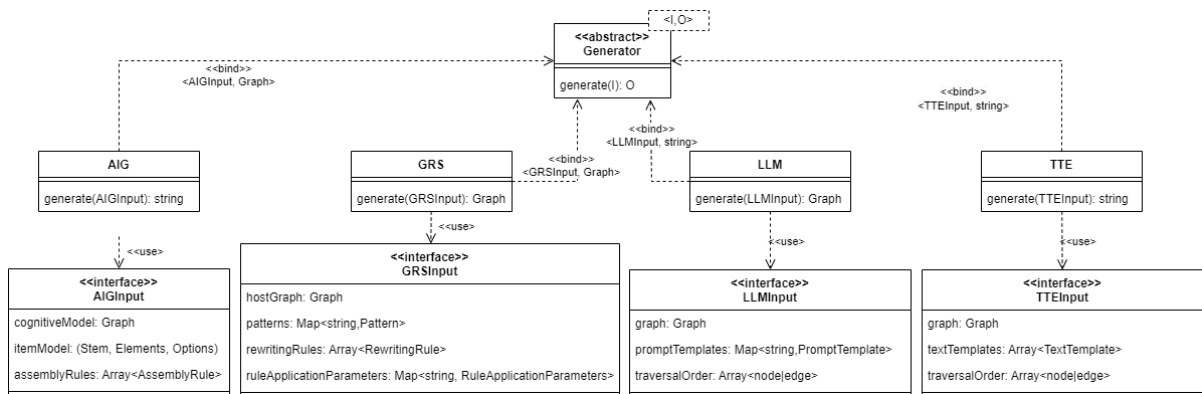


Figure 2: Simplified class diagram of generators.

- A host graph  $g$ , on which all rewriting operations are performed,
- a set of patterns, where each element consists of a name and a pattern. Upon calling, a pattern creates a randomized pattern-instance that conforms to the pattern-structure, a set of input parameters and potential constraints.
- An array of rewriting rules, and each rewriting rule  $r(g, L, R)$  contains a pattern graph  $L$  and a replacement graph  $R$ , and an instance of  $L$  is to be replaced with an instance of  $R$  in  $g$ . A rule is only applied if an instance of  $L$  can be found in  $g$ .
- A set of parameters, that guide the application of the rewriting rules:
  - how often each rule must be applied,
  - an execution order of the rules, and
  - filter options for the selection of potential instances of  $L$  found in  $g$ , from which a random instance is then selected for rewriting.

The generate-method of the LLM receives a LLMINPUT and produces a GRAPH, where each node, edge and subgraph has been labeled. The LLMINPUT consists of:

- an input graph, that is to be labelled,
- a set of prompt-templates, that are associated to patterns and are used to elicit the generation of labels, and
- an order in which to traverse the elements of the input graph.

The generate-method of the TTE receives a TTEINPUT and produces a string. The TTEINPUT consists of:

- an input graph, that may contain labels and is to be transformed into text,

- a set of text-templates, that the TTE uses to transform the associated pattern instances and the contained subgraph-, node-, and edge-labels into text, and
- an order in which to traverse the elements of the input graph.

In order to generate instances of some of the required input and output placeholder types, generators need to be composed into larger generators. E.g. generating labels for a graph, requires the graph to be generated first. To assemble the full item according to the item specification, the generated outputs must be mapped to the placeholders of the item specification. These additional requirements are fulfilled by another component called generator-orchestration service.

### 4.2 Generator-Orchestration Service

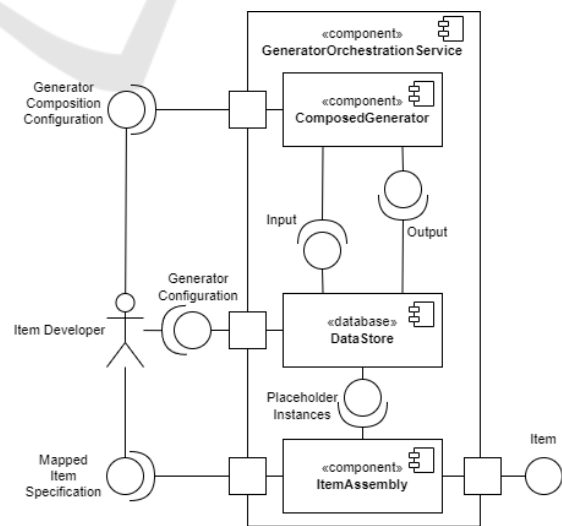


Figure 3: Component diagram of the generator-orchestration service.

Figure 3 gives an overview of the generator-orchestration service. In order to compose generators, the nested generators must follow an execution order, the associated inputs have to be specified, and the intermediate outputs need to be persisted. The execution order and the associated inputs of generators are referred to as *generator composition configuration* and must be provided by the item developer. Intermediate outputs of nested generators and composed generators are persisted in a store for future reference. The input for a nested or composed generator can be provided by the item developer or previously generated outputs of other generators. To assemble the item, the item developer must provide an item specification and map the therein contained placeholders to the data in the data store.

The modularity of the approach allows for reusing nested and composed generators. Furthermore, it enables the extensibility of the approach by adding more types of generators. This increases the potential for generalization across different subject matters and multiple task-types beyond graphical conceptual modeling tasks. Manual efforts for analyzing the item quality can be drastically reduced since the quality of the process of the item generation must only be checked once, as was already pointed out by Stark et al. (Stark et al., 2006), instead of checking the quality of each generated item.

## 5 CASE STUDY

To verify that the solution proposed in section 4 meets both requirements defined in section 2, we present a case study that shows the production of multiple tasks for the subject matter domain of Business Process Modeling. Due to its simple and concise syntax, we utilize Event-Driven Process Chains (EPC) (Keller et al., 1992) as a process modeling language.

It is crucial to understand the notation of EPCs to create generators that yield syntactically valid EPCs. The following provides a short overview of the basic notation of EPCs that is required to follow along.

The EPC notation consists of the following symbols: events, functions, process route signs, logical connectors (AND, OR, XOR) and connection arrows. Roles, responsibilities and data are omitted for the sake of simplicity.

Each process starts and ends with an event. Events and functions must alternate. Symbols must be connected by directed lines. Splitting connectors must have one incoming and at least two outgoing arrows. Conversely, merging connectors must have at least two incoming and one outgoing arrow. Splitting OR-

and XOR-operators must not follow an event.

### 5.1 Specification of a GRS to Generate EPCs

To generate graphical conceptual modeling tasks, we utilize an instance of the GRS-generator as a foundational step for all tasks. In the following, we specify the required inputs for the GRS-generator-instance.

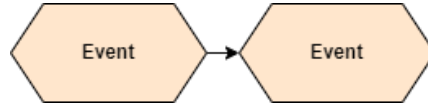


Figure 4: Host graph for the production of EPCs.

Given the above notation, we chose the host graph depicted in figure 4, which consists of two joined events. Note, that the host graph is not yet compliant with the notation of EPCs.

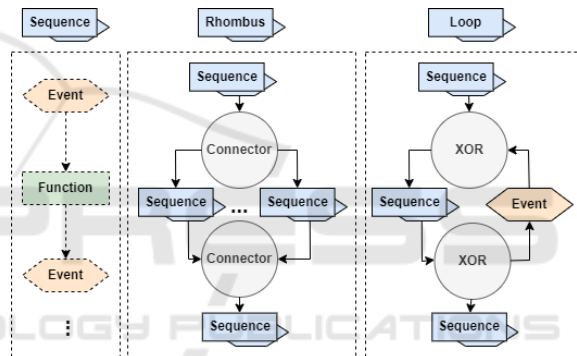


Figure 5: Three possible patterns to create EPCs.

Then, we formulate the patterns depicted in figure 5. Each pattern is depicted as a process route sign. The sequence-pattern includes a variable amount of events and functions, that are alternating and connected by arrows. Its construction function can be described as  $P_s(E_p, E_c, N)$ , where  $E_p$  is a function to determine the required element-type to start the sequence with,  $E_c$  is a function to determine the required element-type to end the sequence with and  $N$  is the number of nodes to generate between the starting and ending element of the sequence.  $E_p$  is described in algorithm 1 and receives the parent node of where the pattern-instance is to be inserted and traverses the chain of parent nodes, until it discovers an element of either type "Event" or "Function" and then returns "Function" or "Event".  $E_c$  functions similarly, but additionally receives the result of  $E_p$  and traverses the chain of child elements instead. The parameter  $N$  specifies the overall length of the EPC to be generated.

```

Data: E: Node
Result: T: string
while E.type ≠ "Event" or E.type ≠
    "Function" do
    | E = E.parent;
end
if E.type = "Event" then
    | return "Function"
else
    | return "Event"
end
    
```

Algorithm 1: Function  $E_P$ , that finds the element-type of the parent-element.

The rhombus-pattern includes a variable number of branches between the connectors and a variable connector-type. Its construction function can be described as  $P_r(N, T)$ , where  $N$  is the number of branches to be created and  $T$  is the connector-type.

The loop-pattern consists of one forward-branch and one backward-branch with exactly one event between the connectors, and the connector-type is fixed to a "XOR"-type.

With those patterns, we formulate a set of rewriting-rules to be performed on the host graph. A rule consists of the pattern graph  $L$  and the replacement graph  $R$ . An instance of  $L$  in the host graph is randomly chosen from a list of found matches. An instance of  $L$  is replaced with an instance of  $R$  in the host graph. The rewrite graph is the resulting graph, after the rewrite rule has been applied to the host graph, and the rewrite graph becomes the new host graph.

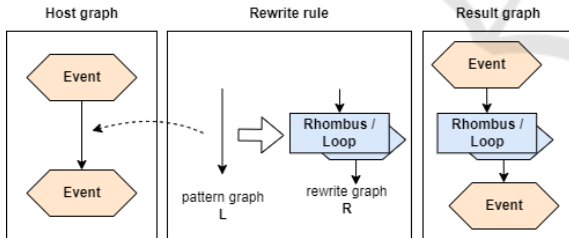


Figure 6: Rewrite rule 1.

Rule 1, depicted in figure 6, shows how complex patterns are introduced into the host graph. The rule selects a pattern graph in the shape of single connection arrow and replaces it with either a rhombus- or loop-pattern and an incoming and an outgoing edge. This rule assumes a random selection of the rhombus- or the loop-pattern for simplification purposes.

Rule 2, depicted in figure 7, shows how complex patterns are instantiated. In the given example, a randomly selected rhombus-pattern is replaced by an instance of a rhombus-pattern. For simplification purposes it is assumed, that a random pattern-instance is

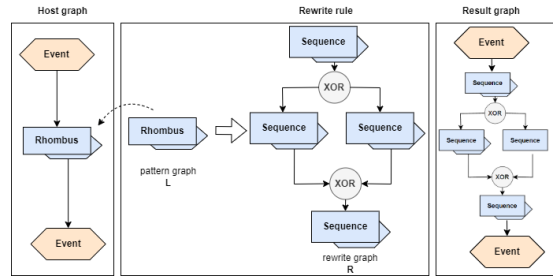


Figure 7: Rewrite rule 2.

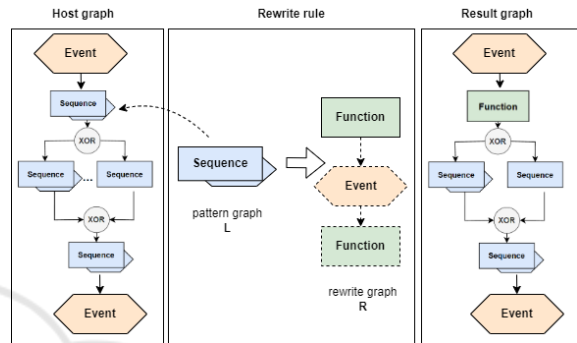


Figure 8: Rewrite rule 3.

instantiated.

Rule 3, depicted in figure 8, shows how sequences are instantiated. In the given example, a randomly selected sequence-pattern is replaced by an instance of a sequence-pattern. The sequence with the least amount of nodes is determined by the function  $E_C$  and  $E_P$  described in algorithm 1 when calling the construction function  $P$ . For simplification purposes, it is assumed that the number of additional nodes  $N$  is 0.

Applying the patterns and rules in a GRS randomly would yield a large amount of vastly distinct EPCs. To achieve a more controlled generation, guidance-parameters may be utilized, such as the exact number a rule must be applied, an execution order of the rules and more restrictive filter operations for the pattern graph selection, such as the selection of nested pattern graphs.

It is important to note, that this is merely one possible formulation of a GRSInput to yield syntactically valid EPCs and that there exist many more. Moreover, the presented patterns and rules do not cover the entire space of possible and syntactically correct EPC.

In the following, we demonstrate the proposed gAIG process. As discussed in section 2, AIG is able to generate tasks for the cognitive processing dimensions *Remember* and *Understand*. As described in section 4, AIG is a generator that can be directly applied in our proposed two-step process. As the approach for AIG has been described many times by



prior works, we only showcase the solution approach for newly introduced generators. We do so with the example of the task types associated with the cognitive process dimensions *Analyze* and *Create*.

### 5.2 Generation of Graphical Conceptual Modeling Tasks for EPCs that Address the Cognitive Process Dimension *Analyze*

Mapping the task for the cognitive process dimension of *Analyze* described in table 1 to the concrete modeling language of EPC requires an unlabeled EPC that includes identifiable error patterns.

Thus, the required generator is a GRS, as described in section 5.1 above, with the addition of rules that introduce error patterns. The addition of error patterns may be performed in a new GRS or after the rewriting rules to construct a valid EPC have been applied. For the sake of simplicity, the following example assumes the host graph to be an already valid EPC.

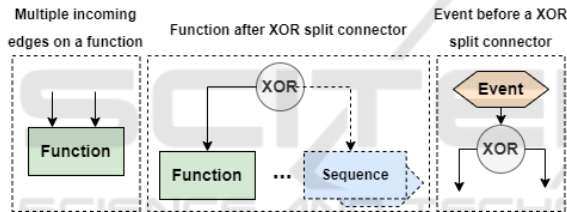


Figure 9: Common error patterns in modeled EPCs amongst learners.

We chose three commonly occurring syntactical errors made by students (Szcześniak, 2011), depicted in figure 9, but arbitrarily many error patterns can be introduced. The error-patterns shown include 1.) multiple incoming edges to a function, 2.) the use of a function or a process route sign (as the hidden subprocess has to start with a function) after a XOR split connector and 3.) the use of an event before a XOR split connector.

To introduce these patterns into the graph, another set of rewriting rules is required.

Error rule 1, depicted in figure 10, shows how error 1.) is introduced into a valid EPC. As this error pattern has no variability and thus can not add unexpected errors elsewhere, no additional conditions are needed. The construction function of this pattern is thus simply its identity function.

Error rule 2, depicted in figure 11, shows how error 2.) is introduced into a valid EPC. To avoid further unintended errors, the construction function of this error pattern needs to ensure that the following se-

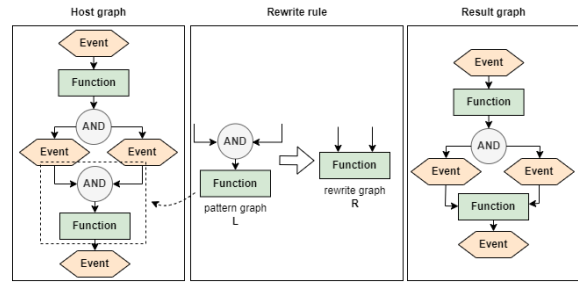


Figure 10: Error rule 1.

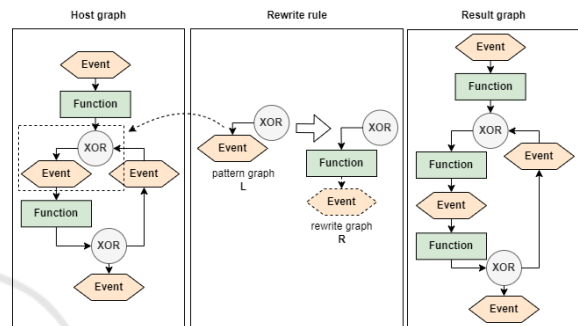


Figure 11: Error rule 2.

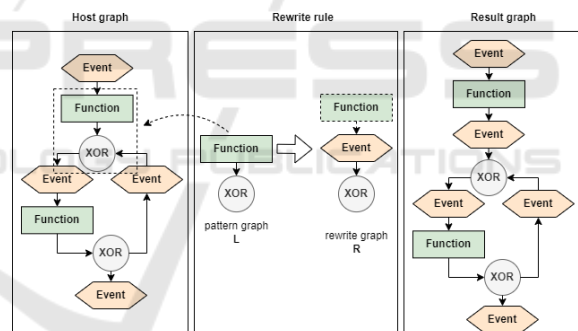


Figure 12: Error rule 3.

quences remain syntactically correct. This is achieved by utilizing the same function  $E_C$ , which is used to ensure the syntactical validity of the EPC when instantiating a sequence-pattern. Thus, the construction function of this pattern is  $P_{e2}(E_C)$ .

Error rule 3, depicted in figure 11, shows how error 3.) is introduced into a valid EPC. Similarly to error rule 2, the construction function of this error pattern needs to ensure the syntactical validity outside the error pattern. This is achieved by utilizing the function  $E_P$  to ensure the validity for the previous sequences. Thus, the construction function of this pattern is  $P_{e3}(E_P)$ .

Note, that applying multiple error patterns simultaneously requires additional constraints to avoid unexpected errors. A simple way to avoid unexpected

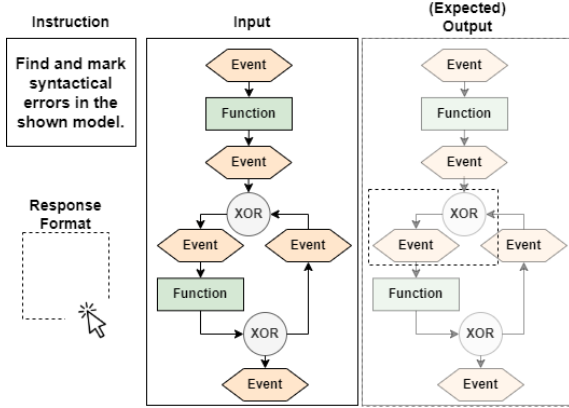


Figure 13: Analyze-task for graphical conceptual modeling.

errors would be to exclude instances of error-patterns created by previous rewrite rules in the host graph.

To finally assemble the complete item, the generated outputs must be associated with the item specification. The generated erroneous model is assigned to the input placeholder (ERRONEOUS MODEL) and the same generated erroneous model with visualised subgraphs for the error-pattern (i.e. the solution) is assigned to the output placeholder (MARKED MODEL).

### 5.3 Generation of Graphical Conceptual Modeling Tasks for EPCs that Address the Cognitive Process Dimension Create

Mapping the task for the cognitive process dimension of *Create* described in table 1 to the concrete modeling language of EPC requires a textual description of a labeled EPC.

To produce these outputs, a composed generator is required, which utilizes the GRS, as described in section 5.1, a LLM and a TTE. For the sake of simplicity and to avoid redundancy, the following example assumes an already generated EPC by the GRS as a starting point.

The LLM then traverses the GRS and applies prompt-templates to recognized pattern-instances in the EPC. The starting point is always the first event-node. For the sake of simplicity, a predetermined path is assumed.

Figure 14 shows the first step of the traversal of an unlabeled EPC and the application of a prompt-template that was matched to the traversed pattern. The start-event is mapped to a corresponding prompt-template. The prompt-template is structured into four sentences. The first sentence contains a placeholder and sets the context for the LLM. As it is the first

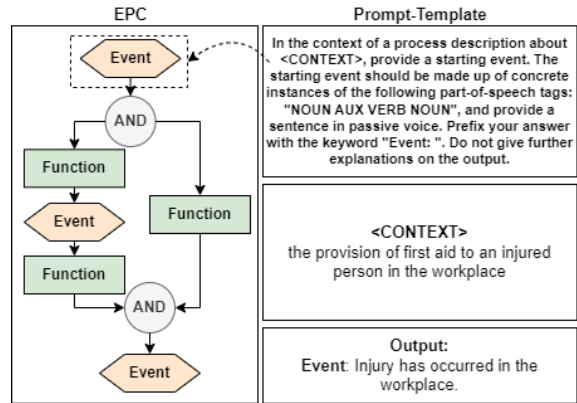


Figure 14: Step 1 of labeling an EPC with prompt-templates and a LLM.

node to be labeled, the context is provided as a start parameter. The second sentence specifies the shape of the desired textual output. The third sentence specifies to generate a prefix before the output, to make it machine-readable. The fourth sentence specifies to limit the generated output, to reduce potential noise.

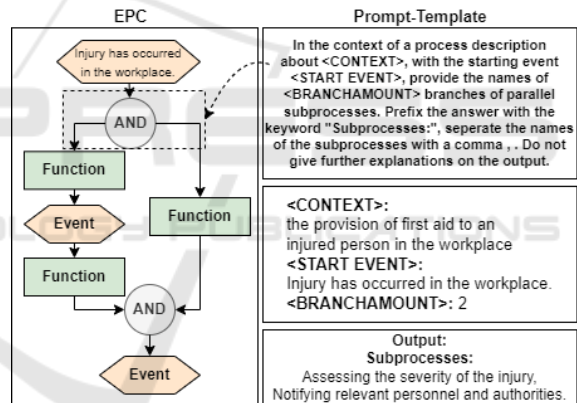


Figure 15: Step 2 of labeling an EPC with prompt-templates and a LLM.

Figure 15 shows the second step of the traversal and labeling. In this step, the branching AND-operator is mapped to a corresponding prompt-template, which describes the start of parallel subprocesses. The placeholders of the prompt-template are filled with the existing context, the generated start-event and the extracted branch-amount of the currently viewed subgraph. The generated output names the two subprocesses for later reference.

Figure 16 shows the third step of the traversal and labeling. In this step, the prompt-template is mapped to a sequence that is nested inside a parallel process. The placeholders are then filled with the generated name of the subprocess, the type of subprocess, and the amount and type of elements in the sequence.

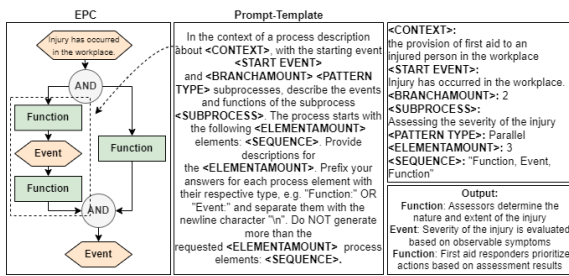


Figure 16: Step 3 of labeling an EPC with prompt-templates and a LLM.

To avoid repetition, the remainder of the labeling process is not explicitly demonstrated. This approach can be extended to match any graph pattern to a defined prompt-template. The context information is built-up incrementally by the generator and reused to generate output that maintains the semantic congruity of the entire process model.

In a similar fashion to the LLM, the TTE also traverses the graph and applies predefined text-templates to the mapped patterns. The TTE extracts the labels and additional structural information from the labeled graph and fills the current text-template. Once the graph is traversed, the individually filled text-templates are joined together to form a textual representation of the graph.

Figure 17 shows one instance of a fully labeled model and a textual representation as an output of the presented composed generator. The output can be used to fill the placeholders <TEXTUAL MODEL DESCRIPTION> and <MODEL WITH LABELS> for the task that addresses the dimension *Create*.

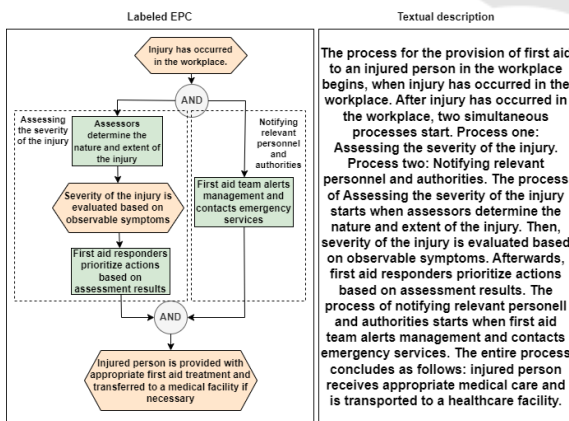


Figure 17: Instances for the input and output placeholders of the *Create*-task.

## 6 CONCLUSION

Graphical conceptual modeling is an important competency in various disciplines, and its mastery requires self-practice and the exposure to tasks that address different cognitive processing dimensions. The production of such tasks in large numbers is challenging, and current automatic production methods either lack scalability or fail to address higher cognitive processing dimensions.

This paper proposes a generalized AIG approach and introduces new generation methods for the production of items. The proposed solution is able to produce graphical conceptual modeling tasks in large numbers that address all cognitive processing dimensions according to the revised Bloom’s taxonomy, and thus meets requirement *R1*. The approach is capable of controlled generation of items with different degrees of complexity and difficulty on the fly, and thus meets requirement *R2*. This may facilitate advancements in computerized adaptive testing. Conversely, the approach also allows for *freezing* parameters that govern the complexity and merely alters surface level features. This allows for the scalable generation of fair exams, that provide different items for each examinee and thus reduces the risk of cheating, without compromising on consistent difficulty levels between exam instances.

The approach presented makes no assumption regarding the presentation and interaction layer for the task types presented. As the approach targets mostly task types with open response-formats, such as constructed-response, future work should consider unifying technology-enhanced items and AIG. Utilizing computerized assessment environments allow for data mining and learning analytics. This may open the path for future work on how to provide individual feedback for a solution attempt on an automatically generated item.

## REFERENCES

ACM, A. f. C. M. (2021). Curricula recommendations. American Educational Research Association and American Psychological Association (2014). *Standards for Educational and Psychological Testing, National Council on Measurement in Education (2014)*. American Educational Research Association, Washington, DC.

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Richard, M., Pintrich, Raths, J., and Wittrock, M. C. (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives*.

Bogdanova, D. and Snoeck, M. (2017). Domain modelling

- in bloom: Deciphering how we teach it. In *The Practice of Enterprise Modeling*.
- Bork, D. (2019). A framework for teaching conceptual modeling and metamodeling based on bloom's revised taxonomy of educational objectives. In *Hawaii International Conference on System Sciences*.
- Delcambre, L. M. L., Liddle, S. W., Pastor, O., and Storey, V. C. (2018). A reference framework for conceptual modeling. In *Conceptual Modeling*, pages 27–42. Cham. Springer International Publishing.
- Dunn, P. K. and Marshman, M. F. (2019). Teaching mathematical modelling: a framework to support teachers' choice of resources. *Teaching Mathematics and its Applications: An International Journal of the IMA*, 39(2):127–144.
- Falcão, F., Pereira, D. M., Gonçalves, N., De Champlain, A., Costa, P., and Pêgo, J. M. (2023). A suggestive approach for assessing item quality, usability and validity of automatic item generation. *Advances in Health Sciences Education*, 28(5):1441–1465.
- Ghosh, S. and Bashar, R. (2018). Automated generation of e-r diagram from a given text in natural language. *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)*, pages 91–96.
- GI, G. f. i. e. (2017). Rahmenempfehlung für die ausbildung in wirtschaftsinformatik an hochschulen.
- Gierl, M., Lai, H., and Turner, S. (2012). Using automatic item generation to create multiple-choice test items. *Medical education*, 46:757–65.
- Gierl, M. J. and Lai, H. (2016). A process for reviewing and evaluating generated test items. *Educational Measurement: Issues and Practice*, 35(4):6–20.
- Gierl, M. J., Lai, H., and Tanygin, V. (2021). Advanced Methods in Automatic Item Generation.
- Gierl, M. J., Zhou, J., and Alves, C. (2008). Developing a taxonomy of item model types to promote assessment engineering. *The Journal of Technology, Learning and Assessment*, 7(2).
- Guarino, N., Guizzardi, G., and Mylopoulos, J. (2019). On the philosophical foundations of conceptual models. In *European-Japanese Conference on Information Modelling and Knowledge Bases*.
- Haataja, E. S., Tolvanen, A., Vilppu, H., Kallio, M., Peltonen, J., and Metsäpelto, R.-L. (2023). Measuring higher-order cognitive skills with multiple choice questions –potentials and pitfalls of finnish teacher education entrance. *Teaching and Teacher Education*, 122:103943.
- Haladyna, Mark R. Raymond, T. M. S. L., editor (2015). *Handbook of Test Development*. Routledge, New York, 2 edition.
- He, X., Ma, Z., Shao, W., and Li, G. (2007). Metamodel for the notation of graphical modeling languages. volume 19, pages 219–224.
- Keller, G., Scheer, A.-W., and Nüttgens, M. (1992). *Semantische Prozeßmodellierung auf der Grundlage" Ereignisgesteuerter Prozeßketten (EPK)"*. Inst. für Wirtschaftsinformatik.
- Kucharski, S., Damnik, G., Stahr, F., and Braun, I. (2023). Revision of the aig software toolkit: A contribute to more user friendliness and algorithmic efficiency. pages 410–417.
- Kurdi, G., Leo, J., Parsia, B., Sattler, U., and Al-Emari, S. (2020). A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education*, 30(1):121–204.
- Laduca, A., Staples, W. I., Templeton, B., and Holzman, G. B. (1986). Item modelling procedure for constructing content-equivalent multiple choice questions. *Medical Education*, 20(1):53–56.
- Meike, U. and Constantin, H. (2023). Automated assessment of conceptual models in education. *Enterprise Modelling and Information Systems Architectures (EMISAJ) - International Journal of Conceptual Modeling*, 15, Nr. 2:1–15.
- Neuwirth, M., Finkensiep, C., and Rohrmeier, M. (2023). Musical Schemata: Modelling Challenges and Pattern Finding (BachBeatles). In *Mixing Methods. Practical Insights from the Humanities in the Digital Age*, pages 147–164. De Gruyter.
- Robinson, S. (2008). Conceptual modelling for simulation part i: Definition and requirements. *Journal of the Operational Research Society*, 59:278–290.
- Rudolph, M. J., Daugherty, K. K., Ray, M. E., Shuford, V. P., Lebovitz, L., and DiVall, M. V. (2019). Best Practices Related to Examination Item Construction and Post-hoc Review. *American Journal of Pharmaceutical Education*, 83(7):7204.
- Schüler, S. and Alpers, S. (2024). *State of the Art: Automatic Generation of Business Process Models*, pages 161–173.
- Soyka, C., Schaper, N., Bender, E., Striewe, M., and Ullrich, M. (2022). Toward a competence model for graphical modeling. *ACM Trans. Comput. Educ.*, 23(1).
- Stark, S., Chernyshenko, O. S., and Dragow, F. (2006). Detecting differential item functioning with confirmatory factor analysis and item response theory: Toward a unified strategy. *Journal of Applied Psychology*, 91(6):1292–1306. Place: US Publisher: American Psychological Association.
- Striewe, M., Forell, M., Houy, C., Pfeiffer, P., Schiefer, G., Schüler, S., Soyka, C., Stottrop, T., Ullrich, M., Fettke, P., Loos, P., Oberweis, A., and Schaper, N. (2021). Kompetenzorientiertes e-assessment für die grafische, konzeptuelle modellierung. *HMD Praxis der Wirtschaftsinformatik*, 58(6):1350–1363.
- Szczeńniak, B. (2011). Syntax errors in flat EPC diagrams made by persons learning the methodology. *Zeszyty Naukowe / Akademia Morska w Szczecinie*, nr 27 (99) z. 2:75–79.
- Taly, A., Nitti, F., Baaden, M., and Pasquali, S. (2019). Molecular modelling as the spark for active learning approaches for interdisciplinary biology teaching. *Interface Focus*, 9(3):20180065.
- Yirik, M. A., Sorokina, M., and Steinbeck, C. (2021). MAY-GEN: an open-source chemical structure generator for constitutional isomers based on the orderly generation principle. *Journal of Cheminformatics*, 13(1):48.