



University of Luxembourg

Faculty of Science, Technology and Communication

A Graph–Oriented Generic Data Model for Game-based Student Response Systems

Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of *Master in Information and Computer Sciences*

Author

Dren GASHI

Supervisor

Prof. Dr. Steffen ROTHKUGEL

Student Number

0130732528

Reviewer

Prof. Dr. Denis ZAMPUNIERIS

Date

August 2018

Advisor

Christian GRÉVISSE

Dedicated to my son, Nuh.

Abstract

Game-based student response systems (GSRs) have a great potential to motivate and engage students in classrooms. Using game-mechanics such as points or leaderboards as the main building block, quizzes feel like actual games. However, current GSRs share a main limitation. The students' answers must exactly match a solution, which is restricted in the way that there could be different semantically equivalent elements where each belongs to a valid solution. In this thesis, we present a graph-oriented generic data model that overcomes this limitation. By solving the well-known open graph isomorphism problem, the evaluation of an activity is able to accept any answer as long as it is semantically equivalent to the solution, although not specified by the teacher in beforehand. Furthermore, teachers can provide multiple graph-based solutions, thus offering more flexible questions. Inspired by the idea of teaching by templates, our approach supports graph-based templates that students can use as a starting point. This enables teachers to ask more complex questions that would otherwise take too much time in traditional game-sessions. Teachers can furthermore define patterns, which are sub-graphs of a solution graph, that can be used as an intermediate feedback system to show the progress on a solution. Additionally, we show how to create a bidirectional mapping between activities and learning materials so that students can access any learning material related to an activity within a game-session, and play associated activities from learning resources at home in order to foster a continuous learning experience. As a proof of concept, we have created three activities based on chemistry, computer science and chemical biology to demonstrate the applicability of our approach in different domains. Additionally, we provide a framework we developed to third-party developers for creating activities based on our data model. The evaluation of an activity is performed by the framework and is hence leveraged from concrete implementations.

Keywords: Graph-oriented Game-based Student Response Systems, Student Response Systems, Gamification, Student Engagement, Graph Isomorphism, Integration of Learning Materials.

Declaration of Honor

I hereby declare on my honor that I am the sole author of the present thesis. I have conducted all work connected with the thesis on my own.

I only used those resources that are referenced in the work. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

This thesis has not been presented to any other examination authority. The work is submitted in printed and electronic form.

Luxembourg, August 2018

Dren Gashi

Acknowledgments

I would like to dedicate this page to several people that supported me during my work and made this thesis possible.

I want to express my deep gratitude to my academic supervisor Prof. Dr. Steffen Rothkugel for his excellent support and guidance throughout the past months of my work. I am grateful for the constructive criticism which always caused me to rethink and helped me to improve. He always had a door open for me, and I want to thank him for all the rewarding discussions.

I would like to thank my advisor Christian Grévisse who has always taken the time to discuss my progress and helped me to come up with new ideas. I have considered him as a mentor, and I have learned a lot from him. I want to thank him for the excellent thesis template and all our interesting discussions, that even went beyond computer science.

I would like to acknowledge Prof. Dr. Denis Zampunieris for accepting to review my work. His excellent programming lectures in the Bachelor's degree were my inspirations for creating the Algorithmic Block activity.

I want to express my gratitude towards Prof. Dr. Pierre Kelsen and Dr. Jean Botev – the experts in graph theory – who have helped me resolving issues related to graphs. I learned a lot from our valuable discussions.

I would also like to thank Nicolas Mayer, Joe Mayer and Laurent Hentges for supporting me throughout my work in our shared office. I enjoyed our discussions and the rewarding feedbacks helped me to improve.

Furthermore, I would like to thank the directors of the *Lycée Technique d'Esch / Alzette* (LTE) for offering me the opportunity to start working as a teacher in computer science right after my studies. I also want to thank my former teachers Dr. Serge Linckels and Roger Kries who initiated this opportunity in the first place. Although they all were not directly involved in my work, they influenced the quality of my work and motivated me a lot with such an opportunity.

Last but by no means least, I want to express my deep gratitude towards my family who always believed in me and encouraged me. Although my seven month old son Nuh has not intentionally encouraged me, he surely did – indirectly – with all our lovely moments. To him, I dedicate this thesis.

Contents

Abstract	iii
Contents	x
List of Figures	xv
1 Introduction	1
2 Background & Related Work	5
2.1 Current GSRS	5
2.2 Other Teaching Tools	9
2.2.1 Teaching by Templates	11
2.3 Chapter Conclusion	13
3 Graph–Oriented Data Model	15
3.1 Meta–Model	15
3.2 Use–Cases	20
3.2.1 Molecule Activity	21
3.2.2 Algorithmic–Block Activity	23
3.2.3 Blood Count Analysis Activity	25
3.2.4 UML–Class Diagram Activity	27
3.3 Graph Comparison	29
3.3.1 Benchmark	37
3.4 Student Evaluation	37
3.5 Graph Properties & Optimization	39
4 Integration of Learning Materials	43

CONTENTS

4.1	Incorporating learning materials into activities	43
4.2	Linking activities with learning materials	44
5	Conclusion	47
5.1	Summary	47
5.2	Future Work	48
	Abbreviations	49
	Bibliography	53
	Appendix	55
A	Screenshots	55
B	Videos	70
C	Simulation study	75

List of Figures

2.1	Kahoot - Teacher Quiz Activity Creation GUI	6
2.2	Yactul - BuildPairs Teacher GUI	8
2.3	Yactul - BuildPairs Student GUI	9
2.4	A possible graph-representation for a concrete programming example	10
2.5	A possible graph-representation for H ₂ O water-molecule in the domain of chemistry	11
2.6	A graph-representation for the caffeine molecule in the domain of chemistry	12
3.1	Our graph-oriented generic data model that acts as a meta-model for all concrete activities	16
3.2	Example of a template with two different solutions in the domain of chemistry	18
3.3	Molecule Activity – Sub-Model created from our Meta-Model . . .	22
3.4	Molecule Activity – Example of a graph resulting from building a water molecule. The edges are unordered and undirected. Moreover, their type is "MoleculeActivityLinkEdge". All elements are within the same composite data-structure.	23
3.5	Algo-Block Activity – Sub-Model created from our Meta-Model . .	24
3.6	Algo-Block Activity – Example of a graph resulting from a control flow where the edges are ordered and directed.	25
3.7	Blood Count Analysis Activity – Sub-Model created from our Meta-Model	26
3.8	BCA Activity – Example of a graph produced from the corresponding sub-model composed of ordered and directed edges.	27
3.9	UML Class Diagram Activity – Sub-Model created from our Meta-Model	28

3.10	Graph Isomorphism approaches. The dotted arrows represent a substitution between two nodes.	30
3.11	A typical graph isomorphism algorithm would find an isomorphism between (a) and (b) since the graph structure is preserved, and a bijection can be produced to attain (b) from (a), which is of course not correct since nodes represent real world activity elements and thus have a meaning.	31
3.12	Simplified UML Class Diagram for the Graph Implementation . . .	32
3.13	Concrete use-case of a graph produced by a molecule activity. The nodes are enumerated only for identification purposes.	33
3.14	Example of substitutions between solution and student nodes. . .	35
3.15	UML Class Activity – Divide & Conquer approach is illustrated using different colors representing each a different type of relation. Each rectangle will represent a sub-graph G_i of the whole input graph G	36
3.16	Student Evaluation – Substitution between two graphs.	38
3.17	Strategy Design Pattern – UML Class Diagram to support multiple evaluation algorithms.	41
4.1	Integration of Learning Materials – UML Class Diagram using the Command design pattern.	44
4.2	Linking activities with learning materials – UML Sequence Diagram illustrating communication flow between different technologies to support opening related activities in documents.	45
1	Molecule Activity – The title page of our prototypical implementation.	55
2	Molecule Activity – Initially the from the template provided molecules are randomly placed and a scale animation is run together with a sound effect to simulate a pop-up.	56
3	Molecule Activity – Student has wrongly combined two oxygen atoms with two chemical bondings.	56
4	Molecule Activity – The evaluation shows a missing hydrogen molecule and colors the oxygen atoms red to illustrate an error.	57

LIST OF FIGURES

5	Molecule Activity – The solution of a water molecule is presented.	57
6	Molecule Activity – One out of three patterns were recognized, namely a bonding between two carbon atoms. The student receives an intermediate feedback to highlight this achievement. The feedback consists of an animation that pops up and fades out very fast to not disturb the student.	58
7	Molecule Activity – Two out of three patterns were recognized, namely a bonding between a carbon atom and three hydrogen atoms. The student receives an intermediate feedback to highlight this achievement. The feedback is based on the number of achieved patterns.	58
8	Molecule Activity – Three out of three patterns were recognized, namely a bonding between a carbon atom and three hydrogen atoms. The activity will shortly terminate, in favour of the student so that no more time elapses that would otherwise cause a point deduction.	59
9	Algo. Block Activity – The available statements on the right hand side are provided through the <i>tools</i> relationship. The student needs to drag-and-drop statements from right to left to create a control flow. The problem is about filling the marmite with potatoes from the left barrow.	59
10	Algo. Block Activity – The student is dragging a <i>while</i> -loop into his list of statements. The loop will show its content as soon as it is placed into the list.	60
11	Algo. Block Activity – The content of the <i>while</i> -loop is now visible and the student is dragging a condition into it.	60
12	Algo. Block Activity – The student has now finished his task and is ready to submit.	61
13	Algo. Block Activity – The evaluation were successful.	61
14	Algo. Block Activity – The solution matches the student's answer.	62
15	Algo. Block Activity – The evaluation were not successful since the student has put both actions in the wrong order which is illustrated using a yellow color.	63

16	Algo. Block Activity – Algo. Block Activity – The evaluation were not successful since the student forgot one action statement which is depicted using a red color. The missing element is injected using a gray color.	63
17	Blood Count Analysis Activity – The student needs to drag-and-drop health-items to the right anatomy in order to influence the counts.	64
18	Blood Count Analysis Activity – As soon as the student drag-and-drops all available health-items into the anatomy, the influences will start using a timer. Positive or negative influences are depicted using a green or red arrow, respectively.	64
19	Blood Count Analysis Activity – The student evaluation were not quite successful and the framework colors the wrong items in red.	65
20	Blood Count Analysis Activity – The solution is shown using ranges of accepted values.	65
21	Message alerting the student that the activity will be finished when presenting learning materials since this could influence the result or score. The message pops up after touching on red icon in the bottom left.	66
22	Learning materials for a molecule activity providing different links to ontologies about chemistry.	66
23	A Browser displaying ChEBI entry for water. ChEBI is a website implementing a dictionary of molecular entities and is backed by an ontology.	67
24	Learning materials in the form of PDF's for an algo-block activity are listed.	67
25	A PDF related to a programming course is presented to the student.	68
26	Dedicated HTTP Links can be opened using our app from everywhere in the iOS platform. Our app then handles query parameters, etc., and the dedicated task.	68

LIST OF FIGURES

- 27 The SoLeMiO add-in in Microsoft PowerPoint lists all resources that have been found for the particular lecture slides. Here, we have an URL and a video resource as listed in the right hand-side. The URL encodes an activity and will be opened by our dedicated app instead of the standard browser. 69
- 28 Benchmarking times – Time elapsed ordered in an ascending fashion where the times are mean values computed from 30 independent runs. An iPad Air 2 were used as a test device. The configurations are written as $(\#H, \#O, \#C)^T$ where H is the number of hydrogen, O the number of oxygen and C being the number of carbons. We can observe that the bigger the number of atoms become, the more time it takes to compute. The combination of two or more high numbers of atoms has a significant impact on the performance. The high peaks represent exponential increases . 75
- 29 Benchmarking times – Iterations ordered in an ascending fashion where the values are means computed from 30 independent runs. An iPad Air 2 were used as a test device. The configurations are written as $(\#H, \#O, \#C)^T$. One iteration represents one execution of the algorithm 1. The high peaks represent exponential increases. 76

LIST OF FIGURES

1 | Introduction

The application of game-design elements and game principles in non-game contexts, known as *gamification*, is the key building block for game-based student response systems (GSRs) [1]. As an active learning environment, GSRs take classic student response systems (SRSs) to a whole new level by making traditional quizzes feel like actual games, thus increasing student attendance, engagement and participation as reported by Morillas *et al.* [2]. Furthermore, a study conducted by Scott Freeman *et al.*, shows that students enrolled in a traditional STEM¹ lecture are 1.5 times more likely to fail than those enrolled in an active learning environment [3]. After all, considering that our present generation has grown up in a very technological environment, it doesn't seem very surprising that SRSs are well received by students nowadays.

GSRs make use of game mechanics such as points, levels, challenges, virtual goods, leaderboards or gifting & charity. These elements help students to compete with each other, to acquire rewards, status and self-expression, all based on human natural desires [4]. Because of this finding, game mechanics are specifically designed to soothe the students' ambitions. As reported by Hakulinen *et al.*, the usage of game mechanics has shown to impact the students' behaviour. Students with more badges² spent more time per exercise, suggesting that they thought more about the problem before submitting [6]. Badges and other achievements provide students with status and rank, ensuring challenge and competition within the class in order to gradually increase engagement while improving knowledge. As a human natural desire, competition is deeply ingrained in the human psyche [4], thus representing an important mechanism to capture students' attention, which is nowadays already difficult enough since very often smart devices seem to be more interesting than the lecture.

Most of the SRSs are web-based, as this is the most convenient solution to handle large heterogeneity of devices, which is clearly the case in traditional classes where today's students are interacting all the time with their mobile devices and laptops. Nowadays there exist several SRSs where most of them are commercial solutions. The amount of GSRs on the other hand is very small. One reason for this is that the term 'gamification' itself is new. The majority of the current GSRs only supports a very limited set of question types. Those question types that we call *activities*, only maintain a question title with several textual answers where

¹Science, Technology, Engineering and Mathematics

²Badges are a validated indicator of accomplishment, skill, quality or interest that can be earned in various environments [5].

one or more can be correct. There are, after all, a minority of current state-of-the-art solutions supporting more than one type of question. A recently proposed GSRs by Grévisse *et al.* [7] comes with a modular architecture that enables a very loose integration of new question types, as they are considered as separate and independent applications. Every activity can be developed separately and plugged into the eco-system, strictly following the separation of concerns (SoC) principle. Each activity application then handles its own business logic, data and student evaluation.

Yet, as the activities are very heterogeneous, a common model is usually not taken into consideration and therefore the different question types need to be separately designed from scratch. This might be a reason why many (G)SRs only offer a very small amount of distinct question types. As most of today's (G)SRs are commercialized, gathering insights in the architecture is very difficult. Nonetheless, the fact that activities are very diverse from their essence and only a very limited amount of use cases exist suggests that there is no common model on which different types of questions are based. An additional effort (and the economical constraints) for a dedicated design would simply not be worth it, considering the minimal requirements, i.e. the support for only a small set of question types. Also, the evaluation of the students' answers is distinct for each and every activity which could be unified using a common model. Another shortcoming is that current SRs only provide one single solution for an activity which were to-date fairly adequate for very simple and non-game activities. However, with more game-based question types, multiple distinct solutions become more realistic.

Furthermore, current (G)SRs do not provide a direct link between questions asked during a game session and the corresponding content of learning materials. But we believe that a continuous learning experience by revisiting course related activities at home while learning is important for fostering the learning process of the individual student.

A common model would be beneficial for many reasons. First of all, the implementation process would follow a common guide, thus being homogenized. Developers can hence write code that is applicable to many different activities and activity components. For example some visual effects such as a fancy explosion can be reused in many different activities. This reusability feature could also be applied to activity components. A line segment or simple shapes that are often present in many activities would only need to be designed once and could then be used many times among different activities. Furthermore, states, actions and effects that are applied to game-based activity components could be encapsulated in modules and shared as well.

Introducing logical links between components for relational purposes would cause

CHAPTER 1. INTRODUCTION

the creation of a graph which then could be used for generalizing the evaluation process because we would abstract from concrete types, and only work with nodes and edges. The graph-oriented approach would moreover allow to accept different, but semantically equivalent solutions. An example for this would be an activity about chemical molecules where students need to drag-and-drop atoms and combine them using edges to build chemical compounds. Suppose a student needs to create a water molecule (H_2O). If there are multiple oxygen and hydrogen atoms, semantically all equal but still distinct instances of the same type, then how would traditional evaluation algorithms deal with such an *ambiguous* situation? The graph-oriented method could handle this by solving the well-known computational graph *isomorphism* problem, hence allowing any combination of semantically equivalent elements of a solution to be valid. Also, the graph-oriented approach would allow teachers to define multiple graphs as solutions, resulting in more flexible questions. Nodes and edges could also be annotated with semantics for a fine-grained integration into learning material to provide students with a continuous learning experience by enabling them to revisit concepts and lead to increased performance in subsequent replays.

Chemical
Molecules
Example

Ambiguity
Problem

With our goal to create a main-model for game-based activities, we propose a graph-oriented generic data model that facilitates development, makes the most of reusability, gives teachers the opportunity to provide different possible solutions – as diverse graphs –, enables a fine-grained integration of activities into learning materials and unifies the evaluation process. Our generic data model acts as a *meta-model* for all concrete activities. Different activity-specific models can be realized using our meta-model. Components with their corresponding behavioural business-logic such as actions with their visual effects are encapsulated in modules and ready to be reused for any type of activity. Upon creation of an activity, teachers can specify more than a single solution, all backed by different graphs. Teachers can also specify a *template*, which is a pre-defined graph based on a solution, that students can use as a starting-point to work on. Additionally, *tools* can be specified per activity. These are elements the students can use to create their graph and manipulate the template. Furthermore, teachers can define *patterns*, which are sub-graphs of a solution graph. These are meant to be used as an intermediate feedback system and help students highlight their current progress. Nodes are semantically enriched with meta-information to be compliant with contents from learning materials so that students can play activities given a concrete learning concept. The evaluation algorithm is generic and takes as input two graphs where one is constructed by the student during gameplay and the other represents a solution of a specific activity. By solving the graph isomorphism problem, the algorithm is able to evaluate the graph for correctness and enrich the nodes with additional information about possible errors.

Coming back to our example with the chemical molecules, the graph evaluation algorithms takes care about elements or nodes that are semantically equivalent and accepts – strictly speaking – different instance-solutions as the one provided by the teacher since the equivalence is no longer checked using a strict value-comparison. This is the main advantage of our work compared to current state-of-the-art solutions.

Our concept is realized as an iOS framework that provides GSRs developers the important tools to create outstanding activities while making the most of reusability using our data model. The framework also handles the evaluation process which hence does not need to be implemented by the concrete activities.

Our objectives were driven by the following research questions.

Research Questions:

- RQ1** Assuming activities are built on a graph-based model, can the evaluation of an activity then be generalized so that it is the same among different activities?
- RQ2** What are the advantages of such a model?
- RQ3** How can we integrate activities into learning materials in order to establish a continuous learning experience?

Chapter Overview: In chapter 2, we analyse current GSRs and show the main shortcoming that our work overcomes. We additionally present some ideas of other teaching tools outside the realm of (G)SRs that could be very rewarding to have implemented in GSRs based on our model. Our graph-oriented generic data model is elaborated in chapter 3 together with some use case examples of activities based on our proposed model. We also show how the students' answers can be evaluated using graphs. Furthermore, we discuss some optimization options and graph properties. Chapter 4 describes how we can create a mapping between learning materials and activities so that students can play related activities during their learning process at home. Finally we conclude in chapter 5 and present some future work extensions.

2 | Background & Related Work

Contents

2.1	Current GSRS	5
2.2	Other Teaching Tools	9
2.2.1	Teaching by Templates	11
2.3	Chapter Conclusion	13

In this chapter, we will present two GSRSs from both commercial and research fields, and discuss how activities are created and evaluated in current approaches. Since to the best of our knowledge we could not find anything closely related to our work, we consider our approach as *novel* and we will thus show additional ideas of possible activities that could be created using our method, that would be otherwise difficult to design with current approaches. We were also inspired from teaching tools outside the realm of (G)SRS to gather missing features in GSRSs that could be beneficial to have.

2.1 Current GSRS

The number of today's GSRSs is very limited. Most of them are commercial solutions, but there are also some initiatives driven by research. The probably most popular commercial GSRS is *Kahoot*¹. *Yactul*² in the other hand, is a recently proposed GSRS by Grévisse *et al.* [7] that has emerged to overcome the limitations of current solutions.

Kahoot currently offers four different activities: *Quiz*, *Jumble*, *Discussion* and *Survey*. A Quiz is a sequence of questions where one or more answers can be correct. A Jumble is a sequence of questions where students need to drag answers in the correct order. A Discussion is a way to initiate a debate or brain-storming with the students and Survey can be used to gather the audience's opinions. Both Quiz and Jumble are gamified whereas Discussion and Survey do not support any game mechanic. Also, Quiz and Jumble are the only activities that can be evaluated, i.e. compared with a solution provided by the teacher.

Kahoot itself only gives very little insights into their architecture. We know that their architecture is build on *micro-services*, where each application is responsible for handling a small part of their business-logic [8]. However, we do not know

Kahoot

¹<https://kahoot.com/>

²<https://yactul.uni.lu>

Experiment
Kahoot

whether they are using a common model for their activities, and whether the model is graph-based. However, we can demonstrate the ambiguity problem using a simple experiment with our chemical molecule activity example as mentioned in chapter 1. To the best of our knowledge, neither Kahoot nor any other GSRSs natively supports such an activity, but we can still illustrate the shortcoming. We will use Kahoot's Quiz type for this example. In the following screenshot (figure 2.1), we can see the graphical user interface (GUI) used by teachers during activity creation.

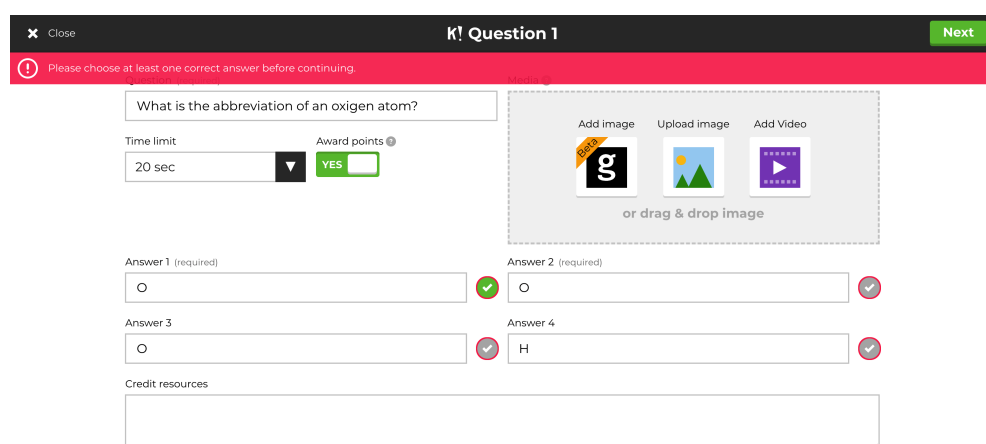


Figure 2.1: Kahoot - Teacher Quiz Activity Creation GUI

Besides the limitation of only allowing up to 4 answers, the teachers need to specify which answer is correct. Note that the question is asked in such a way that more answers can be correct. This feature is of course possible, but it requires the teacher to select each and every element by hand. Without a restriction on the number of possibly correct answers, this setup would certainly not be feasible as the number of elements grows. Also we shall point out that this scenario is illustrated with only one *variable*. Imagine having a question that requires to create a water molecule by combining any two hydrogen atoms with one out of multiple oxygen atoms. In this example, there would be two variables, namely "O" and "H". Thus any combinatorial enumeration of existing oxygen and hydrogen atoms would need to be specified in before-hand. To emphasize this, suppose the student plays with two oxygen instances, O_0 and O_1 , and two hydrogen instances, H_0 and H_1 . Furthermore, assume the teacher specifies the solution as (O_0, H_0) and (O_0, H_1) , where (x_i, y_j) is a *relation* between x_i and y_j symbolizing a link between atoms to represent a chemical bond. However, the student has now several possibilities to come up with a semantically equivalent solution. Those can be described as

$$\{(O_i, H_j), (O_i, H_k)\} \mid \forall i, j, k \in \{0, 1\}, j \neq k\},$$

resulting in

- 1) $\{(O_0, H_0), (O_0, H_1)\}$
- 2) $\{(O_0, H_1), (O_0, H_0)\}$
- 3) $\{(O_1, H_0), (O_1, H_1)\}$
- 4) $\{(O_1, H_1), (O_1, H_0)\}$

where the first listing corresponds to the solution provided by the teacher. A possibility to handle such situations would be to allow multiple solutions, but the teacher would need to enumerate them by hand. The evaluation process would then check all possible solutions until it finds a match. Simple boolean algebra could be used as follows:

$$\begin{aligned} & \{(O_0, H_0) \wedge (O_0, H_1)\} \vee \\ & \{(O_0, H_1) \wedge (O_0, H_0)\} \vee \\ & \{(O_1, H_0) \wedge (O_1, H_1)\} \vee \\ & \{(O_1, H_1) \wedge (O_1, H_0)\}. \end{aligned}$$

But this is certainly not scalable. Another problem with such an approach would be that it only works if the teacher has a *deterministic* view of the elements of an activity in before-hand. Activities where students can create additional elements cannot be solved using this method as the teacher would no longer be able to enumerate all possible solutions.

Although we do not know if Kahoot is using a graph as the main model for their activities, we can however motivate that using a graph-oriented method would make the evaluation more flexible by comparing semantically equivalent nodes, thus allowing any solution to be valid as long as the graph structure is preserved. This characteristic could handle situations as described in the experiment.

Yactul currently supports six different activities that can be all evaluated. The GSRS has been designed with the goal to leverage the tightly coupled relation of activities with the core system by making its architecture extensible wrt. new quiz types as stated by Grévisse [7]. The architecture is designed in a very similar way to Kahoot's microservices. Activities are implemented as individual and separate applications, each encapsulated in a module to facilitate the plugging into the ecosystem. Yactul is providing a minimalistic model on which activities are based. However, the model does not cope with activity specific elements, but rather with meta-information (figure 2.2) such as activity name, question title, time, difficulty level, *et cetera*. The evaluation for a specific activity is outsourced from the eco-

Yactul

system and implemented by each concrete activity application to make the most of the separation of concerns (SoC) principle.

Although it is known that activities in Yactul are not graph-oriented, we can make a similar experiment as above to illustrate the same shortcoming. For this, we will use the quiz type *Build Pairs*, where students need to connect two related elements to build a pair. This process is being repeated until all available elements are within a pair. As in the first experiment, we will use a chemistry example. In the following figure, the GUI for creating BuildPairs activities is depicted.

Figure 2.2: Yactul - BuildPairs Teacher GUI

As we can see in figure 2.2, there are several identical elements. However, internally those elements are being distinguished using an identification (ID) key which causes the evaluation algorithm to only accept answers that have the exact same ID as the solution. The students have thus no chance to figure out which solution elements were taken by the teacher as seen in figure 2.3.

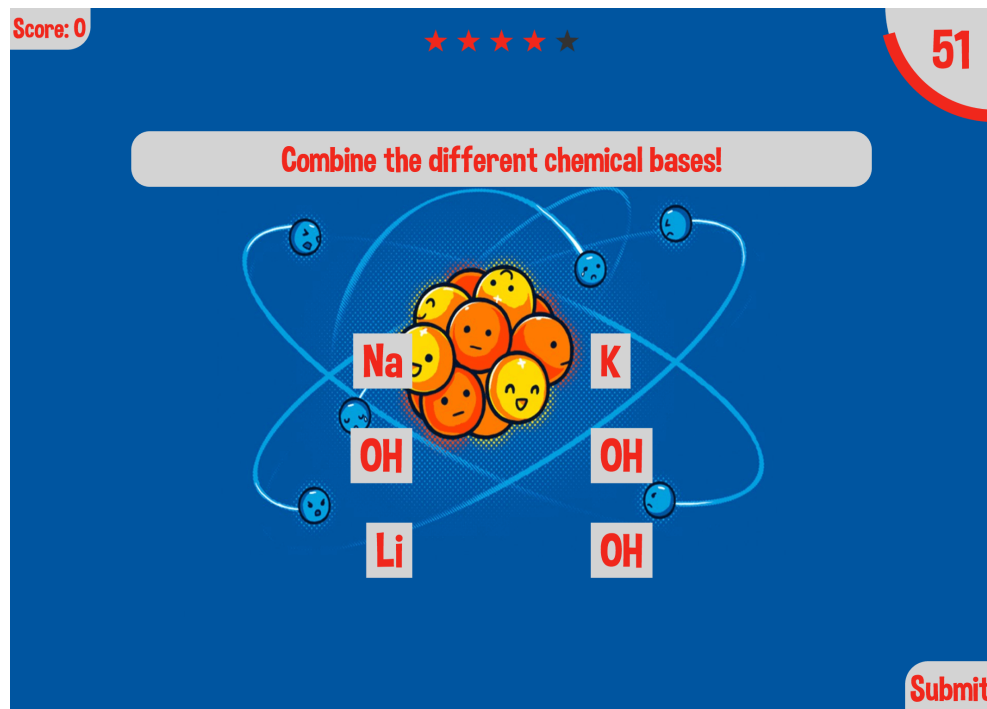


Figure 2.3: Yactul - BuildPairs Student GUI

The evaluation process will not be successful if the student did not combine the exact same instances as specified by the teacher. This ambiguous situation could be avoided using a graph as the main model for activities because the algorithm would compare the graph structure rather than concrete instances.

2.2 Other Teaching Tools

In this chapter, we will analyse some features outside of the realm of (G)SRS in order to gather useful attributes that are missing in current (G)SRS that could be an integral part of a graph-oriented data model. As a concrete instance to illustrate the problem, we considered a programming activity which is, to the best of our knowledge, non-present in today's (G)SRS. Also, such an activity would help students in computer science classes in many ways. Kelleher *et al.* [9] report that using graphical symbols for programming tasks simplifies the syntax because only certain combinations of symbols are allowed, so that syntactic rules are inherent and students do not need to cope with this difficulty. Indeed, the syntax represents a real obstacle to students of introductory programming courses because it prevents them to focus on the logic, structure and problem solving involved in programming as observed by Po-Yao [10].

Programming
Activity
Example

Most of activities of this kind are used in *visual programming environments* (VPEs) and *intelligent tutoring systems* (ITSs). VPEs are tools that make use

of graphical symbols which can be interactively manipulated in such a way that they result in an executable program. ITSs in the other hand, are, in many ways, very similar to human tutors [11]. They provide customized instructions with immediate feedback to learners. Based on cognitive science and Artificial Intelligence (AI), ITSs have proven their worth in multiple domains in Education [12][13]. For programming tasks, current ITSs expect student to write code that can be compiled on the fly and evaluated for correctness. These kind of tools are suited for more experienced students that already know the syntax.

A graph-oriented model would be excellently suited for a programming activity. Every programming concept such as conditionals, loops or ordinary statements can be represented as a node in a graph. *Directed* and *ordered* edges between nodes would represent following statements. In the figure 2.4, a possible graph-representation of a program is depicted that could be created using a graph-oriented model for activities.

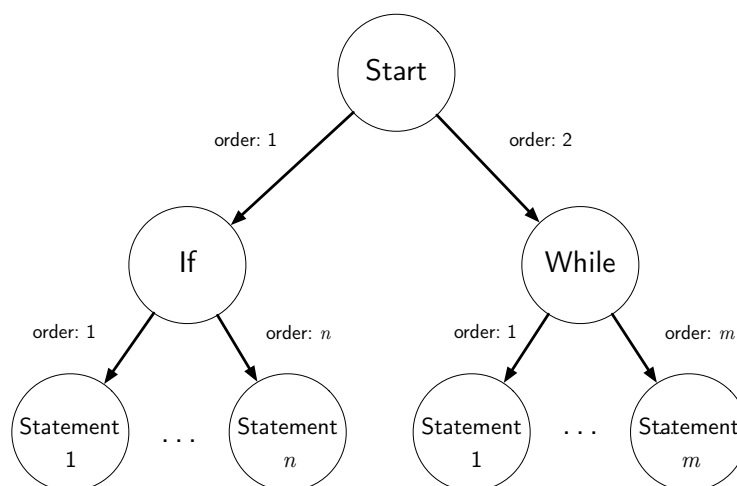


Figure 2.4: A possible graph-representation for a concrete programming example

Edges are ordered to represent the position in a sequence of statements (nodes). Teachers could then specify a solution and do not need to worry about unique elements. The evaluation algorithm would cope with semantically equivalent elements, and accept any solution as long as the graph structure is preserved.

Moreover, we shall point out that the applicability of a graph-oriented approach is not limited in the domain of programming. For instance, Chem-Tutor³ [14] is an ITS for undergraduate and high-school students to learn about foundational chemistry concepts related to atomic structure and bonding. In the academic research, Gupta *et al.* propose *Chemistry studio*, an ITS which aims to automatically solve problems, in the domain of Periodic Table and its properties, using proper logic and reasoning to generate solutions and explanations in accordance with the

³<https://chem.tutorshop.web.cmu.edu/>

interest and knowledge of the student [15]. Let us take this idea for a concrete chemistry–activity example to showcase the usability of a graph–oriented model using the molecule activity as described in chapter 1. In contrast to the programming activity where we had directed and ordered edges, molecules or atoms would now be connected using *undirected* and *non-ordered* edges. In figure 3.4, a possible graph representation is depicted where atoms are represented by nodes. Here, a water–molecule is created by combining two hydrogen atoms with an oxygen atom.

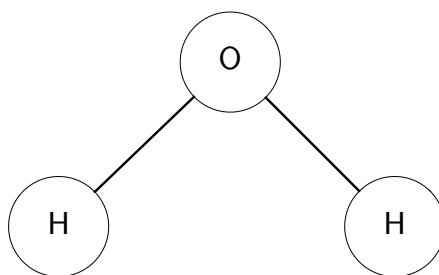


Figure 2.5: A possible graph–representation for H_2O water–molecule in the domain of chemistry

In such an activity, a graph–oriented model would be well suited since the redundancy of the elements is higher than in other activities because many chemical compounds include a lot of elements of the same type, thus many different but semantically identical solutions would arise. Chemical bonds would be represented as undirected edges, and this already forms a graph without any pre-transformation, which is very practical.

2.2.1 Teaching by Templates

Teaching by templates is the idea of guiding students, through the use of templates, in their learning processes. Many states that templates help to achieve a well planned learning path and enforce the important concepts and constructs [16]. Today’s (G)SRSs do not offer such a feature, and we believe that this idea is very useful for students. Teachers can provide a template in form of a graph that acts as a starting point, and students can hence interactively manipulate the template until they consider it as finished and ready to be evaluated. Such a feature would introduce new *kinds* of questions. Teachers could ask questions about correcting a given template. The ability of *spotting* and correcting an error could be tested. Or, questions about expanding and completing a template could be interrogated. This would allow teachers to ask more complex questions by providing a certain percentage of the solution, that would otherwise take a lot of time for students to create. Current (G)SRSs are designed to provide time-based questions, and the work of reaching the solution takes time. Therefore, a lot of

2.2. OTHER TEACHING TOOLS

solutions are fairly small because students will be busy by interactively working with the game-elements of an activity. Templates could provide help, by allowing teachers to pre-define a subset of the solution that students can take as a starting point. To illustrate this, let us take the molecule activity example as mentioned in chapter 1. In the following figure, an example of a template for the molecule activity is presented.

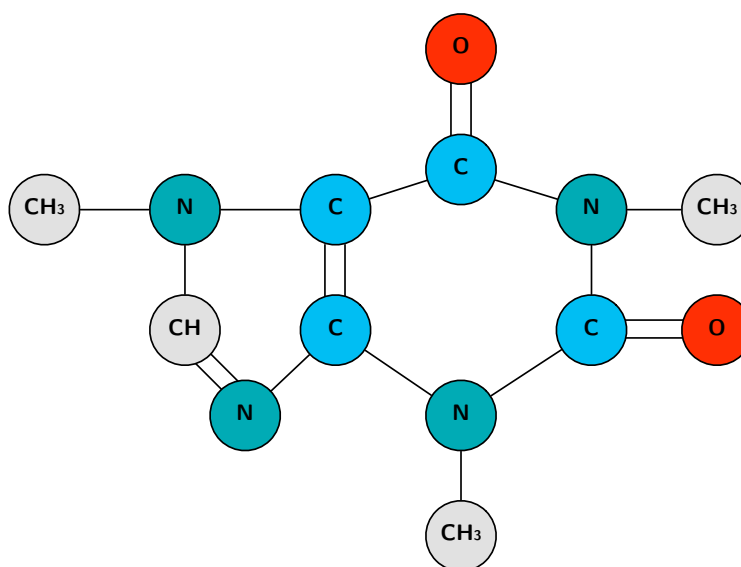


Figure 2.6: A graph-representation for the caffeine molecule in the domain of chemistry

The molecule depicted in figure 2.6 represents the quit complex molecule caffeine. In a molecule activity, the students would need to drag-and-drop atoms and combine them using links, which takes time to finish. Although this time-consuming gameplay is very important for gamification reasons, such complex molecules would rather not be part of a game-session in a chemistry class due to time constraints. However, using templates, the teacher could provide a subset of the solution graph and ask the the student to extend it. Another possibility would be to introduce errors, and ask students to fix them. Challenging questions could be templates that are exact same as the solution and ask students to find errors, although there is none.

We consider a graph-oriented data model as perfectly suited for the incorporation of templates. The template would consist of a graph, either a *sub*- or *superset* of the solution-graph. Students can thus interactively manipulate the template until they submit it for evaluation.

2.3 Chapter Conclusion

In this chapter, we have presented some current GSRs and their main short-coming about not being capable of handling different solutions that are, however, semantically equivalent. In current systems, teachers would need to enumerate all possible solutions in before-hand, assuming the elements of an activity are *static*. Each quiz type in current systems handles the evaluation separately, although this could be generalized using a graph-oriented data model. We have shown that concepts from different domains such as programming or chemistry can be seen as nodes. Edges between nodes could be used to represent relations between concepts. For molecules, the edges would serve to create chemical bonds between two atoms. For programming concepts, a directed and ordered edge would represent an arranged flow from one statement to another. The evaluation of a graph-based model would rely on the preservation of the graph structure.

Teaching by templates is a very interesting idea that we believe could be beneficial to integrate into GSRs. They offer more flexible questions and have the potential to help students in their learning process by providing a starting point. Moreover, they enable teachers to provide more complex questions that would otherwise take too much time to ask in a classroom. A graph-oriented data model is perfectly suited for incorporating templates because they would be considered as a sub- or superset of the solution graph.

2.3. CHAPTER CONCLUSION

3 | Graph-Oriented Data Model

Contents

3.1	Meta-Model	15
3.2	Use-Cases	20
3.2.1	Molecule Activity	21
3.2.2	Algorithmic-Block Activity	23
3.2.3	Blood Count Analysis Activity	25
3.2.4	UML-Class Diagram Activity	27
3.3	Graph Comparison	29
3.3.1	Benchmark	37
3.4	Student Evaluation	37
3.5	Graph Properties & Optimization	39

This chapter covers the core of our work which is the graph-oriented generic data model. We will present and motivate the main concepts of our approach. Next we will show some concrete use-cases derived from the our data-model. Then, we will describe how two models of the same type can be compared without prior knowledge of the type of nodes. This is related to the evaluation process of an activity since the students' answers will consist of graphs which then need to be compared against one or more solution graphs. Next we will elaborate on the points distribution during the student evaluation process. Finally we will discuss some graph-properties and consider some optimization options.

3.1 Meta-Model

Our graph-oriented generic data model acts as a *meta-model*. Kelsen *et al.* describe meta-modelling as the process of defining a model as a language for expressing other models [17]. The sub-models are thus built by instantiating the *metaclasses* and *associations* from the meta-model. The meta-model is used for each concrete activity as a fundamental basis. Our meta-model is presented in figure 3.1.

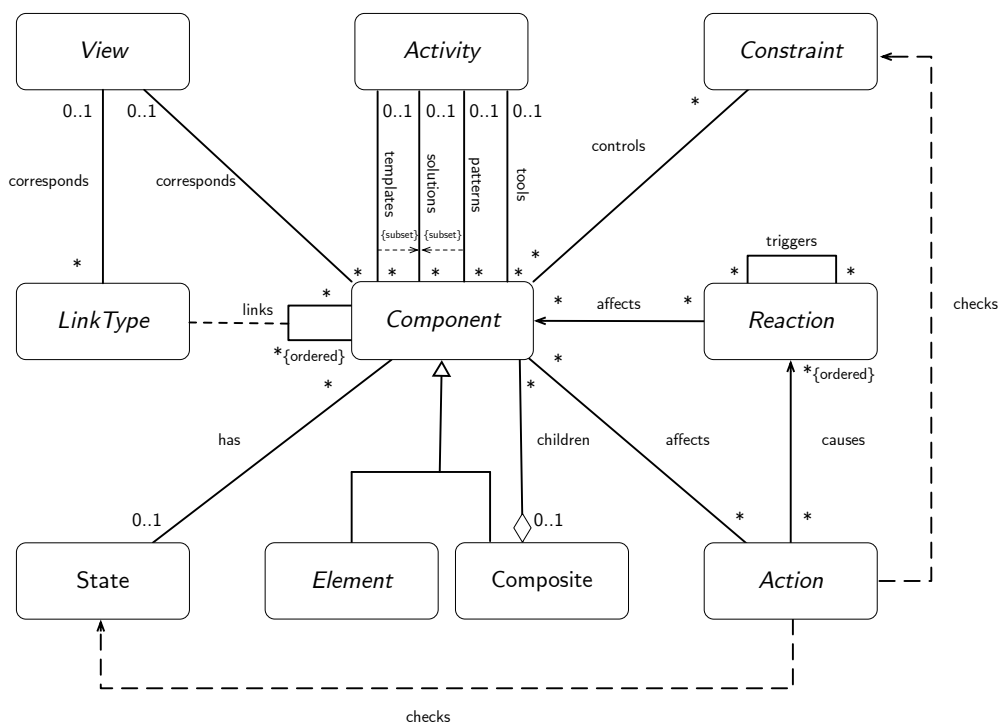


Figure 3.1: Our graph-oriented generic data model that acts as a meta-model for all concrete activities

The Composite Design Pattern, proposed by the Gang of Four (GoF) [18], is applied to treat a group of elements the same way as a single instance of the same activity type. We consider this feature as very useful in the context of GSRs. Considering our molecule example from chapter 1, a molecule would be implemented as a composition of multiple atoms. This adequately simplifies the implementation. For instance, moving and dragging a molecule is made simpler because we would consider a set of atoms as a single instance. Note that the design pattern is *optional*, so that developers can choose whether or not they want to adopt it. A *component* or *element* represents an entity in an activity that students can interact with.

The Component–LinkType relation is used to create the graph. Elements are used as nodes and an edge between two components is realized as a logical *typed* link (*LinkType*). The type is important for supporting links that have different meanings. A concrete example for this is an UML Class Diagram. We could image an activity where students can create and manipulate UML classes for learning *polymorphism* or *inheritance* concepts as often required in object-oriented programming (OOP) tasks that are otherwise difficult to ask in current activity types. Each relation such as association or inheritance can be denoted using a different typed edge. We need to distinguish the edges for the evaluation later on. It is certainly important to know what type

of edge the student has used to connect two components. An inheritance edge for instance is different than an implementation edge, thus the evaluation algorithm will need to distinguish such cases when comparing against a solution.

Components and links can both have an optional *view* which is an abstract representation of a graphical content that will be later used in an activity. The edges can be *ordered* so that we can define a position of an element in a sequence of components. Considering our programming example from chapter 2.2, an ordered edge would represent an arranged flow from one statement to another. A *while*-block can contain several statements that have a certain position in a sequence. From the point of view of a graph, a *while*-node would comprise several edges to it's containing elements where the order represents the position in a sequence as illustrated in figure 2.4. Furthermore, links between nodes in such a programming activity would not have a view. The link is only *logical* and is needed for creating the graph. For the molecule activity in the other hand, links would have a view that would consist of an undirected line segment between two atoms. Moreover, the meta-model does not specify a *direction* for an edge. The interpretation whether an edge is directed or undirected should be specified in the sub-models of concrete activities. For instance, in our molecule example, the edges are undirected because the direction is not important, i.e. $O-H = H-O$ with H being a hydrogen atom and O being an oxygen atom. However, considering our programming activity, the direction is gaining more importance. Directed edges are defining the control flow from one statement to another. Having two statements s_a and s_b , then clearly $s_a \longrightarrow s_b \neq s_b \longrightarrow s_a$ since program statements are written in a sequential order.

The Activity–Component quaternary relation is used to define which components are part of the solutions, templates, available tools or patterns. Solutions, templates and the patterns are all graphs that are defined by the teacher. The tools relation defines what elements students have at their disposal. This feature can be used to only show elements related to some aspects covered in class. For instance, in an UML class activity, we could imagine that students can only use inheritance and associations on classes, and interfaces with realization features are not available because of the current progress of the class.

The pattern relationship is used to define several sub-graphs created from a solution graph. This feature helps students to follow their progress, and acquire intermediate feedback. Furthermore, if n out of n patterns were found, the activity immediately terminates in favour of the student. This way, no points can be deducted because of additional time expenses.

3.1. META-MODEL

Patterns and templates can be created from a given solution, therefore the relation is denoted with the subset constraint. However, concerning template, the opposite is also true, resulting in the template being a superset of the solution graph by introducing additional elements that are not present in the solution. The teacher can specify a template for a given activity on which the students can then work on. A template is optional, so it is up to the teacher to decide whether he would like to provide one. The solution graph is, of course, compulsory but also not limited to one. The teacher can specify different solutions per activity which we believe is very useful because it offers more flexible questions. Considering our molecule activity, the teacher could ask a question about building an alcohol molecule such as *ethanol* or *methanol*. Through a template, one could provide atoms for both, resulting in two different solutions. An example of this is depicted in the figure 3.2 where the teacher does not specify any links but rather only components. These relations have emerged as a response to the research question **RQ2** which is about advantages of a graph-oriented model. Solutions, templates and patterns are each represented by a graph, independent of the concrete type of activity which makes the implementation very simple.

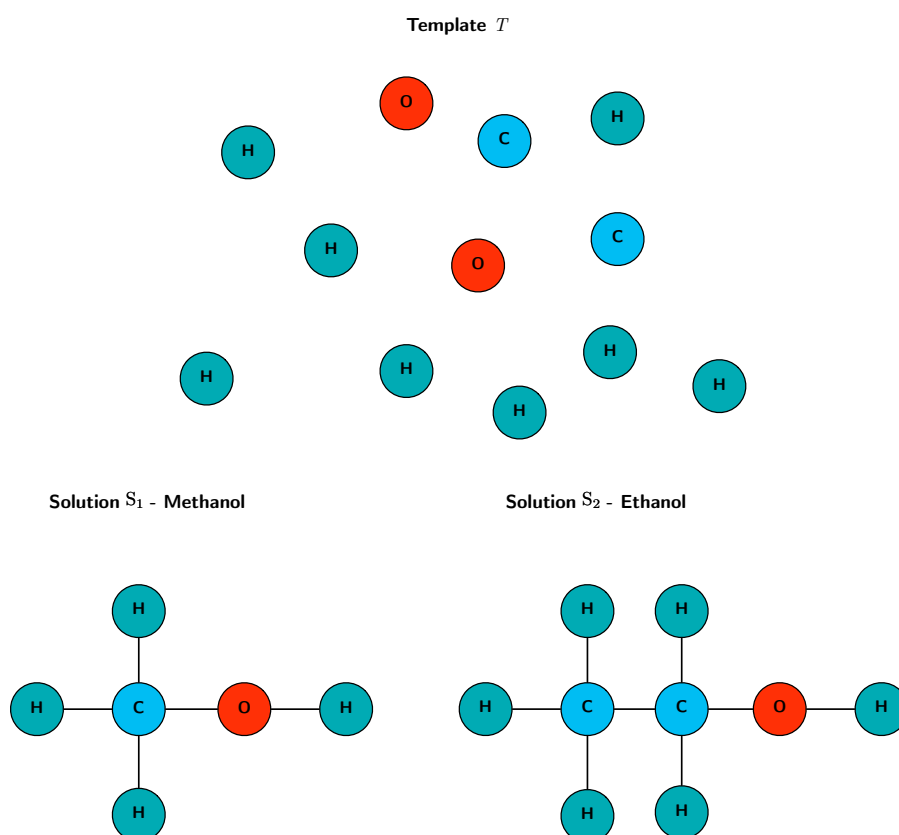


Figure 3.2: Example of a template with two different solutions in the domain of chemistry

We can observe that $S_1 \subset T$ and $S_2 \subset T$ or $T \supseteq S_1$ and $T \supseteq S_2$, considering only the components. There are, of course, more than two alcohol molecules. However, given the template, only those two were possible. The template feature might be very useful depending on the type of activity, but it requires the teacher to carefully think about the questions in advance since he might not have thought about all possible solutions.

Component–State relation is useful for *behavioural* reasons. For instance, in our molecule activity, atoms could maintain a state

$$s \in \{\text{NOT_CONNECTED}, \text{CONNECTED}, \text{FULLY_CONNECTED}\}$$

where the atoms *act* differently depending on s . For example, two atoms, i and j , would accept a chemical bond with each other, if

$$s_i \neq \text{FULLY_CONNECTED} \wedge s_j \neq \text{FULLY_CONNECTED},$$

meaning that i and j have not reached the maximal number of bonds which is limited by their *chemical valency* constraint. Furthermore, states can contribute to the gamification of an activity by drawing the elements differently depending on their state. Visual effects and sounds can be various across different states, introducing a *dynamic* user experience. Moreover, the states can help to simplify the evaluation process. In this example, it is more convenient to compare two different states rather than compute a common property.

Component–Constraint relation is used to define clear restrictions for elements in activities. As a concrete example, the number of possible chemical bonds of an atom is restricted by its valency number. Or, the condition block of a *while* concept can only accept a statement that is a condition. We cannot put a *Loop* concept into the condition part of another *Loop* statement. This design is meant to simplify the development by outsourcing the constraints so that other classes such as *Action* can use them for validation purposes.

The Component–Action–Reaction ternary relation is based on the *causality* principle from physics. Bunge describes causality as a *relationship* between causes and effects [19]. In simple words, if A caused B , then A happened before B , which implies that causes always occur before their resultant effects as stated by Harris [20]. For GSRs, this idea can be applied because every student interaction will cause something that is either *valid* or not. Many *actions* are caused by the student by manipulating elements. The actions executed by the students will be checked for *validity*. The validity is based on the state of one or more elements in conjunction with coupled constraints.

Depending whether the action is valid, i.e. all constraints are valid and the state is as intended, then a corresponding *reaction* will be caused. The reaction will perform the main intention of the student. For instance when the student tries to create a link between two atom, a corresponding action is performed. It will draw the line that is following the student's finger and upon release, it will internally check the state of the atom and its valency. Depending on the success of the test, it will cause a reaction that will finally establish a chemical bond between the two atoms by drawing the line, performing some animation and sound effects, updating the states, creating a composite out of the atoms and inserting a new edge into the graph. Reactions can trigger other reactions as well, either in parallel or sequential execution. For instance, a reaction about creating a molecule out of two atoms can cause a different reaction that handles the composition of both atoms. This is relevant because any of the two atoms might have been in a composite before the new chemical bond. If both were in a composite, then a new *composite-reaction* would handle the *merge* of the two composite data-structures.

All UML classes except for *State* and *Composite* are represented as *abstract* classes. The meta-model tackles three main objectives. First and foremost it is graph-oriented due to providing logical links between components. This very abstract construct is the main building block of our approach. Secondly, the model offers four important relations, namely *solutions*, *templates*, *patterns*, and *tools* which are features that were not yet considered in current GSRS so far. In combination with a graph that the students create during gameplay, these features are very simple to integrate because all four parts are forming a graph as well. Last, it provides a guide for developers to make the most of *reusability*. Since all elements are based on a *common protocol*, actions, reactions and constraints can be shared among different activities. The model thus provides a loosely coupled design by outsourcing the main concepts.

3.2 Use-Cases

In this section, we will present some concrete use-cases of activities that we created using our meta-model. We will show a conceptual high level view using UML class diagrams. The following use-cases cover activities in different domains that we intentionally tackled to demonstrate the applicability of our work. The fields cover chemistry, computer science and chemical biology. The first three out of four presented activities were implemented, whereas the last activity in this chapter will only be presented conceptually.

Concerning all sub-models, we have removed the activity component with its four relations because it is static for all sub-models. Moreover, we are using `SpriteKit`¹ as our graphics framework and therefore we replaced the abstract `View` component from the meta-model in all sub-models with the highest abstraction of a graphical entity in `SpriteKit`, which is `SKNode`.

3.2.1 Molecule Activity

The molecule activity, that we described in chapter 1, were the first idea of an activity that we came up, perhaps because its structure is the most similar to a graph. Nodes and edges are represented by atoms and chemical bonds, respectively. Also, an atom maintains a symbolic label to describe its value which is equivalent for labelled nodes in an ordinary graph.

The idea of this activity is held very simplistic. The students need to combine atoms using lines to create molecules. Atoms and molecules can be dragged around and bonds between two atoms can be deleted and created. Since GSRs are not meant to be used as an exam platform but rather to help students in their learning process using gamification, we have added the valency of an atom below its label. This information reminds the students about how many bonds an atom can have. Therefore, atoms with a zero valency such as helium cannot have a link, thus whenever the student tries to create one, the action will not work. The sub-model for the molecule activity produced from our meta-model is presented in figure 3.3 and some concrete screenshots of the GUI are depicted in the figures 2 – 5 in part A of the appendix. Furthermore, screenshots of an example of usage for the pattern relationship are included in the figures 6 – 8 in part A of the appendix. In video 1 from part B of the appendix, a molecule-dedicated gameplay is shown.

¹<https://developer.apple.com/spritekit/>

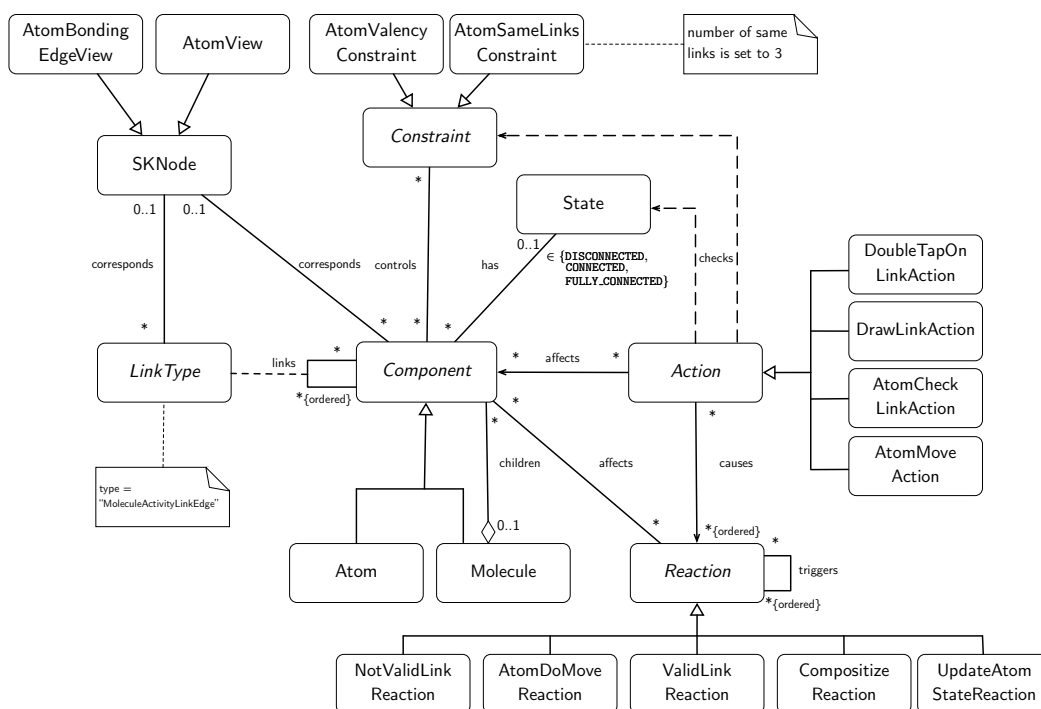


Figure 3.3: Molecule Activity – Sub-Model created from our Meta-Model

In the core of the model, the composite has been replaced with *Molecule* and the abstract *Element* component is now represented by a concrete *Atom* class. Molecules are containers for atoms and are created using at least two atoms. Both atoms and links have their corresponding view, *AtomView* and *AtomBondingEdgeView*, respectively. Two constraints are used for restricting an atom in terms of the number of links. An atom cannot have more bonds than its valency allows, and we restricted the number of *same* links to three for two reasons. Firstly, more than three links between two nodes were not realistic in real world scenarios and secondly, a visual upperbound is important in order to not unnecessarily overflow the view and negatively influence the user experience. Three states have been introduced to provide information about the bonds of an atom, i.e. whether an atom is connected or has reached its maximal number of bonds. Moreover, there exists only one type of edge between components. The type is a string and should be unique among the different activities because we are working with a common meta-model, thus a single *LinkType* table in a database will be shared among all concrete activities.

Several actions and reactions are deployed. Former are used to handle user interactions such as allowing the student to delete a chemical bond by double tapping (*DoubleTapOnLinkAction*), drawing a link while sliding with the finger (*DrawLinkAction*), checking the constraints when trying to combine two atoms using a link (*AtomCheckLinkAction*) or simply for moving a molecule (*AtomMove*

Action).

Latter are used to perform some of the intended actions. For instance *ValidLinkReaction* and *NotValidLinkReaction* are executed depending on the constraints verified in *AtomCheckLinkAction*. If the link is valid, meaning that both constrains *AtomValencyConstraint* and *AtomSameLinksConstraint* do not fail, the former reaction creates a composite out of the atoms or merges two composites by invoking a *CompositizeReaction*. The latter action removes the temporary line that the student has drawn. Depending on the action, the states of atoms are updated in the *UpdateAtomStateReaction* component.

An example of a graph resulting from the sub-model is depicted in figure 3.4 with the nodes being labelled as "*X : Y*" where *X* is the type of atom and *Y* is the concrete class from the sub-model. Furthermore, all three elements or nodes are within a composite construct.

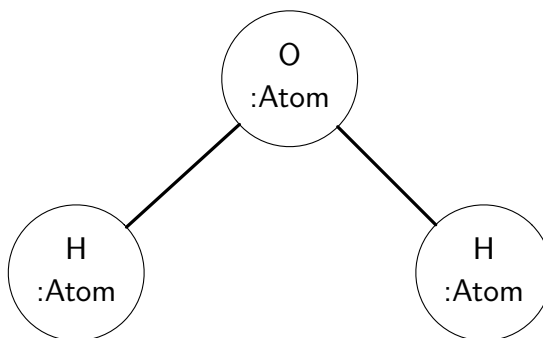


Figure 3.4: Molecule Activity – Example of a graph resulting from building a water molecule. The edges are unordered and undirected. Moreover, their type is "MoleculeActivityLinkEdge". All elements are within the same composite data-structure.

3.2.2 Algorithmic–Block Activity

The *Algorithmic–Block* or *Algo–Block* activity is based on the programming example from chapter 2.2. The main idea is about enabling students to drag-and-drop statements provided through the tools relationship in order to create a *control flow*. The tools or statements are *inexhaustible*.

The provided statements in our activity are abstracted away, i.e. *methods*, *procedures* or *functions* are grouped as *actions*. The activity is not meant to be used for syntax verification, nor to be compilable, although the latter could be integrated on top of our work. Syntactic rules are inherent to help students to focus on the problem solving tasks. For instance, similar to the molecule activity where students cannot create a link with a helium atom because of its zero valency, in this activity students cannot put non-conditional statements into the condition of

a loop. Currently the activity supports an *If*-conditionals, a *While*-loops, boolean conditions and actions. We shall point out that the activity is certainly *extendible*, but as a proof of concept we did not consider all possibilities such as supporting different loops or conditionals and therefore, the sub-model is *incomplete*.

The sub-model for the described activity produced from our meta-model is presented in figure 3.5 and some concrete screenshots of the GUI are depicted in the figures 9 – 16 in part A of the appendix. In video 2 from part B of the appendix, a gameplay of an Algo-Block activity is shown.

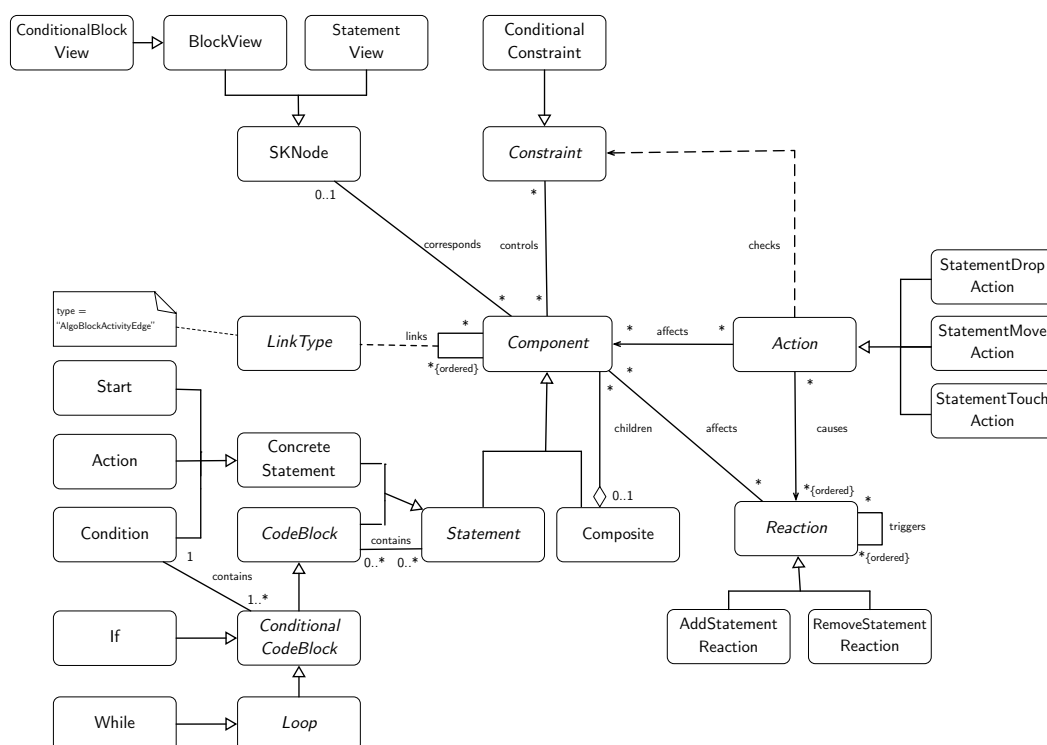


Figure 3.5: Algo-Block Activity – Sub-Model created from our Meta-Model

In the core of the model, the abstract *Element* component is now represented by an abstract *Statement* entity. In contrast to the molecule activity, we have not used the composite component in this sub-model to demonstrate its optional use. We have achieved the same intention by introducing *CodeBlock*-statements that are an abstract notion of a group of statements together with *directed* and *ordered* edges to represent the workflow of the statements.

The inheritance hierarchy from the abstract *Statement* entity is designed to support single statements as well as a group of statements within a block. Programming concepts such as loops and conditionals contain a condition, therefore both conform to the *ConditionalCodeBlock* which in turn conforms to *CodeBlock*. Similar to the molecule activity, some UI elements based on *SKNode* are intro-

duced, and actions mainly handle user interactions such as touches on statements or simply moving elements around. Depending on the *ConditionalConstraint* that is used to check whether a student tries to add a non-condition element as a condition to a *ConditionalCodeBlock* entity, the action invokes an *AddStatementReaction* that performs the needed tasks to append an element to the list of the control-flow or to a *CodeBlock* component.

An additional *Start* statement is introduced to make the graph *connected*. The top-most statements added by the student are ordered, but the graph specifies ordered edges. We were therefore forced to maintain a *root* node that points to the top-most statements with ordered edges. This root node does not have a view nor can it be manipulated by the student. It represents a *static* entity which is part of every graph. This process is a *compromise* of our approach, and in a sense it introduces an *overhead*, which we, however, believe is negligible due to the main benefits of our method.

An example of a graph resulting from the sub-model is depicted figure 3.6.

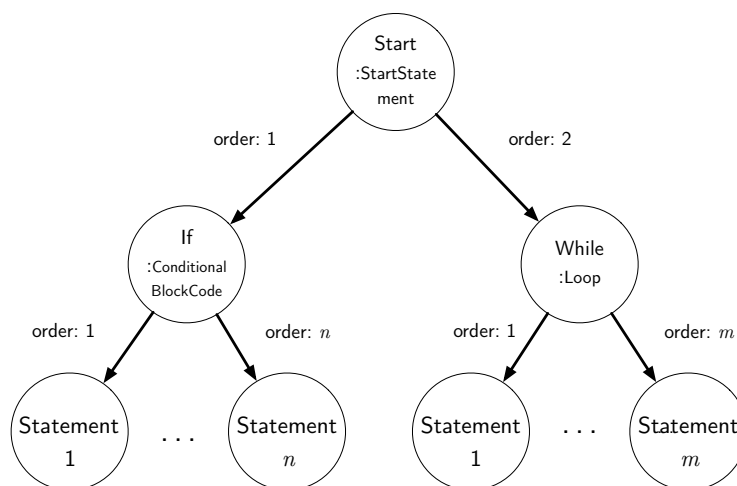


Figure 3.6: Algo-Block Activity – Example of a graph resulting from a control flow where the edges are ordered and directed.

3.2.3 Blood Count Analysis Activity

The third and last *implemented* activity that we call *Blood Count Analysis (BCA) Activity* is based on chemical biology, which is to the best of our knowledge not implemented in any other (G)SRS. The main idea is to interrogate students about *blood counts* or other vitals such as blood pressure. The teacher specifies some counts provided through a template that needs to be adjusted using *health items* which are given through tools. Each health item has a positive or negative influence on a subset of counts. The students need to drag-and-drop health items

from a health bag into an anatomical view of a human which in turn activates the influence process on certain counts. The items influence the counts on a timed-interval to simulate real-world waiting times. For instance, increasing low blood pressure with caffeine as an item can be achieved much faster than increasing the iron in the blood with iron supplements. Moreover, the time-based effects are not *inexhaustible*. The students need to pay attention to not exceed the *range* of a count, which they need to know by heart. After a certain amount of time, the students have to put the health item back to the health bag and add it again to the human anatomy to reactivate the procedure. This is done on purpose to introduce an additional level of time pressure.

The activity is meant to control the knowledge of students in different ways. Firstly, the students can be checked whether they know the biological abbreviations by heart. Next, they have to know the possible *range* of a count. Last but not least, the students must also know about the influences of health items that are used to increase or decrease the counts.

The sub-model of the BCA activity created from our meta-model is presented in figure 3.5 and some concrete screenshots of the GUI are depicted in the figures 17 – 20 in part A of the appendix. In video 3 from part B of the appendix, a gameplay of a BCA activity is presented.

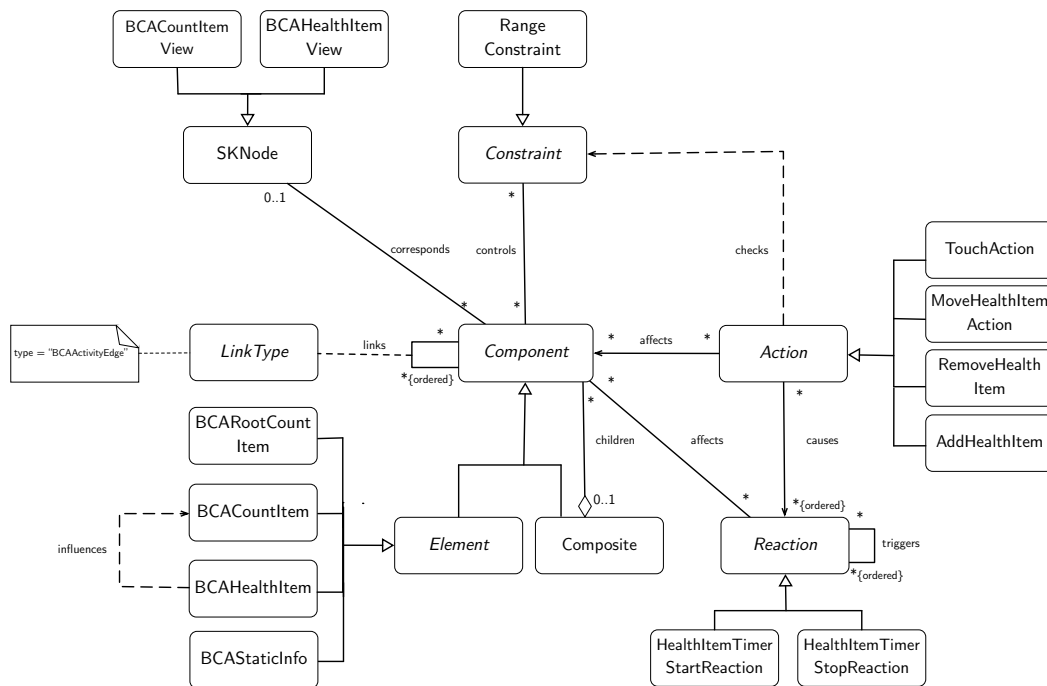


Figure 3.7: Blood Count Analysis Activity – Sub-Model created from our Meta-Model

For this type of activity, the composite design pattern is not needed since the

task is about modifying elements provided by the teacher rather than maintaining a hierarchy of components. The elements are hence *static*. The discussed count and health items are conforming to *Element*. Furthermore, the teacher can specify some additional information such as the age or gender of a person. Health items influence count items, either positively or negatively, which is denoted using a *dependency* relation. Similar to the previous algorithmic activity, we need to introduce a designated node that is also conforming to *Element* to imply an *order* on the edges in order to maintain a connected graph.

For this type of activity, we have implemented a single constraint that is used to ignore influences when they are about to exceed a pre-defined range of a count item. The provided actions handle user interactions such as moving elements or adding or removing items from the health bag. Depending on where the items are placed, either in the health bag or the human anatomy, a dedicated reaction is triggered to respectively stop or start the influences.

An example of a graph resulting from this sub-model is depicted figure 3.8.

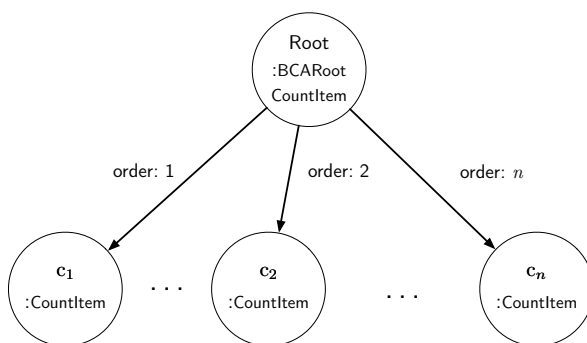


Figure 3.8: BCA Activity – Example of a graph produced from the corresponding sub-model composed of ordered and directed edges.

3.2.4 UML–Class Diagram Activity

The last activity is designed for modelling purposes in computer science classes. The activity provides students with elements from standard UML class diagrams. Similar to the molecule activity, students need to combine UML classes using typed edges such as inheritance, association, etc., and drag-and-drop UML methods or attributes into the classes. This type of activity can be used for training important OOP concepts such as polymorphism that is usually done via programming. However, we believe that such an activity is better suited for this kind of tasks because it leverages from the difficulties of programming languages such as syntax and rather concentrates on the main task. The sub-model as presented in figure 3.9 is by no means complete, i.e. does not support all features of a standard UML class diagram. Many concepts such as *keywords* (static, public,

etc.) or *method and attribute* types are abstracted away for simplification reasons, which can, however, be integrated in a later stage. Due to time constraints, we were not able to implement this activity. Nevertheless, we wanted to show its sub-model due to a specific reason. In contrast to all sub-models so far, this type of model supports more than one type of link. The evaluation of such an activity is slightly different, since UML relations are a mix of ordered and unordered edges, but graphs cannot support both at the same time. We elaborate on this specific issue in section 3.3.

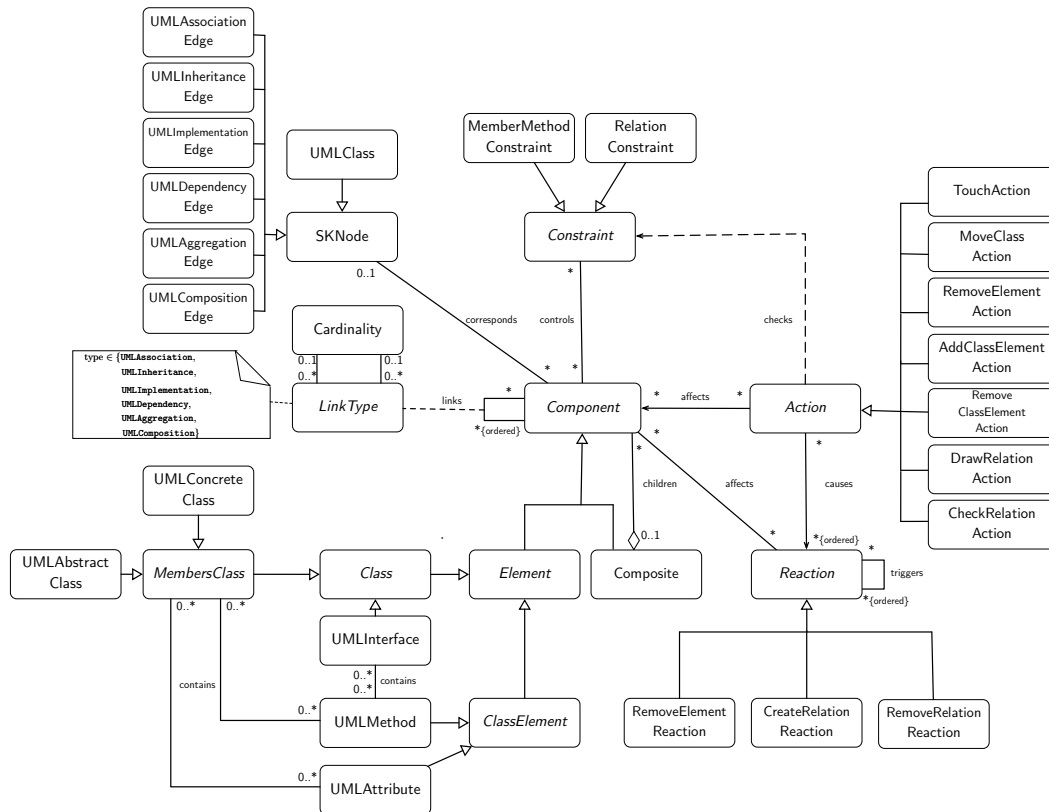


Figure 3.9: UML Class Diagram Activity – Sub-Model created from our Meta-Model

The components coming from *Element* are defining a hierarchy of inheritance. *Interfaces* cannot contain *attributes* or *members* but rather only *methods*, and concrete and abstract classes can incorporate both, methods and members. The UML standard supports six different types of *relations*, that are each realized as a type of link. Links are extended so that they can support *cardinalities* on both endings, which is the case for *association* relations. Each type of link is associated with a corresponding view since the visualisation differs from edge to edge.

MemberMethodConstraint and *RelationConstraint* are proposed in this context. Former is used to make sure that attributes and methods are right-placed into a class entity while drag-and-dropping by the student. Latter is introduced to

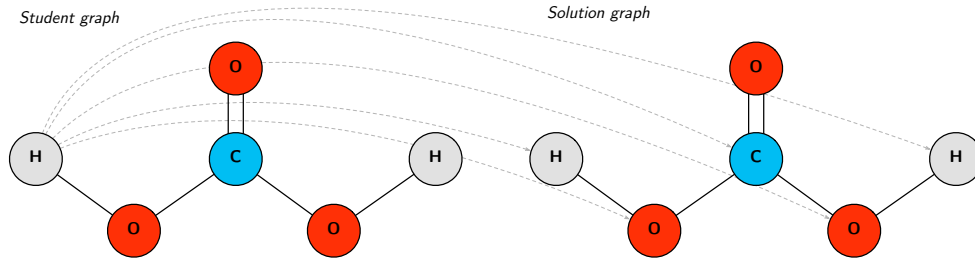
disallow some relations. For instance, when a student tries to use inheritance on an interface, then this constraint will fail, because an *UMLImplementationEdge* should be used instead. Although we could have omitted such a constraint and let the student produce errors, we, however, believe that activities should help students in their learning process. The effect is similar to the molecule activity where the valency is shown below a molecule, and constraints make sure that zero-valency atoms cannot be connected by the student. This is done on purpose to remind the students on some facts.

Actions are mainly used for user interactions and check the constraints for validity before actually triggering any reaction. Reactions are taking care of data-structures, i.e. deleting or creating additional elements such as relations between two specific classes or adding or removing some methods or attributes.

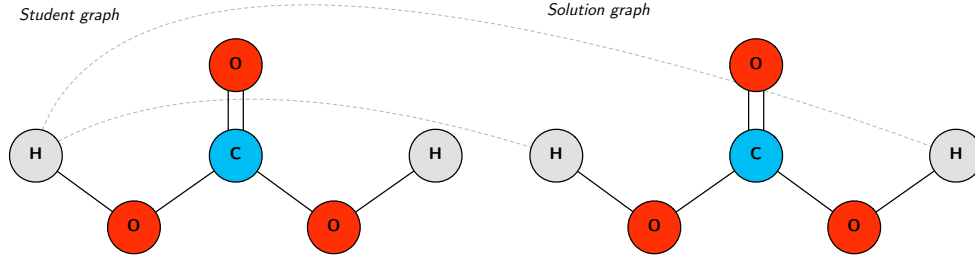
3.3 Graph Comparison

The graph comparison between the students' answers and the solution provided by the teacher is an essential part of our work. We need to compare graphs for evaluation purposes which leads to the well-known open computational *graph isomorphism* problem. The problem is about determining whether two graphs are *isomorphic*, that is, whether there is a *bijective mapping* from the vertices of one graph to the vertices of another graph such that the edge adjacencies are preserved. However, the complexity of such a comparison is not known to be feasible in polynomial time, and no efficient algorithm currently exists [21]. As a consequence, *brute force* algorithms are proposed. However, run-times of these techniques exponentially increase with the size of the input graphs, restricting the applicability of such methods to rather small graphs. We however articulate the view that *scalability* in the realm of GSRs is rather less important because activities are simply not meant to contain an enormous amount of elements. Yet alone from a UI perspective, it would be difficult to fit a significant number of nodes within a screen of a size of a laptop, smartphone or tablet. We have nevertheless benchmarked the comparison algorithm, and we elaborate on the test runs in more detail in section 3.3.1.

Standard isomorphism algorithms try to substitute each node with each other because there is no prior knowledge about the input graphs. However, in our use-cases, we always need to compare two graphs of the same *type*. This prior knowledge gives us an opportunity to skip unnecessary comparisons between unrelated nodes. To illustrate this, let us consider the simple *carbonic acid* compound in figure 3.10.



(a) Graph Isomorphism Example with the standard approach on a H_2CO_3 -molecule when substituting the hydrogen atom.



(b) Graph Isomorphism example with our approach on a H_2CO_3 -molecule when substituting the hydrogen atom.

Figure 3.10: Graph Isomorphism approaches. The dotted arrows represent a substitution between two nodes.

Due to the lack of prior knowledge of the type of graph, traditional isomorphism algorithms would compare an hydrogen atom with any other atom from the solution graph. This process is then repeated for all nodes, resulting in

$$\binom{n}{2} = \binom{6}{2} = 15$$

substitution comparisons where n is the number of atoms as depicted in figure 3.10a. In our approach, however, we only compare semantically equivalent nodes, resulting in

$$\sum_{s \in S} \binom{|s|}{2} = \binom{2}{2} + \binom{3}{2} + \binom{1}{2} = 1 + 3 + 0 = 4$$

substitution comparisons where $S = \{\{H_0, H_1\}, \{O_0, O_1, O_2\}, \{C_0\}\}$ and A_k denotes an atom with an unique identifier k for distinction purposes.

There are some *heuristics* that can be used for speeding up the decision about isomorphism. Generally, the *order*, *size*, *vertex degree*, etc., are taken into consideration to determine whether two graphs are *not* isomorphic because these can be verified relatively fast. However, what we really want to achieve is to not reject a student's attempt but rather evaluate his graph, although not being correct. Even more, we would like to tell the students exactly what they did wrong, or what they did forget. Therefore, such heuristics are unfortunately not an option.

Furthermore, nodes are represented by activity elements, and have more than just a label. They have attributes and concrete values that need to be verified as well. We need thus to go *beyond* isomorphism, that is, not only comparing the graph structure, but also each semantically equivalent pair of nodes for value accordance. The problem of only comparing an isomorphism is illustrated in figure 3.11.

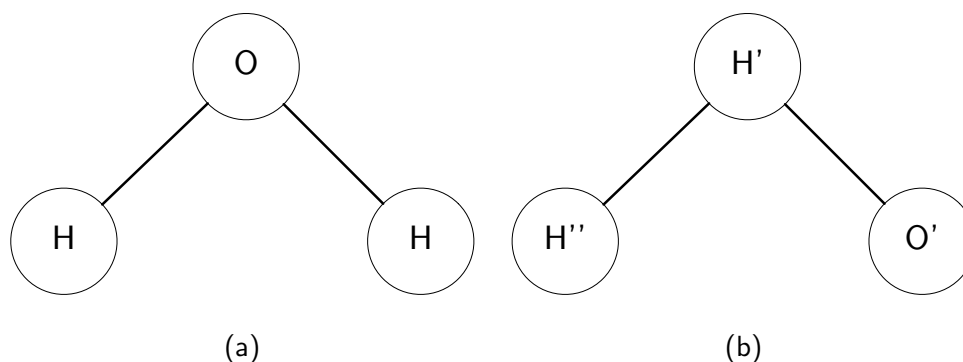


Figure 3.11: A typical graph isomorphism algorithm would find an isomorphism between (a) and (b) since the graph structure is preserved, and a bijection can be produced to attain (b) from (a), which is of course not correct since nodes represent real world activity elements and thus have a meaning.

In figure 3.12, a simplified UML class diagram for the graph modelling is presented that our evaluation algorithm 1 is based on.

The *Element* abstract class in the bottom right represents the same *Element* component from the meta-model. The *Node* or more specifically *EvaluationNode* has been introduced as a *wrapper* for the elements, and to separate the elements from evaluation *annotations* such as fitness scores. The same is being done with edges. The two first methods in the *Element* signature are very important because they influence the evaluation algorithm. For instance, in our molecule example, two atoms are semantically equal if they are of the same type. The *GraphProtocol* represents a graph in an abstract form where both methods are implemented differently depending on the direction of a graph. Undirected graphs ignore the *direction* whereas directed graphs do not. Both graphs conform to *EvaluationGraphProtocol* and implement several procedures to perform the evaluation, while respecting the direction that depends on the sub-models of concrete activities. Each concrete activity implements a strategy for the graph direction, that is later used by the *GraphEvaluator* to create an instance of a concrete graph in order to perform the evaluation.

For the following algorithm, let us use figure 3.13 as a basis.

3.3. GRAPH COMPARISON

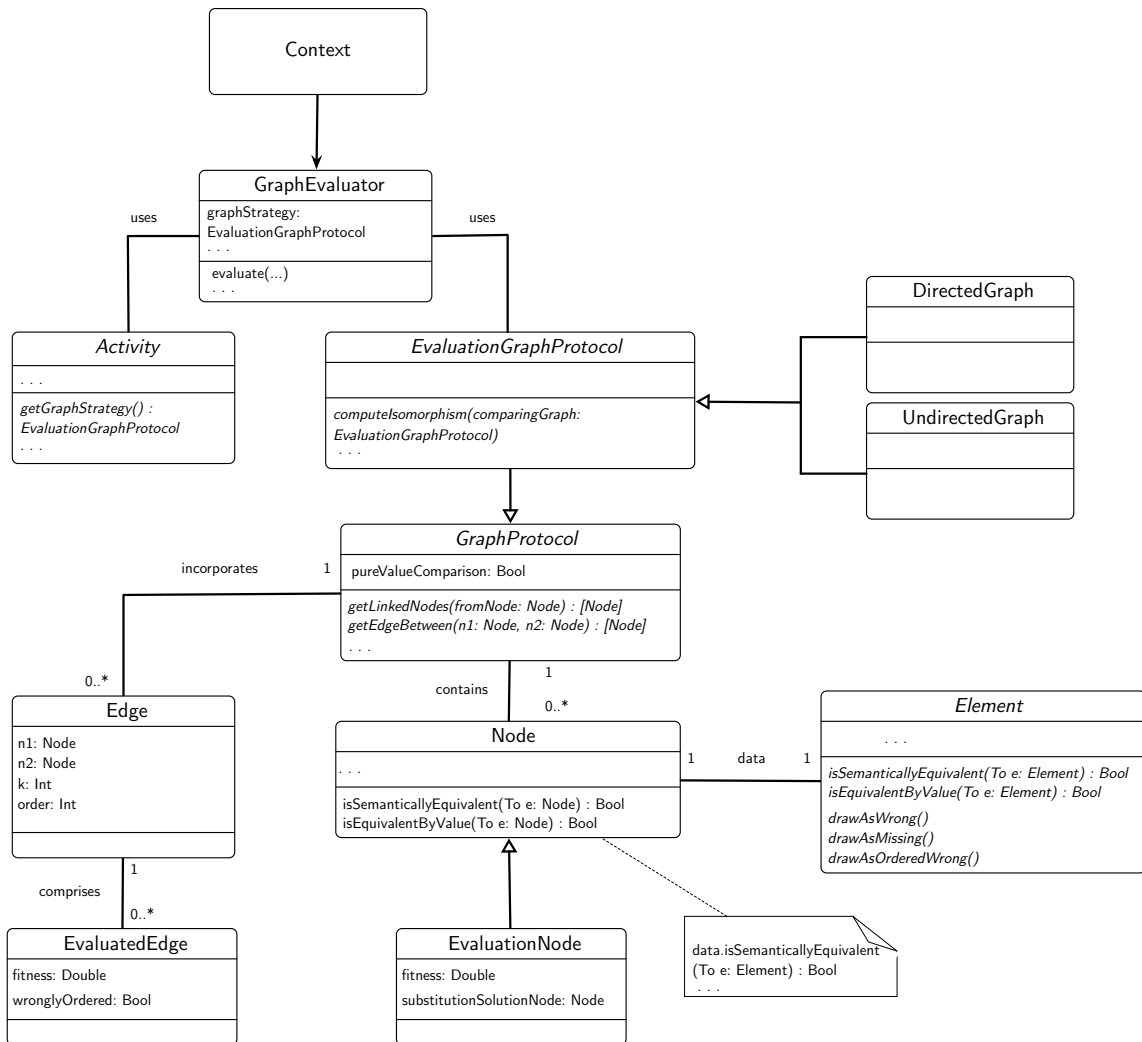


Figure 3.12: Simplified UML Class Diagram for the Graph Implementation

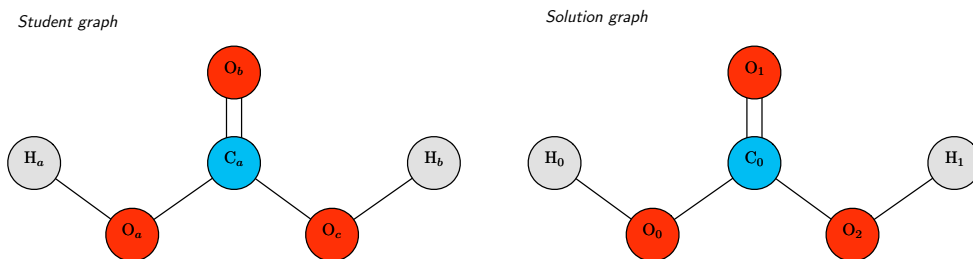


Figure 3.13: Concrete use-case of a graph produced by a molecule activity. The nodes are enumerated only for identification purposes.

Algorithm 1 Algorithm for finding the best isomorphism

```

1: procedure Isomorphism( $\mathcal{G}$ )                                ▷ The solution Graph  $\mathcal{G}$ 
2:    $\mathcal{S} \leftarrow \text{createSubstitutions}(\mathcal{G})$ 
3:    $\mathcal{P} \leftarrow \text{createPermutationGroups}(\mathcal{S})$ 
4:    $\mathcal{A} \leftarrow \text{computePermutationAlternatives}(\mathcal{P})$ 
5:    $a_0 \leftarrow \mathcal{A}.\text{first}$ 
6:    $f_0 \leftarrow 0.0$ 
7:   for  $a \in \mathcal{A}$  do
8:      $i \leftarrow \text{computeAndRateIsomorphism}(a, \mathcal{G})$ 
9:      $f \leftarrow \text{computeFitnessScore}(i)$ 
10:    if  $f > f_0$  then
11:       $f_0 \leftarrow f$ 
12:       $a_0 \leftarrow a$ 
13:      if  $f_0 == 1.0$  then
14:        break                                              ▷ 100% match found
15:      end if
16:    end if
17:  end for
18:  return  $a_0$                                               ▷ The best alternative
19: end procedure
    
```

The algorithm starts by creating a substitution mapping \mathcal{S} by grouping semantically identical elements. Based on figure 3.13, the following substitution mapping will be created:

Solution groups	Student groups
(H_0, H_1)	(H_a, H_b)
(O_0, O_1, O_2)	(O_a, O_b, O_c)
(C_0)	(C_a)

Table 3.1: Concrete substitution mapping for 3.13

The next step involves creating different permutations based on each student group that we store in \mathcal{P} . These will be the following:

Inter-group student nodes permutations
$(H_a, H_b), (H_b, H_a)$
$(O_a, O_b, O_c), (O_a, O_c, O_b), (O_b, O_a, O_c),$ $(O_b, O_c, O_a), (O_c, O_a, O_b), (O_c, O_b, O_a)$
(C_a)

Table 3.2: Inter-group permutations based on the substitutions in 3.13

The permutations are used to create *alternative* student answers. The atoms in each entry from the substitution table 3.1 will be substituted with the atoms in the corresponding index in the student group. Let $A_x \leftrightarrow A_y$ denote a substitution between A_x and A_y . For hydrogen, there are two atoms, hence $2! = 2$ different substitutions, which are $H_0 \leftrightarrow H_a$, respectively $H_1 \leftrightarrow H_b$ and $H_0 \leftrightarrow H_b$, respectively $H_1 \leftrightarrow H_a$. For oxygen, there are $3! = 6$ different permutations possible. When a student forgets an atom, the inter-group permutation will simply put an empty place-holder to mark the atom as missing. For example, assume the student did forget atom O_c . Then the following permutations would have been created:

Inter-group permutations
$(H_a, H_b), (H_b, H_a)$
$(O_a, O_b, \emptyset), (O_a, \emptyset, O_b), (O_b, O_a, \emptyset),$ $(O_b, \emptyset, O_a), (\emptyset, O_a, O_b), (\emptyset, O_b, O_a)$
(C_a)

Table 3.3: Inter-group permutations based on the substitutions in 3.13 omitting O_c .

In the other hand, additional non-required nodes will be ignored. The third step involves the construction of \mathcal{A} , which is the set of different *alternatives*. These are a combinatorial enumeration of the individual permutations, each representing a possible solution as shown in the figure 3.4.

#	Alternatives permutations
1	(H_a, H_b)
	(O_a, O_b, O_c)
	(C_a)
2	(H_b, H_a)
	(O_a, O_b, O_c)
	(C_a)
3	(H_a, H_b)
	(O_b, O_a, O_c)
	(C_a)
4	(H_a, H_b)
	(O_b, O_c, O_d)
	(C_a)
...	...
12	(H_b, H_a)
	(O_c, O_b, O_a)
	(C_a)

Table 3.4: Combinatorial permutations of possible solutions.

There will be 12 alternative permutations since $2! \cdot 3! \cdot 1! = 2 \cdot 6 \cdot 1 = 12$ where each x_i represents the number of semantically same atoms or nodes. Next, for each entry, the nodes at each index from the solutions will be substituted with the elements at the same index as illustrated below in figure 3.14.

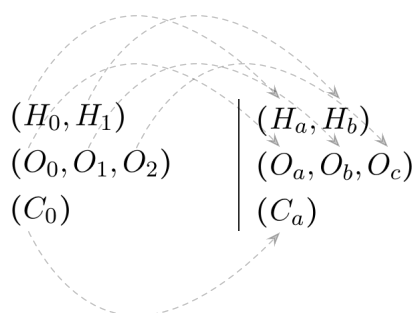


Figure 3.14: Example of substitutions between solution and student nodes.

The last step covers the iteration of all alternative permutations and the creation of a new graph based on the substitutions. The graph will be then evaluated using a *fitness* score. We elaborate on the evaluation in section 3.4. When a fitness score of 1 is found, the algorithm will stop. Otherwise, the best alternative

so far will be picked.

The presented approach works for a single type of link, i.e., for a graph that is either directed or undirected. For the UML Class activity where we support multiple edges of different directions, we simply run the algorithm multiple times with different sub-graphs as input based on a *divide & conquer* approach as illustrated in figure 3.15.

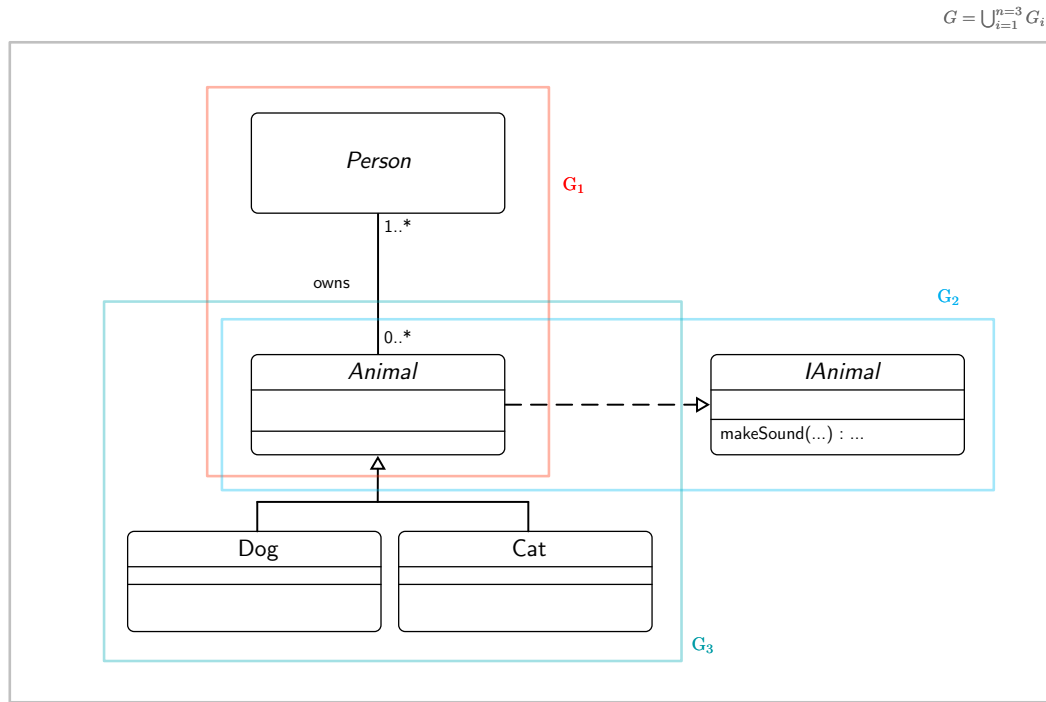


Figure 3.15: UML Class Activity – Divide & Conquer approach is illustrated using different colors representing each a different type of relation. Each rectangle will represent a sub-graph G_i of the whole input graph G .

For instance, we can see that three different relations are used in this example. *Inheritance* and *Implementation* edges are directed whereas *Association* edges are undirected. The graph will be split into three parts, each evaluated separately. At the end, the scores are accumulated considering all sub-graphs. Moreover, the complexity is being reduced due to the split.

Our approach covers the first research question **RQ1** about whether the evaluation can be generalized for all activities. It also partially covers **RQ2**, since this is a main advantage compared to non graph-based solutions. We have achieved this by solving the graph isomorphism problem using the described algorithm 1 where we mainly consider the graph structure rather than a naive value comparison. All elements conform to a common protocol where all must implement dedicated methods that define which elements are considered to be semantically equal. This enables us to abstract from concrete types, hence to create a generic solution.

3.3.1 Benchmark

Although we argue that our approach is not meant to be scalable in the context of GSRS, we have nevertheless benchmarked our method. We have chosen the molecule activity for the test runs. The molecule activity is the candidate with the probably highest number of semantically equivalent nodes as they are required by many chemical compounds. To demonstrate the impact of substitutions between nodes, we have limited the benchmark with three atoms where we gradually increase the number of each atom. For each setup, we have performed a simulation study consisting of 30 independent runs to collect estimated means. The results are shown in part C of the appendix. We are measuring the number of iterations and the time elapsed for running algorithm 1. An iteration is a comparison between two random graphs based on the configuration, i.e., the solution graph and an alternative substitution of the student's graph. Although the time elapsed for a comparison varies between devices, and the iteration number is a better metric, we have still put it as a reference. In general, we can observe what was to expect. The higher the number of atoms, the higher the mean values. However, the impact on the performance is highly dependent on the number of atoms *within* a group because this implies many combinatorial permutations. Note that we have tested the performance locally on an iOS device. However, the results could be further improved using parallelization options when outsourcing the algorithm to a dedicated server.

3.4 Student Evaluation

The student evaluation is an essential part of our work, and is based on the graph comparison algorithm. The point distribution is computed in line 8 in algorithm 1 by rating an alternative graph substitution. The nodes and edges of the graph are being decorated using a fitness score between 0 and 1 as shown in figure 3.12. Furthermore, the fitness value of an edge can be 1.0 but its order may be wrong. Therefore the edges contain an additional boolean value indicating whether the order is wrong. The points are being distributed in a rather *naïve* way. In figure 3.16 a student alternative graph G' with its corresponding solution graph G are depicted where k and o represent the number of links between two nodes and the index of the order of an edge from a sequence of edges, respectively.

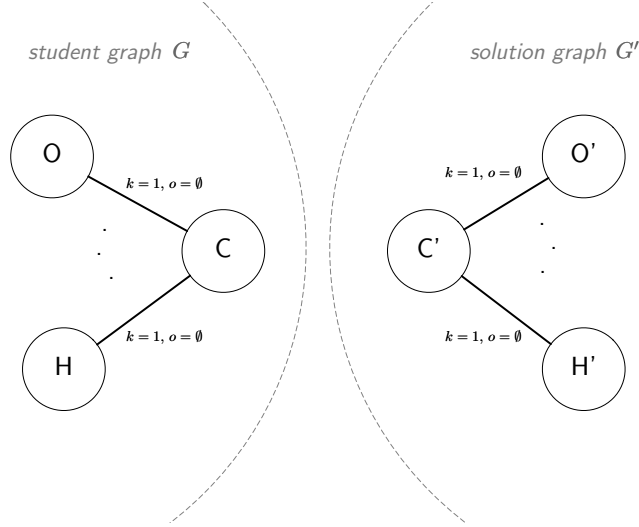


Figure 3.16: Student Evaluation – Substitution between two graphs.

Let us assume the substitutions have been found, i.e., $O \leftrightarrow O'$, $C \leftrightarrow C'$, etc., and let N and N' define the set of nodes in G and G' respectively. Moreover, let E and E' define the set of edges in G and G' , respectively, and let s_x denote the substitution of x , where $x \in \{n, e\}$, $n \in N$, $s_n \in N'$, $e \in E$ and $s_e \in E'$. Then,

$$\begin{aligned} \forall n \in N \mid n =_v s_n &\implies c_{n_i} = c_{n_i} + 0.5 \\ \forall e_n \in E \mid e_n.k = s_{e_n}.k &\implies c_{n_i} = c_{n_i} + 0.25 \\ \forall e_n \in E \mid e_n.o = s_{e_n}.o &\implies c_{n_i} = c_{n_i} + 0.25 \end{aligned}$$

where e_n is an edge outgoing from n , c_n is a counter for n ranging from 0 to 1 for each comparison between two nodes n and s_n , and $x =_v y$ is a value comparison between x and y . Finally, the fitness is computed as

$$f_n = \frac{\sum_{i=0} c_{n_i}}{\max(l_n, l_{s_n})}$$

where l_x represents the number of links outgoing from x .

In general, we are counting the number of correct matches between any two nodes that are considered as a substitution. We are distributing a score for each node and we give the most importance to the value comparison by providing 50% of a possible score. The remaining 50% can be obtained depending on whether the student has used the same number of links k between two nodes and whether the order index o is the same between two edge substitutions. This process will be then repeated for each linked element from a given node. Finally, the score of a node will be computed by counting how many *sub-points* between 0 and 1

are collected. This number will be then divided by either the number of links of the current node or its substitution. A maximum function is used to penalize the student when he has used more or less links than needed. This approach is based on both a value comparison between nodes and the verification of presence/absence of an edge.

In the BCA activity, however, we have noticed that the edges are *not dynamic*, hence not changing. The main task is to adjust attributes of elements while the logical links are not modified. We have noticed a very high score most of the time, although containing a lot of errors because the student evaluation is based 50% of the time on the correctness of the edges, that are static in this kind of activity, resulting in a 100% match. We were forced to handle such situations, and have provided a boolean flag indicating if a pure value comparison should be considered as shown in figure 3.12. Per default, this flag is not being set. However, if it is set, the fitness score of a node is then computed as

$$f_n = \begin{cases} 1, & \text{if } n =_v s_n \\ 0, & \text{otherwise} \end{cases}$$

ignoring any verification of edges, since they are assumed to be the same.

We have also wanted to show students the error location. Therefore, we have introduced some *error indications* using different colours based on the *Element* protocol as depicted in figure 3.12. Each element must implement three dedicated methods to manipulate their view according to whether an element is determined to be missing, wrong or wrongly ordered. In figure 15 and 16 in part A of the appendix, the evaluation colouring is shown for an instance of an Algorithmic Block activity. The colouring can also be seen in videos 1 – 3 from part B of the appendix.

3.5 Graph Properties & Optimization

We have build our graph evaluation algorithm without *prior* knowledge about the *nature* of the input graphs. We have, however, observed that the graphs produced by an *Algorithmic Block Activity* and a *BCA Activity* are *trees*.

Proof. We will prove the statement using the following definition of a tree taken from [22]:

A connected graph with n vertices is a tree if and only if it has $n - 1$ edges.

3.5. GRAPH PROPERTIES & OPTIMIZATION

Concerning the resulting graph of a BCA activity, every count item will be a node without any outgoing edges. We introduced a *root* node (*BCARoot-CountItem*) to connect all the count items for ordering reasons, and to be graph compliant, hence the resulting graph is *inherently* connected and the count items will consequently be *leaves*. Now, if there are n count items and one root node, then we will have $n + 1$ nodes. For each leaf we introduced a link to the root, resulting in n edges, that is exactly one less than $n + 1$, which is the statement. ■

Now, considering the resulting graphs of an *Algorithmic Block Activity* entity, we have the following implicit properties:

- 1) There exist two types of statements: *Block-Statements* such as conditionals or loops that can contain other statements or *Non-Block-Statements* that cannot enclose any other statement.
- 2) There are no *parallel* links between any two statements. Parallel links are for instance possible in a molecule activity, thus introducing cycles in the graph.
- 3) Every non-*StartStatement* has exactly one parent.
- 4) The root node or *StartStatement* has no parent.

Let L denote the set of "top-most" elements in the algorithm list and let C be the set of children of every block-statement in L obtained by transitivity. Let R be the root node and S denote a statement the student would like to add. Furthermore, let $(S_i \rightarrow S_j)$ denote a directed edge from S_i to S_j which causes S_i to be the parent of S_j .

The student has two possibilities to add S into the algorithm list. Either he adds it to L , hence $(R \rightarrow S)$ or he adds it to another block-statement B where $B \in L$ or $B \in C$, resulting in $(B \rightarrow S)$. Similar to the BCA activity, the root node is introduced to connect all statements in L , thus $\forall L_i \in L \mid (R \rightarrow L_i)$. This implies that the graph is connected.

Now, since the graph is connected and no parallel edges are possible, we know that between any two statements, there is exactly one edge. Let n denote the number of nodes in the graph and let e denote the number of edges. Starting with $n = 1$, only maintaining a root node, there will be no edges, hence $e = 0 \Leftrightarrow e = n - 1$. Every new introduced statement causes another statement to become a parent of it, hence for every new node a new edge is introduced between two vertices which means that if e increases by one, so does n for $n \geq 1$. Therefore, the relationship $e = n - 1 \Leftrightarrow n = e + 1$ is maintained throughout the whole construction of the graph, and we will have $n - 1$ edges, which is the statement. ■

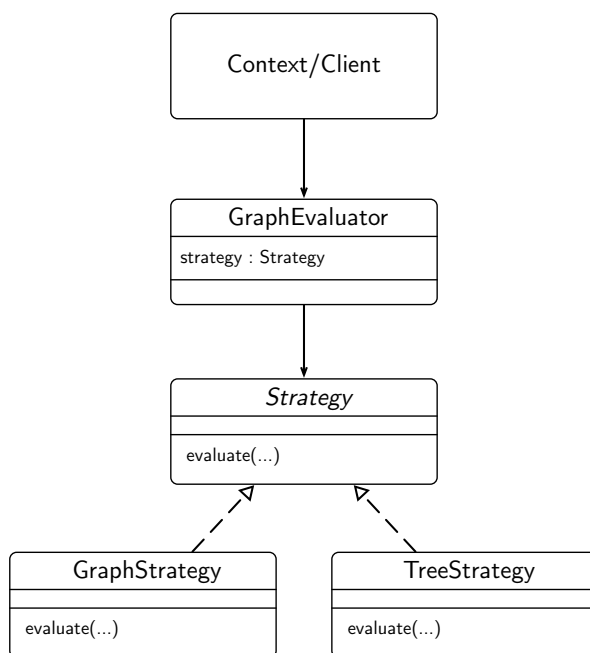


Figure 3.17: Strategy Design Pattern – UML Class Diagram to support multiple evaluation algorithms.

Although we have tried to optimize the brute-force algorithm by only considering semantically equivalent nodes, there is still room to improve knowing that some input graphs are trees. The complexity for solving a *tree-isomorphism* problem is known to be linear [23][24][25][26], which is clearly an optimization option to consider. Due to time constraints we were not able to implement an alternative algorithm for handling tree data-structures. However, in figure 3.17 we present a mechanism based on the *strategy* design pattern [18] on how to integrate an alternative algorithm for the evaluation into our architecture.

The GoF defines the strategy-pattern as follows:

"Strategy defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from the clients that use it".

This extensible design enables a smooth integration of different evaluation algorithms. Note that, before we can decide which algorithm to run, it is required to identify whether the input graph is indeed a tree. This could be done by verifying the absence of cycles in the graph.

Another optimization factor to take into consideration is the implementation of the equivalence comparison of entities of the *Element* class as shown in 3.12. For instance, considering the molecule activity, two atoms are semantically equivalent if they are of the same type. In the other hand, considering the UML class

3.5. GRAPH PROPERTIES & OPTIMIZATION

activity, it is advised to implement the semantic equivalence not only based on the type. In such an activity, the probability of having a lot of concrete classes is high. This will not only cause the resulting graph to increase in size, but also the possible substitutions will increase as well. For better performance, the number of substitutions should be kept as small as possible. An option for this kind of activity would be to compare two concrete classes also for the name. Hence, two concrete class instances would be considered as semantically equivalent if they share the same class name.

4 | Integration of Learning Materials

Contents

4.1	Incorporating learning materials into activities	43
4.2	Linking activities with learning materials	44

In this chapter, we describe how we tackle research question **RQ3**, i.e. about how to establish a *bidirectional mapping* between activities and learning materials so that students can access any learning material related to an activity within a game-session, and play associated activities from learning sources at home in order to foster a continuous learning experience. We believe that the integration of context-based learning materials into activities helps avoiding context switches, that would otherwise interrupt the students' line of thought [27] or even cause an abandonment of the current learning task [28]. Additionally, it helps students to instantly retrieve activity-related materials, which eliminates the need to search from a vast collection of documents, that would require a significant amount of time and might lead to an information overload [29].

4.1 Incorporating learning materials into activities

The integration of learning materials into activities is based on the *Linked Data* principle and is realized through *semantically annotating* activities, nodes and edges of the corresponding solution graphs. In figure 4.1 a UML class diagram is presented to showcase our implementation.

Every element contains a set of human readable terms that can be specified by the teacher. Per term, multiple learning resources can be linked. As a proof of concept, we have limited ourself to web pages and pdf files. In order to enable a smooth support of new learning materials, we have used the *command* design pattern [18].

The GoF defines the command pattern as follows:

"It encapsulates a request as an object, thereby letting you parametrize clients with different requests, queue or log requests, and support undo-able operations."

4.2. LINKING ACTIVITIES WITH LEARNING MATERIALS

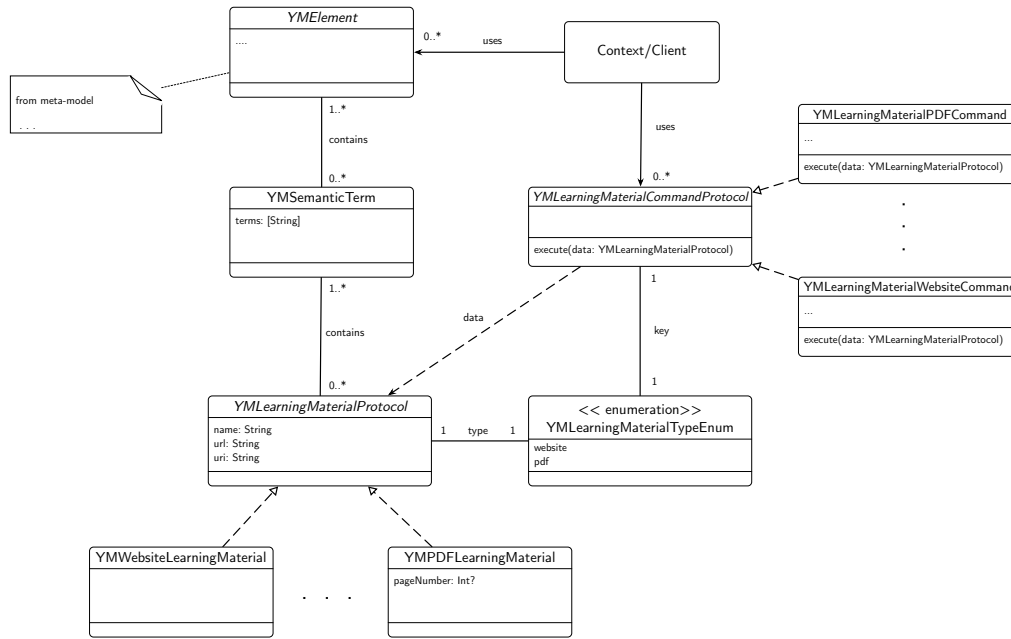


Figure 4.1: Integration of Learning Materials – UML Class Diagram using the Command design pattern.

Each command contains a unique key, which is of the same type as a learning material defined in the *YMLearningMaterialProtocol*. The business-logic for displaying a web page or downloading and showing a pdf is each encapsulated in a different command class. Whenever new types such as video or personal notes are introduced, a new dedicated command can be realized dealing with the task to present the resources to the student. This design enables a loosely coupled system, and provides a relative simple way to support new types of materials.

The *Unique Resource Identifier* (URI) and an *Uniform Resource Locator* (URL) are used to be conform with the Linked Data principle as described by Tim Berners-Lee, the inventor of the *World Wide Web* (WWW) [30]. Using the URL, we can fetch any document from the WWW for further treatment by the individual commands.

Screenshots showcasing the integration of learning materials are depicted in the figures 21 – 25 in part A of the appendix. In video 4 from part B of the appendix, a gameplay showcasing the usage of learning materials is presented.

4.2 Linking activities with learning materials

In the previous section, we have shown how we integrate different types of learning materials into a related activity. In this section, we will discuss the other direction flow, i.e., obtaining activities from, and related to, a learning material.

CHAPTER 4. INTEGRATION OF LEARNING MATERIALS

This enables students to directly switch to our application and play an activity related to a learning resources. However, this would require learning resources or even single pages to be semantically annotated. Therefore, we use the two following technologies to accomplish our goal.

The ALMA (Adaptive Literacy-aware learning Material IntegrAtion) repository contains learning resources such as sideshows or book excerpts and Web resources. where each resource is annotated with programming concepts from the ALMA ontology [31].

The SoLeMiO (Semantic Integration of Learning Material in Office) tool is a Microsoft Office Add-in that is able to recommend and integrate learning material via semantic entity recognition [32]. Using ALMA as the main source of documents, the tool is able to fetch resources as well as concepts and present the retrieved data. The tool is also able to identify concepts from a text, and then request related documents from ALMA.

In figure 4.2, an example of a communication flow between our architecture and SoLeMiO and ALMA is illustrated.

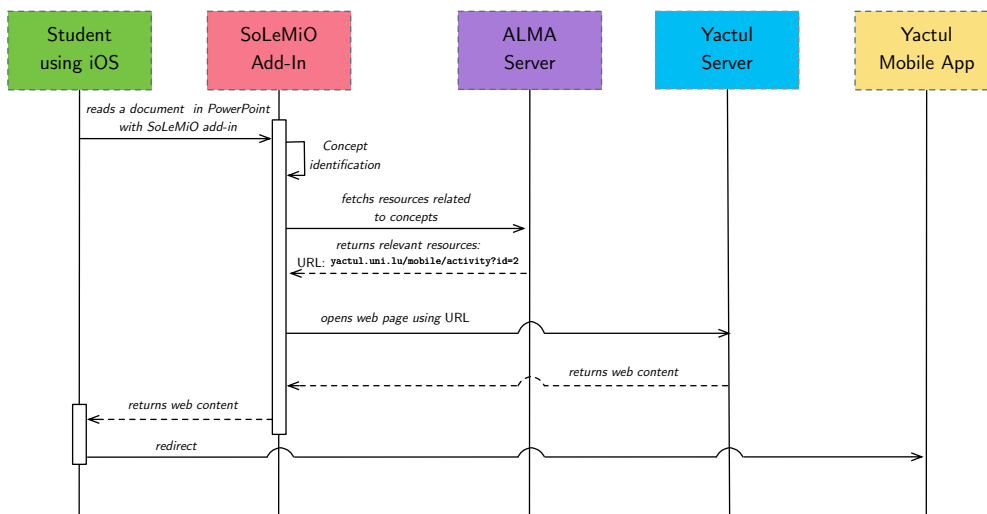


Figure 4.2: Linking activities with learning materials – UML Sequence Diagram illustrating communication flow between different technologies to support opening related activities in documents.

The communication flow starts with the student reading a document in an Office application with the SoLeMiO add-in installed. SoLeMiO will identify the key concepts in the content using state of the art concept recognition and entity linking tools. It will then send a request to the ALMA server to retrieve relevant resources related to the identified concepts. For our use case, activities are web

4.2. LINKING ACTIVITIES WITH LEARNING MATERIALS

resources and are being retrieved through a REST interface using an URL. This information is returned to the add-in by the Yactul server. Yactul Mobile, our proof of concept, and the Yactul server are pre-configured to allow certain URI's to be redirected to our dedicated application. The iOS platform will recognize the specific URL and redirect the student to the local installed mobile application where the activity will be launched. The URL¹ that is specified for redirection purposes contains a *wildcard* to allow any following path with zero or more characters. This redirection feature is realized using *Universal Links*² that allows HTTP URL's to be opened using dedicated apps instead of the browser. Upon redirection, our application then handles the rest, i.e., reading the URI and *query parameters* and creating the activity. Note that the activity may not be stored locally, so the application needs to contact the Yactul server to retrieve missing information.

A screenshot showing that custom HTTP links can be opened using our app can be found in figures 26 and 27 from the part A of the appendix. In videos 5 and 6 from part B of the appendix , the usage of universal links is presented.

¹`yactul.uni.lu/mobile/*`

²`https://developer.apple.com/ios/universal-links/`

5 | Conclusion

5.1 Summary

In this thesis, we propose a graph-oriented approach for modelling activities. The main benefit of using our generic data-model lies in the evaluation of an activity which covers research questions **RQ1** and **RQ2** about generalizing the evaluation and advantages of a graph-oriented data model. Based on graphs, any answer that is semantically equivalent to the solution will be accepted. By solving the well-known computational graph isomorphism problem, the preservation of the graph structure is considered rather than a strict value comparison with unique identifiers as commonly used in today's (G)SRS. This enables teachers to deploy many activity elements of the same type while only defining a solution using a subset of the instances.

We additionally propose new features based on our method which also cover research question **RQ2**. Teachers can define multiple graph-based solutions, resulting in more flexible questions. Inspired by the idea of teaching by templates, teachers can provide a template which is a sub- or superset of a solution graph. Both teachers and students can benefit from this idea. Students obtain a basis or a starting point to work on and teachers are thus able to prepare more complex problems that would otherwise be too time-consuming in traditional game-sessions. Next, we proposed patterns, which are sub-graph of a solution graph. Whenever a student creates a part of the solution, hence a pattern, an intermediate feedback is presented. When all patterns are found, the activity immediately terminated in favour of the student in order to avoid point deductions because of remaining time constraints.

Furthermore, we have established a bidirectional mapping between activities and learning materials as response to research question **RQ3**. Activity-related course resources are integrated to enable a direct access within an activity in order to avoid context switches that could cause an abandonment of the learning process. Based on the ALMA repository and using the SoLeMiO add-in, students can additionally access activities from a learning resource during their learning process. To demonstrate the applicability of our approach in different domains, we have deployed three new activities based on computer science, chemistry and chemical biology using our own dedicated framework. The framework implements our proposed meta-model, handles the evaluation, copes with learning materials associated to an activity and comes with many graphical views to enable a smooth development of activities.

5.2 Future Work

Tree Isomorphism Although we came to the conclusion that scalability is not of utmost importance, we still would like to implement a different algorithm for handling tree data-structures as they may be present in many activities. As discussed in the thesis, tree isomorphism can be solved in linear time which is a clear benefit since the time complexity of our generic solution is exponential.

Point Distribution Weights The current point distribution method is based on a *naïve* approach. It simply counts the number of correct edges and compares the values of two nodes that are connected using an edge. However, we could think of giving nodes and edges some *weights* and then in a final step divide the accumulated score with the sum of the weights. This implies that some nodes and edges are more important and thus better rated. This approach could be useful for instance in an UML activity where the teacher can consider a part of the diagram as more difficult, and could therefore annotate the nodes with a higher weight value to symbolize a *dominance* in terms of point distribution.

Support for more learning materials We would like to support more types of learning materials and thus enable richer content for teachers and students. At the moment, web pages and PDF's are available, but we can think of video material, personal notes or any tabular sheets that could be useful to students.

New activities in more domains Although we have tried to show the applicability of our work in as different domains as possible, we still strive to achieve even more diverse activity scenarios that could be based on our work. For the moment, we are interested in mathematics, and would like to find out what is possible and realistic in this realm in terms of activities.

Port to Server As for our proof of concept, we have chosen the iOS ecosystem as our platform. However, teachers need to create activities in advance, usually from a centralized location. Therefore, we plan to extend our current GSRS system to support the meta-model with our three implemented use-cases.

Port to Android In order to reach as many users as possible, a port to the Android operating system is inevitable and certain. The meta-model is realized using nearly primitive types or classes from our own framework, which makes a port very easy.

Open-Source Last but not least, an additional objective is to make our work available as Open Source to enable a free distribution of the source code to drive innovation.

List of Abbreviations

ALMA	Adaptive Literacy-aware learning Material IntegrAtion
BCA	Blood-Count Analysis
GoF	Gang of Four
GUI	Graphical User Interface
GSRS	Game-based Student Response System
HTTP	Hypertext Transfer Protocol
ITS	Intelligent Tutoring System
OOP	Object–Oriented Programming
REST	Representational State Transfer
RQ	Research Question
SoC	Separation of Concerns
SoLeMiO	Semantic Integration Of LEarning Material In Office
SRS	Student Response System
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPE	Visual Programming Environment

Bibliography

- [1] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From Game Design Elements to Gamefulness: Defining "Gamification"," pp. 9–15, Proceedings of the 15th International Academic MindTrek Conference, 2011.
- [2] C. M. Barrio, M. Muñoz-Organero, and J. S. Soriano, "Can gamification improve the benefits of student response systems in learning? An Experimental Study," pp. 429–438, IEEE Transactions on Emerging Topics in Computing, 2016.
- [3] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, *Active Learning increases Student Performance in Science, Engineering and Mathematics*. Proceedings of the National Academy of Sciences (PNAS), 2014.
- [4] Bunchball, *Gamification 101: An introduction to the use of game dynamics to influence behavior*. Bunchball, Inc., 2010.
- [5] C. Kevin, "A Future Full of Badges," The Chronicle of Higher Education, April 2012.
- [6] L. Hakulinen, T. Auvinen, and A. Korhonen, "The Effect of Achievement Badges on Students' Behaviour: An Empirical Study in a University-Level Computer Science Course," pp. 18–29, International Journal of Emerging Technologies in Learning, 2015.
- [7] C. Grévisse, J. Botev, and S. Rothkugel, "Yactul: An Extensible Game-Based Student Response Framework for Active Learning," XVIII Encuentro Internacional Virtual Educa Colombia, June 2017.
- [8] "Who and what is behind kahoot!." <https://kahoot.uservoice.com/knowledgebase/articles/464890-who-and-what-is-behind-kahoot>. Accessed: 2018-07-28.
- [9] C. Kelleher and P. Randy, "Lowering the Barriers to Programming: A Survey of Programming Environments and Languages for Novice Programmers," pp. 83–137, ACM Comput. Surv, 2005.
- [10] P.-Y. Chao, "Exploring students computational practice, design and performance of problem-solving through a visual programming environment," pp. 202–215, Computers & Education, 2016.

- [11] E. R. Sykes and F. Franek, "An Intelligent Tutoring System Prototype for Learning to Program Java," pp. 78–83, IASTED International Conference on Computers and Advanced Technology in Education including the IASTED International Symposium on Web-Based Education, July 2003.
- [12] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, "Cognitive Tutors: Lessons learned," pp. 167–207, *The Journal of the Learning Sciences*, 1995.
- [13] E. R. Sykes and F. Franek, "Growth and Maturity of Intelligent Tutoring Systems: A Status Report," pp. 100–144, *Smart machines in education*, 2001.
- [14] "Chem Tutor: Learn with Visual Representations!." <https://chem.tutorshop.web.cmu.edu/>. Accessed: 2018-07-05.
- [15] A. Gupta, A. Mittal, A. Karkare, S. Gulwani, A. Tiwari, and R. Majumdar, "Chemistry Studio: An Intelligent Tutoring System," pp. 9–15, *Proceedings of the 15th International Academic MindTrek Conference*, 2011.
- [16] S. Al-Imamy and J. Alizadeh, "On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process," pp. 271–283, *Journal of Information Technology Education*, 2008.
- [17] P. Kelsen, Q. Ma, and C. Glodt, "Models within Models: Taming Model Complexity Using the Sub-model Lattice," pp. 171–185, *Part of the Lecture Notes in Computer Science (LNCS) book series*, 2011.
- [18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 2009.
- [19] M. Bunge, *Causality—The Place of the Causal Principle in Modern Science*. Harvard University Press, London, 1979.
- [20] P. Harris, *Special Relativity*. CreateSpace Independent Publishing Platform, 2014.
- [21] P. Foggia, C. Sansone, and M. Vento, "A Performance Comparison of Five Algorithms for Graph Isomorphism," pp. 188–199, *Proc. 3rd IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, 2001.
- [22] R. Diestel, *Graph Theory*. Springer book – Graduate Texts in Mathematics series., 5 ed., 2016.
- [23] Z. Baida, T. Yuhua, W. Junjie, and X. Shuai, "LD: A Polynomial Time Algorithm for Tree Isomorphism," pp. 2015–2020, *Procedia Engineering*, 2011.

BIBLIOGRAPHY

- [24] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," pp. 31–42, *Journal of the ACM*, 1976.
- [25] G. Valiente, *Algorithms on Trees and Graphs*. Springer – AMC Computing Classification, 3 ed., 1998.
- [26] S. R. Buss, "A Logtime Algorithm for Tree Isomorphism, Comparison and Canonization," pp. 18–33, *Lecture Notes in Computer Science* (including subseries *Lecture Notes in Artificial Intelligence* and *Lecture Notes in Bioinformatics*), 1997.
- [27] L. Kirsch, J. Botev, and S. Rothkugel, "Context-Based Authoring and Management of Documents," pp. 2431–2441, In *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, Oct 2013.
- [28] C. Grévisse, R. Manrique, O. Mariño, and S. Rothkugel, "SoLeMiO: Semantic Integration of Learning Material in Office," in *Proceedings of E-Learn: World Conference on E-Learning 2018*, Association for the Advancement of Computing in Education (AACE), in press.
- [29] A. F. Farhoomand and D. H. Drury, "Managerial Information Overload," *Communications of the ACM*, vol. 45, pp. 127–131, 7 2002.
- [30] "Linked Data." <https://www.w3.org/DesignIssues/LinkedData.html>, 2006. Accessed: 2018-07-21.
- [31] C. Grévisse, J. Botev, and S. Rothkugel, "An extensible and lightweight modular ontology for programming education," in *Advances in Computing* (A. Solano and H. Ordoñez, eds.), (Cham), pp. 358–371, Springer International Publishing, 2017.
- [32] C. Grévisse, R. Manrique, O. Mario, and S. Rothkugel, "Knowledge Graphs Enabled Automatic Semantic Representation to Improve the Retrieval of Learning Resources," in *Proceedings of the 13th Colombian Conference on Computing*, Springer International Publishing, in press.

Appendix

A Screenshots



Figure 1: Molecule Activity – The title page of our prototypical implementation.

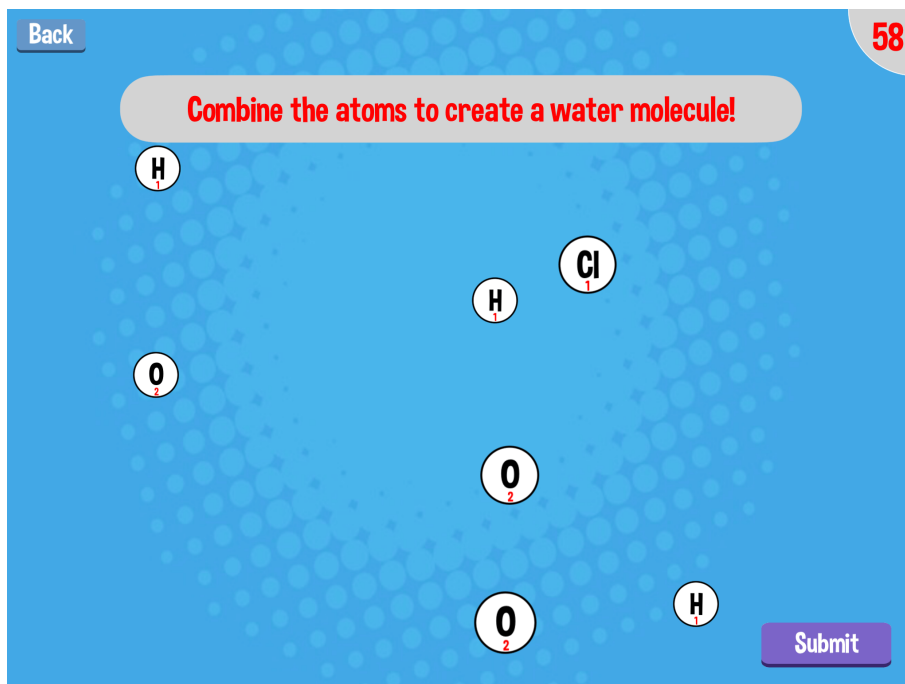


Figure 2: Molecule Activity – Initially the from the template provided molecules are randomly placed and a scale animation is run together with a sound effect to simulate a pop-up.

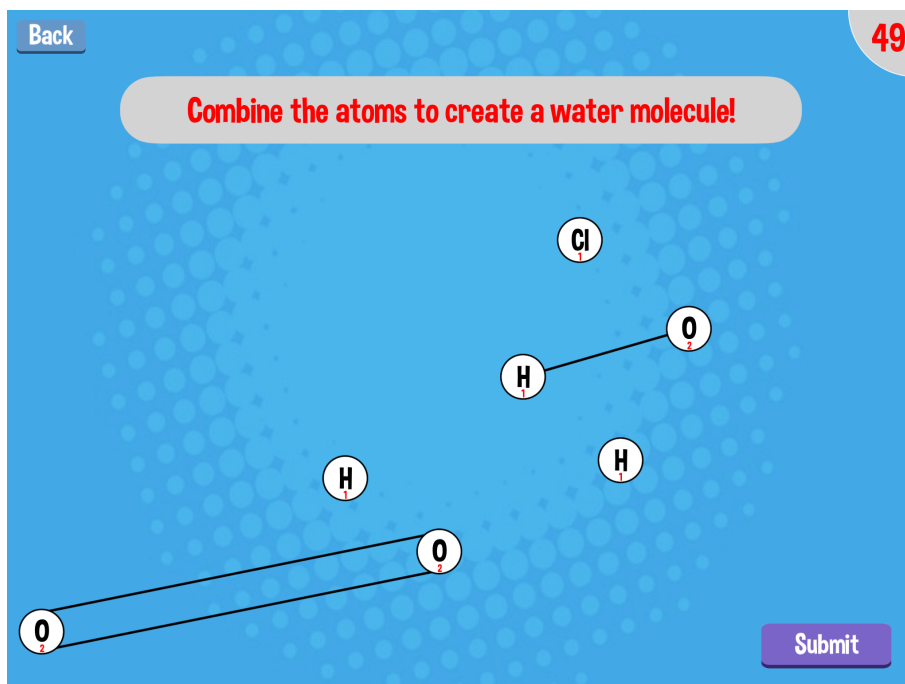


Figure 3: Molecule Activity – Student has wrongly combined two oxygen atoms with two chemical bondings.

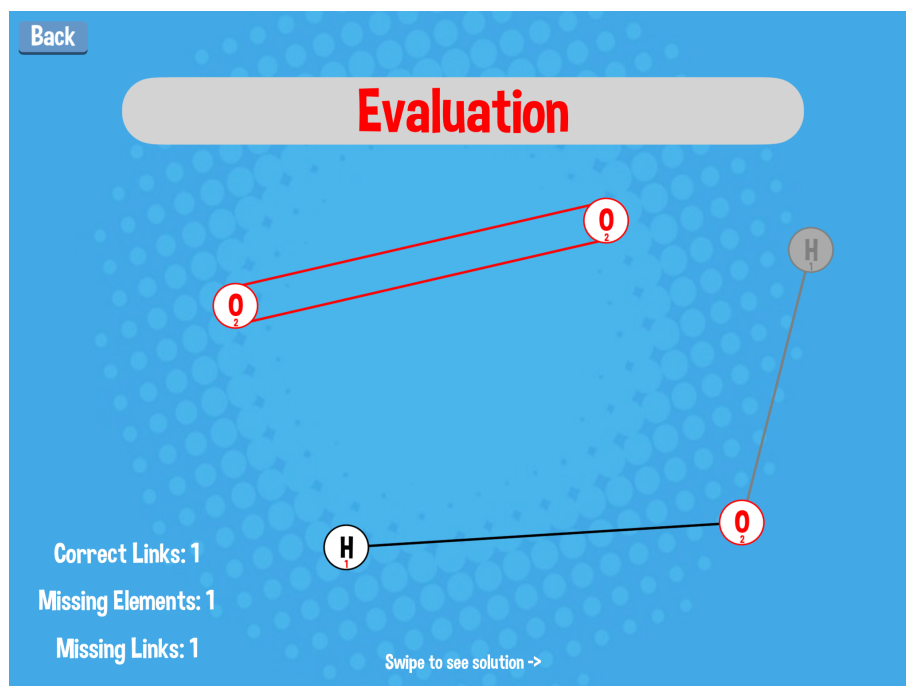


Figure 4: Molecule Activity – The evaluation shows a missing hydrogen molecule and colors the oxygen atoms red to illustrate an error.

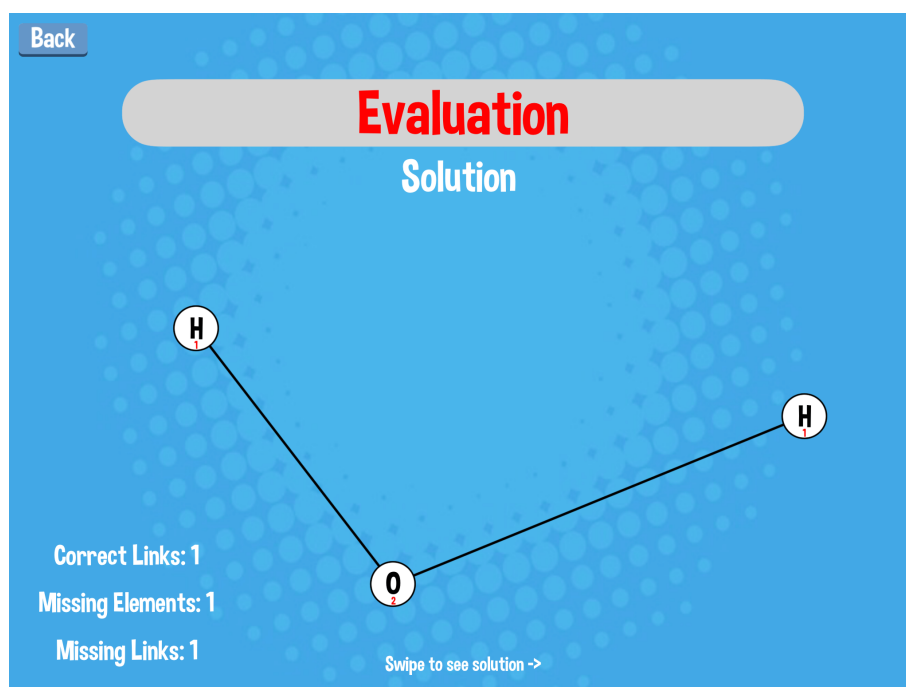


Figure 5: Molecule Activity – The solution of a water molecule is presented.

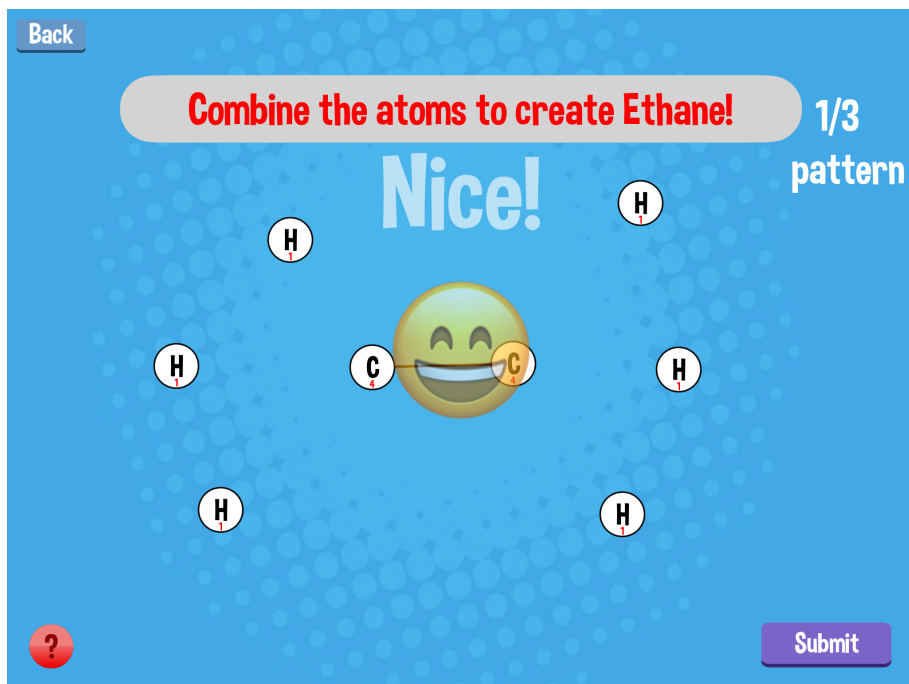


Figure 6: Molecule Activity – One out of three patterns were recognized, namely a bonding between two carbon atoms. The student receives an intermediate feedback to highlight this achievement. The feedback consists of an animation that pops up and fades out very fast to not disturb the student.

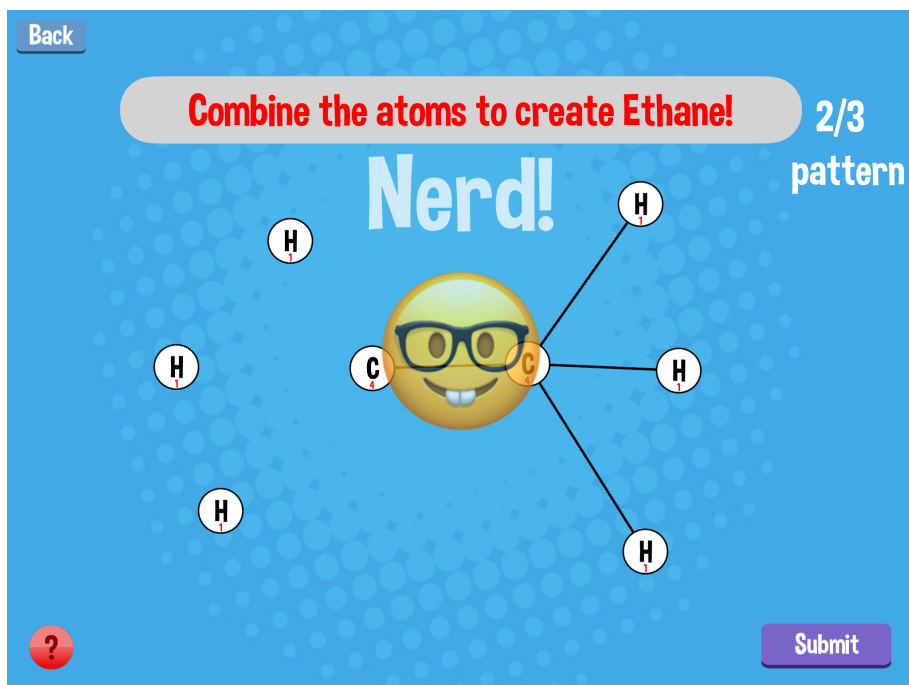


Figure 7: Molecule Activity – Two out of three patterns were recognized, namely a bonding between a carbon atom and three hydrogen atoms. The student receives an intermediate feedback to highlight this achievement. The feedback is based on the number of achieved patterns.

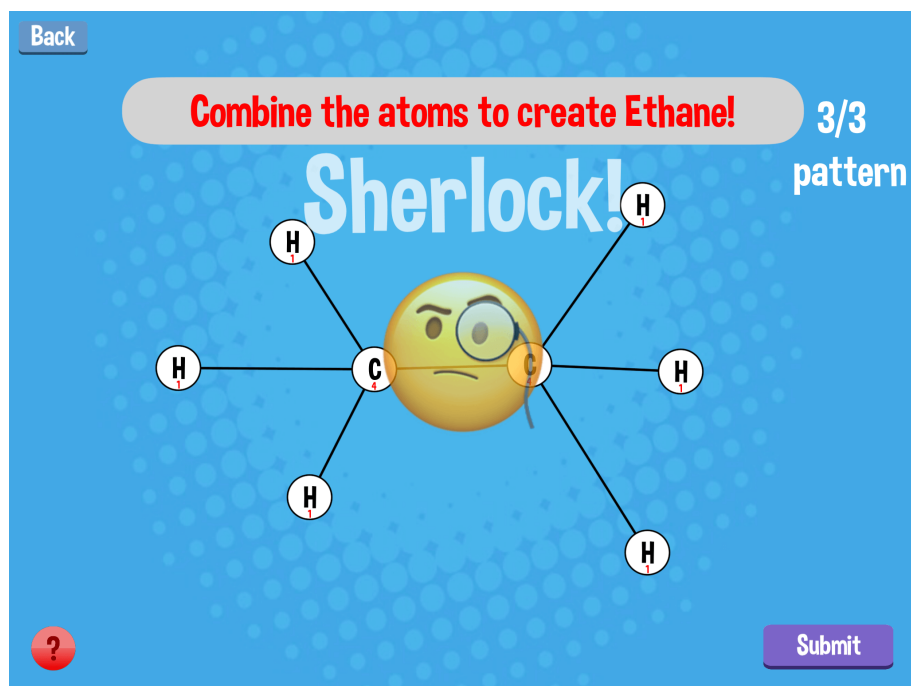


Figure 8: Molecule Activity – Three out of three patterns were recognized, namely a bonding between a carbon atom and three hydrogen atoms. The activity will shortly terminate, in favour of the student so that no more time elapses that would otherwise cause a point deduction.

Available Statements

Conditions
pot.isEmpty
not marmite.isFull
Actions
Fill-In
Peel
Conditionals

Submit

Figure 9: Algo. Block Activity – The available statements on the right hand side are provided through the *tools* relationship. The student needs to drag-and-drop statements from right to left to create a control flow. The problem is about filling the marmite with potatoes from the left barrow.

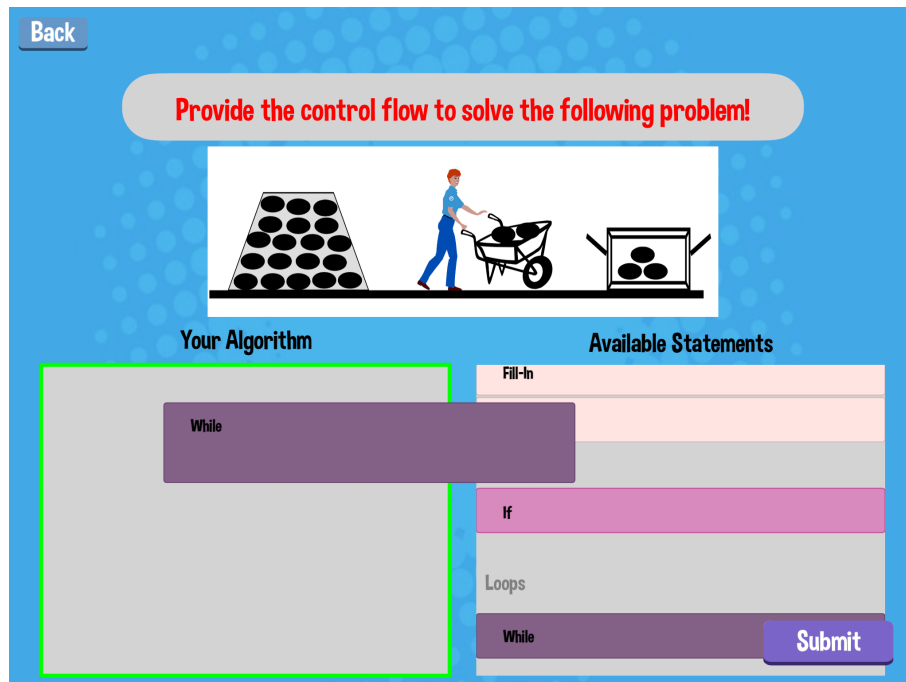


Figure 10: Algo. Block Activity – The student is dragging a while-loop into his list of statements. The loop will show its content as soon as it is placed into the list.

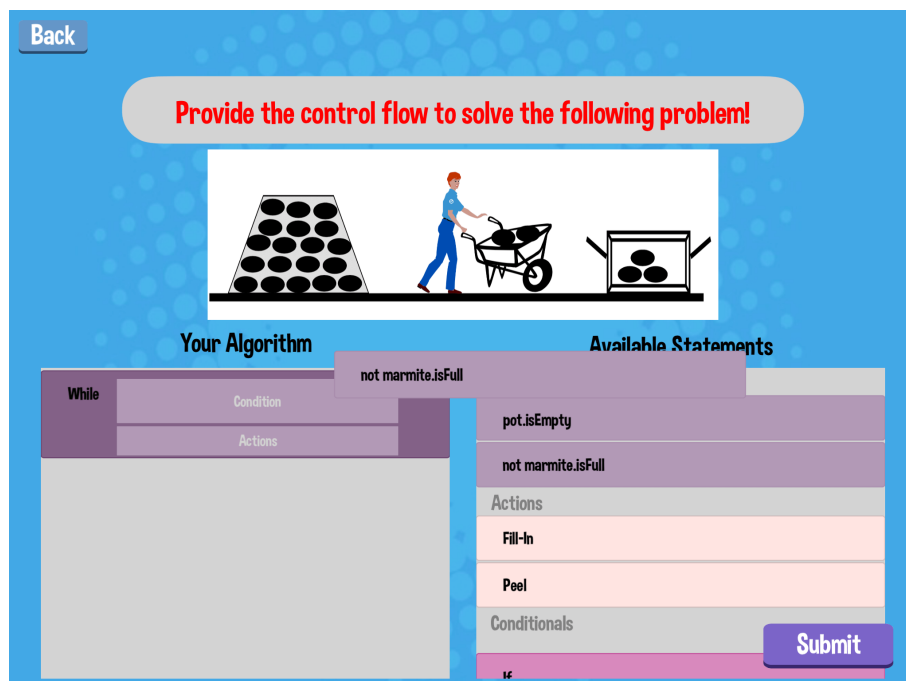


Figure 11: Algo. Block Activity – The content of the while-loop is now visible and the student is dragging a condition into it.

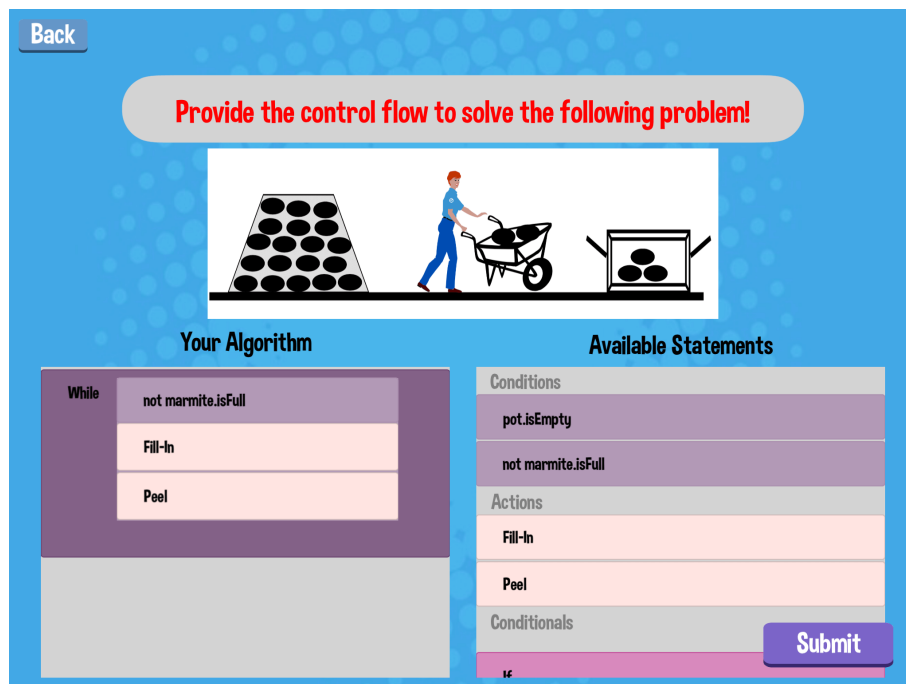


Figure 12: Algo. Block Activity – The student has now finished his task and is ready to submit.

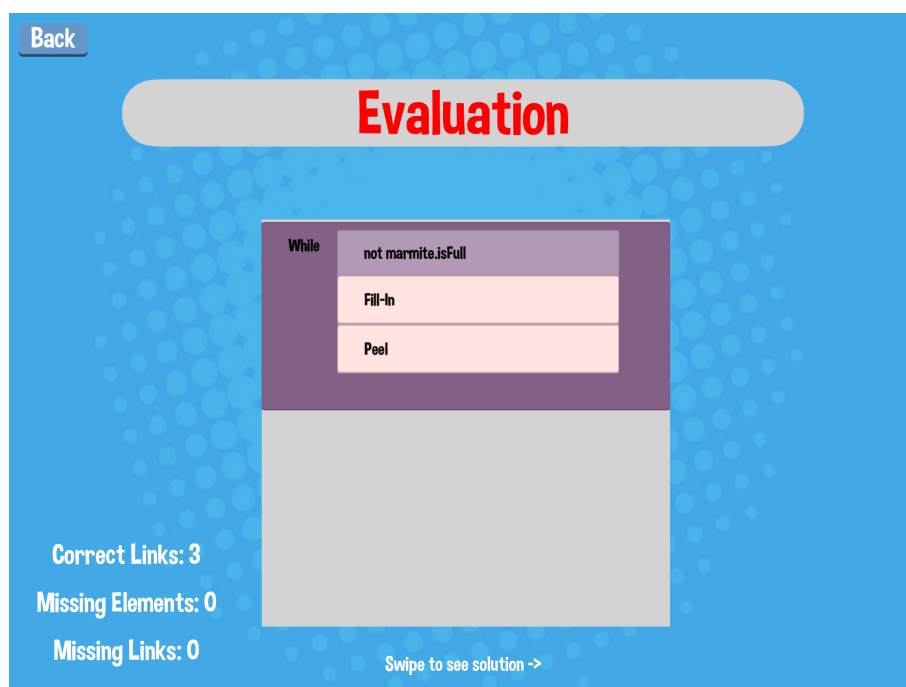


Figure 13: Algo. Block Activity – The evaluation were successful.

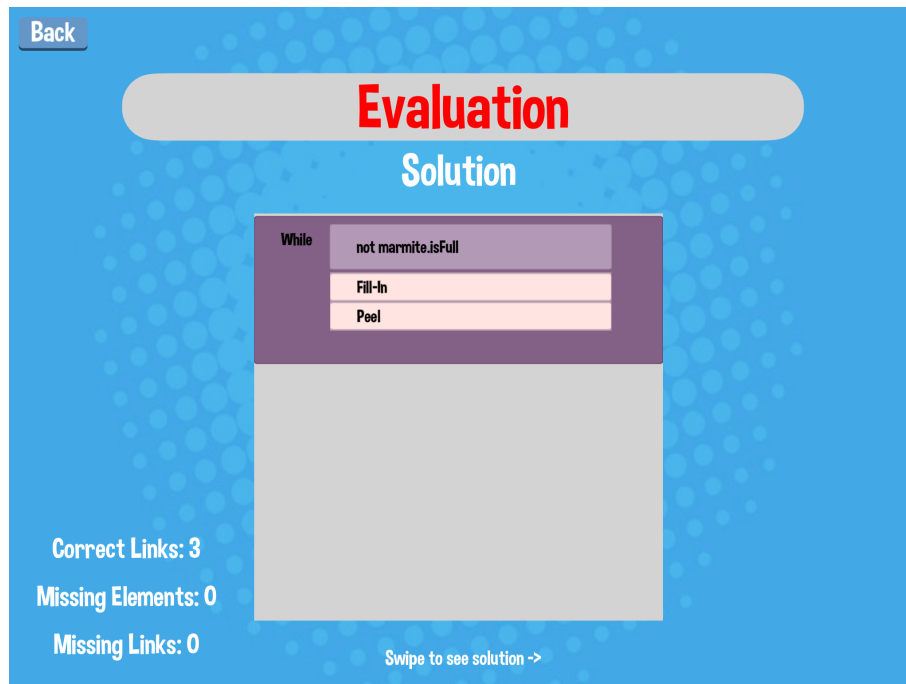


Figure 14: Algo. Block Activity – The solution matches the student's answer.

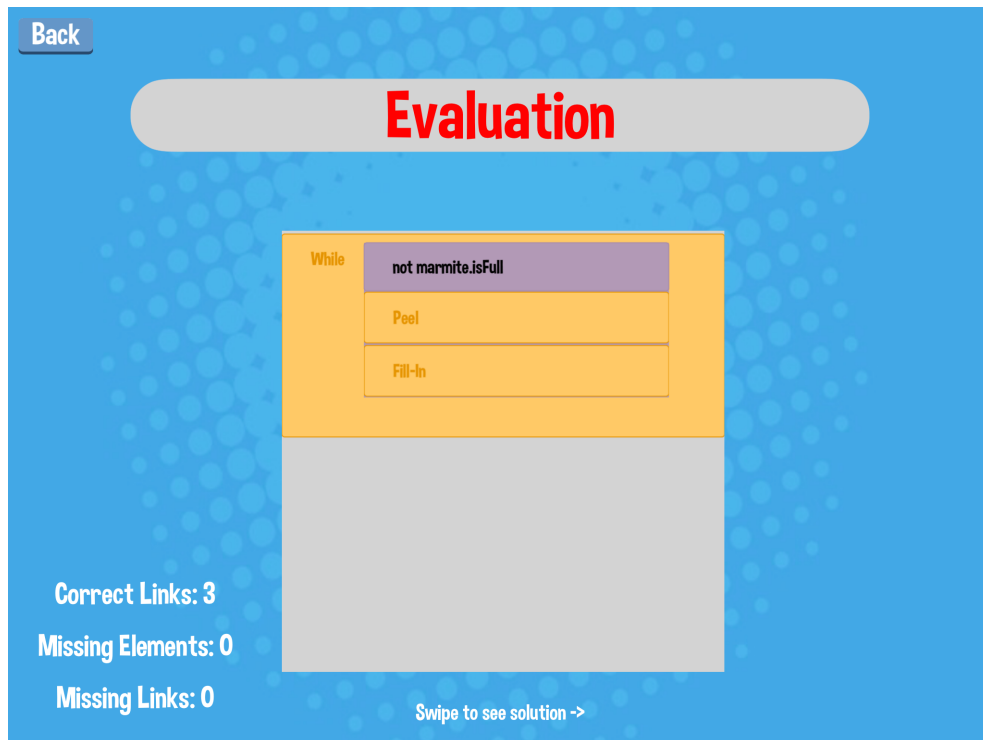


Figure 15: Algo. Block Activity – The evaluation were not successful since the student has put both actions in the wrong order which is illustrated using a yellow color.

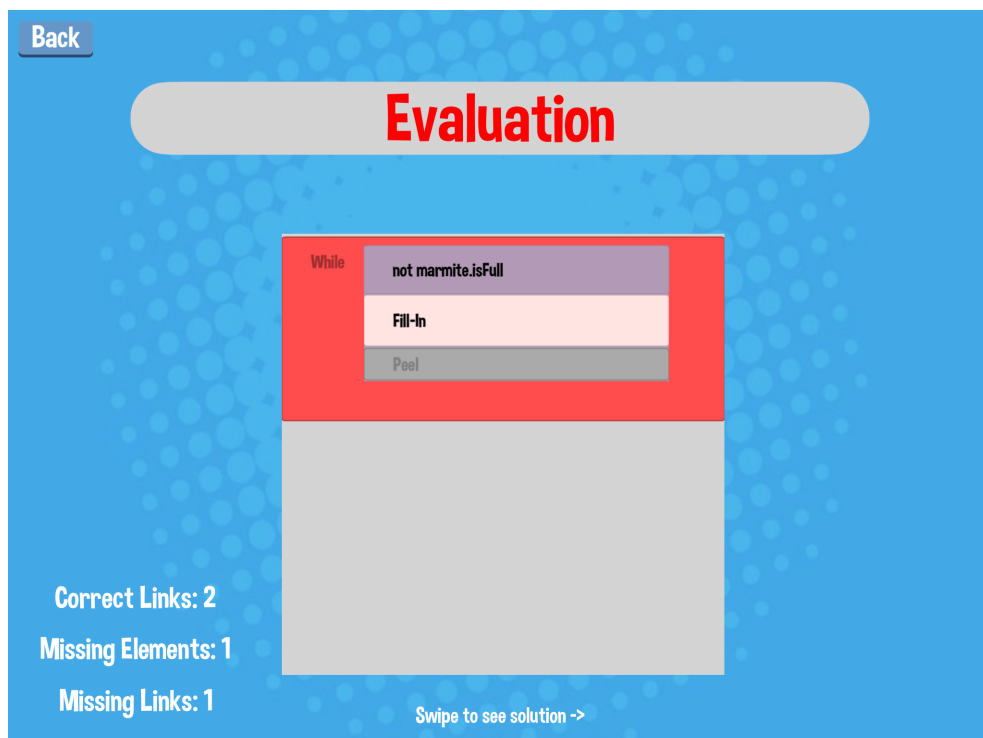


Figure 16: Algo. Block Activity – Algo. Block Activity – The evaluation were not successful since the student forgot one action statement which is depicted using a red color. The missing element is injected using a gray color.

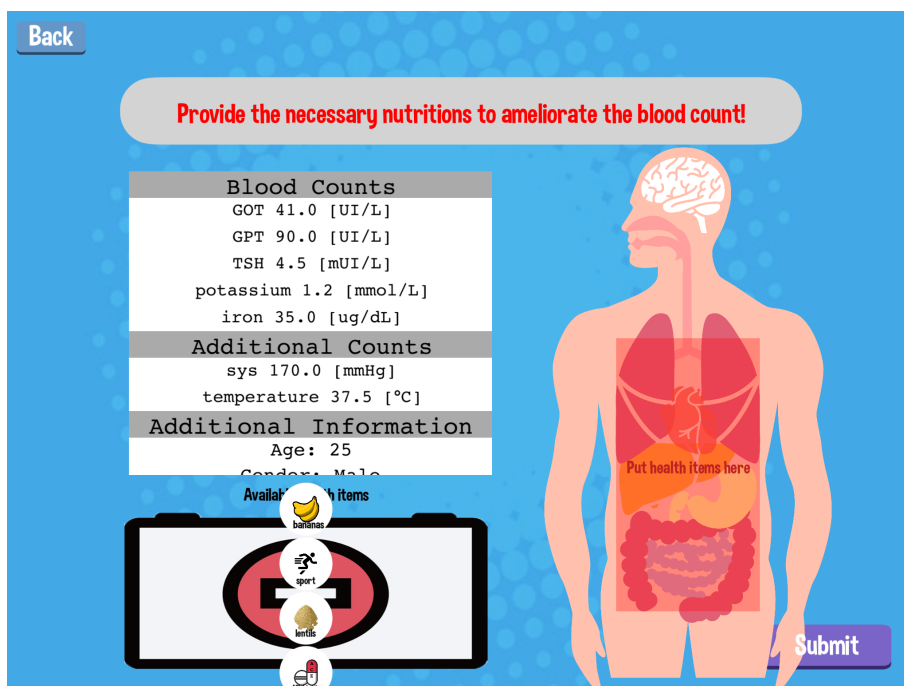


Figure 17: Blood Count Analysis Activity – The student needs to drag-and-drop health-items to the right anatomy in order to influence the counts.

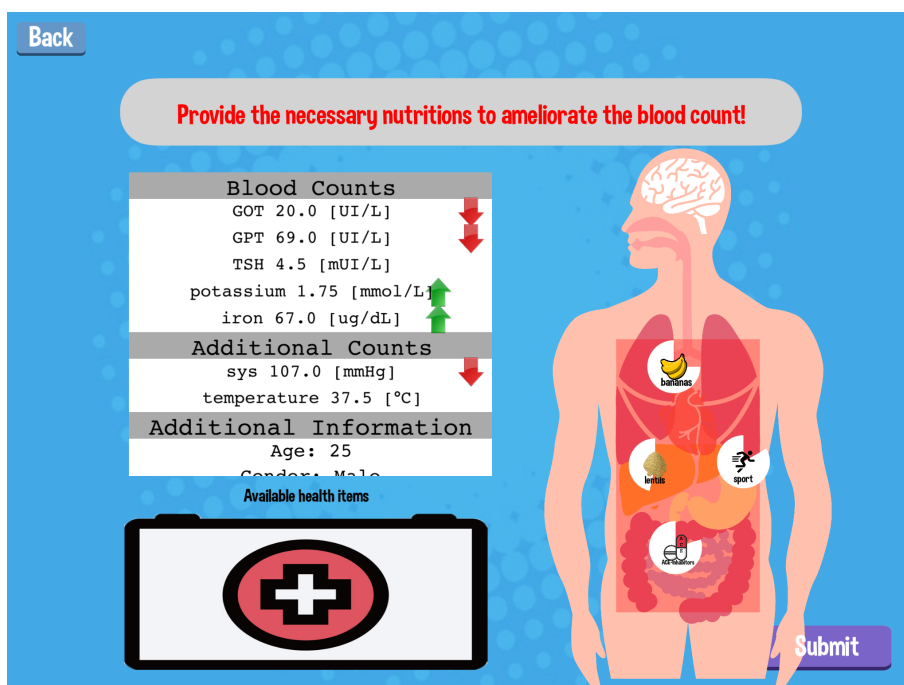


Figure 18: Blood Count Analysis Activity – As soon as the student drag-and-drops all available health-items into the anatomy, the influences will start using a timer. Positive or negative influences are depicted using a green or red arrow, respectively.

[Back](#)

Evaluation

GOT 5.0 [UI/L]
GPT 51.0 [UI/L]
TSH 4.5 [mUI/L]
potassium 1.9 [mmol/L]
iron 79.0 [ug/dL]
sys 89.0 [mmHg]
temperature 37.5 [°C]

Correct Links: 7
Missing Elements: 0
Missing Links: 0

Swipe to see solution ->

Figure 19: Blood Count Analysis Activity – The student evaluation were not quite successful and the framework colors the wrong items in red.

[Back](#)

Evaluation

Solution

GOT [5.0...34.0] [UI/L]
GPT [0.0...55.0] [UI/L]
TSH [0.35...4.5] [mUI/L]
potassium [3.5...5.1] [mmol/L]
iron [31.0...144.0] [ug/dL]
sys [90.0...120.0] [mmHg]
temperature [36.5...38.0] [°C]

Correct Links: 7
Missing Elements: 0
Missing Links: 0

Swipe to see solution ->

Figure 20: Blood Count Analysis Activity – The solution is shown using ranges of accepted values.

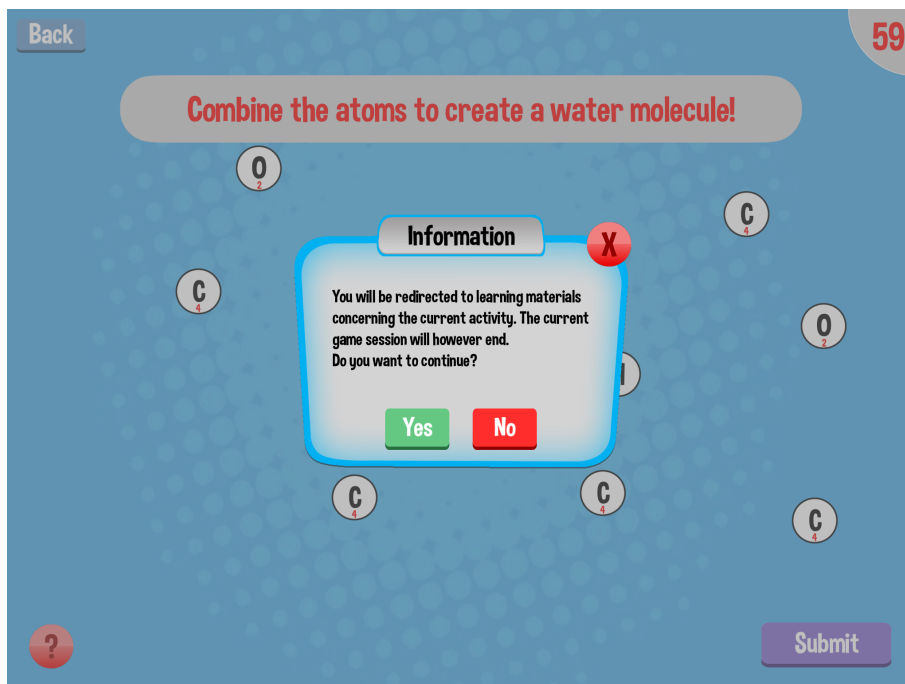


Figure 21: Message alerting the student that the activity will be finished when presenting learning materials since this could influence the result or score. The message pops up after touching on red icon in the bottom left.

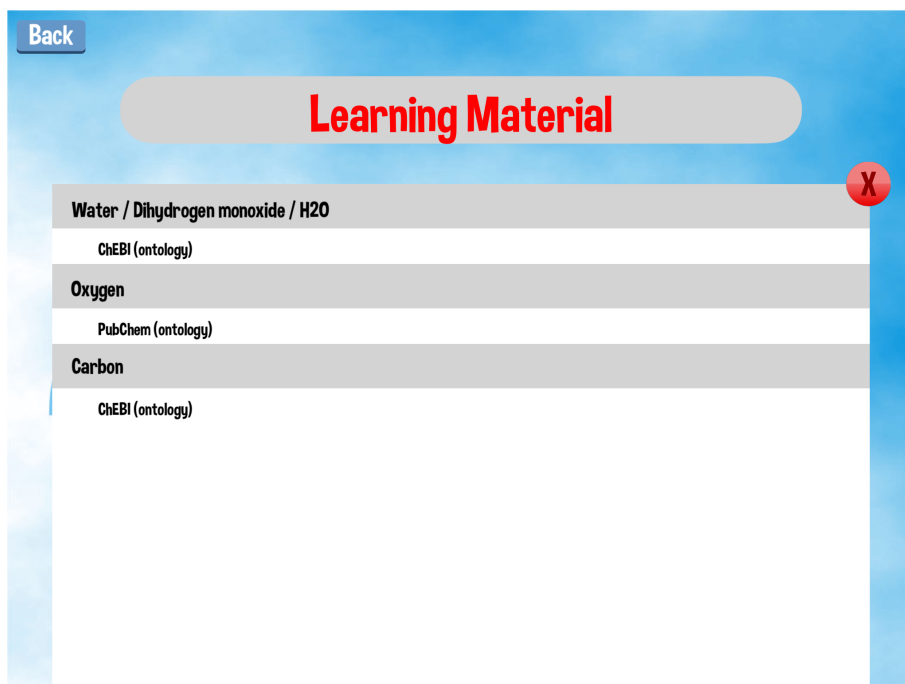


Figure 22: Learning materials for a molecule activity providing different links to ontologies about chemistry.

Back

Learning Material



EMBL-EBI

Services Research Training About us

ChEBI

Search

Examples: iron*, InChI=1S/CH4O/c1-2/h2H,1H3, caffeine

Home Advanced Search Browse Documentation Download Tools About ChEBI Contact us Submit

ChEBI > Main

CHEBI:15377 - water

Main ChEBI Ontology Automatic Xrefs Reactions Pathways Models

Chemical structure of water (H₂O):

```

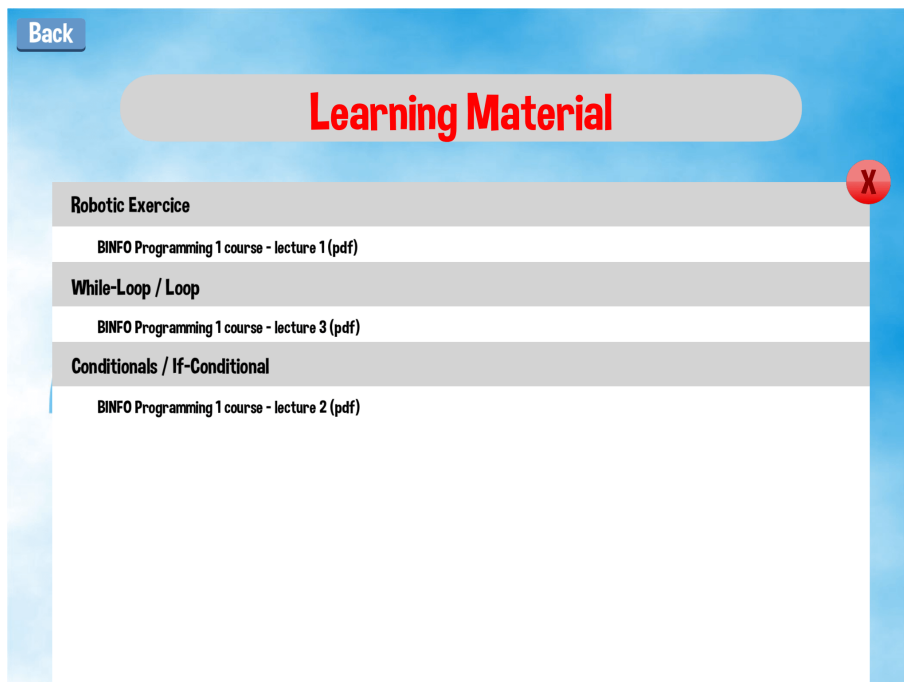
  H
   \
    O
   /
  H
  
```

ChEBI Name	water
ChEBI ID	CHEBI:15377
Definition	An oxygen hydride consisting of an oxygen atom that is covalently bonded to two hydrogen atoms.
Stars	☆☆☆ This entity has been manually annotated by the ChEBI Team.
Secondary ChEBI IDs	CHEBI:5585, CHEBI:42857, CHEBI:42043, CHEBI:44292, CHEBI:44819, CHEBI:43228, CHEBI:44701, CHEBI:10743, CHEBI:13352, CHEBI:27313

Figure 23: A Browser displaying ChEBI entry for water. ChEBI is a website implementing a dictionary of molecular entities and is backed by an ontology.

Back

Learning Material



Robotic Exercise

BINFO Programming 1 course - lecture 1 (pdf)

While-Loop / Loop

BINFO Programming 1 course - lecture 3 (pdf)

Conditionals / If-Conditional

BINFO Programming 1 course - lecture 2 (pdf)

Figure 24: Learning materials in the form of PDF's for an algo-block activity are listed.

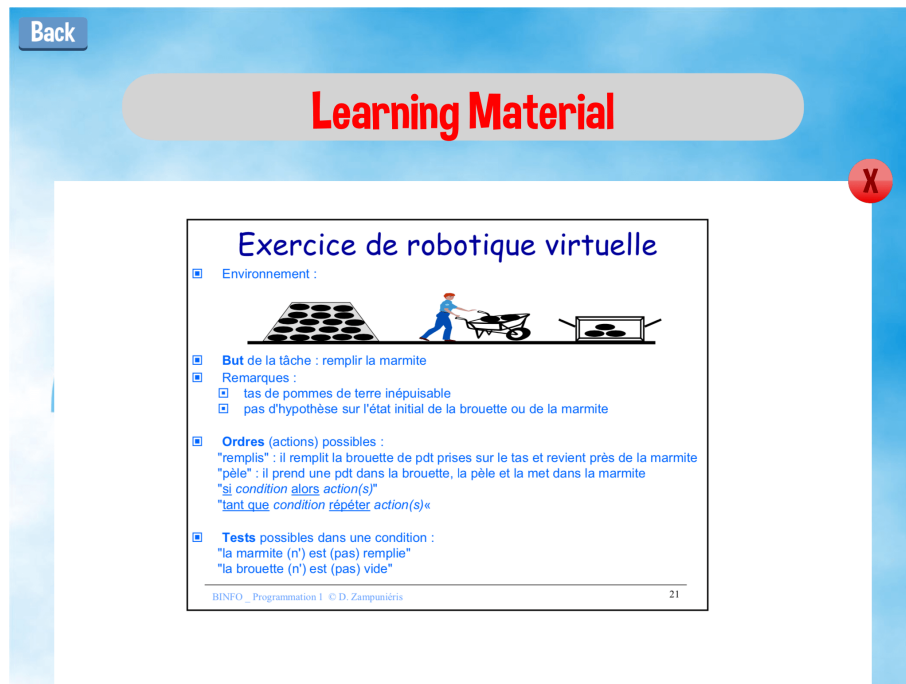


Figure 25: A PDF related to a programming course is presented to the student.

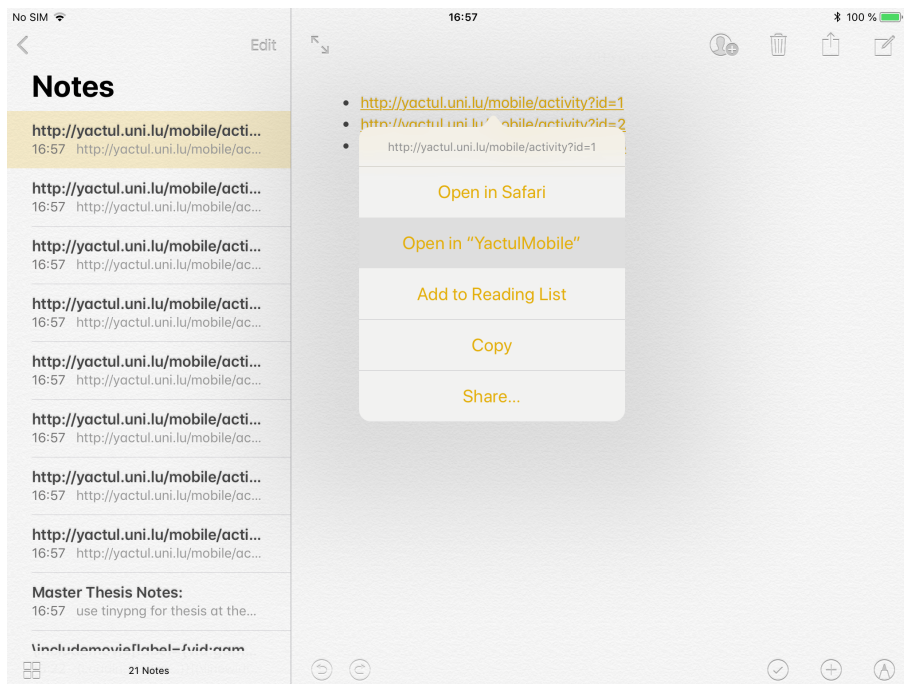


Figure 26: Dedicated HTTP Links can be opened using our app from everywhere in the iOS platform. Our app then handles query parameters, etc., and the dedicated task.

The screenshot shows a Microsoft PowerPoint presentation with the SoLeMiO add-in active. The main slide, titled "Exercice de robotique virtuelle", depicts a robot moving sand from a pile at point A to a container at point B. The SoLeMiO sidebar on the right, titled "Existing Annotations", lists resources for Slide 43 and Slide 5. For Slide 43, it includes a "While-Loop [Programming]" annotation and a video resource. For Slide 5, it includes a "While-Loop [Programming]" annotation and a video resource. The sidebar also shows a "Notes" button at the bottom.

Figure 27: The SoLeMiO add-in in Microsoft PowerPoint lists all resources that have been found for the particular lecture slides. Here, we have an URL and a video resource as listed in the right hand-side. The URL encodes an activity and will be opened by our dedicated app instead of the standard browser.

B Videos

Some videos about the implemented activities, learning materials and universal links are included in this section. Note that the videos cannot be played by every pdf reader.

Video 1 Gameplay of a molecule activity.

(Video Content – Use any PDF viewer that supports embedded videos.)

Video 2 Gameplay of an algo-block activity.

(Video Content – Use any PDF viewer that supports embedded videos.)

Video 3 Gameplay of a blood count activity activity.

(Video Content – Use any PDF viewer that supports embedded videos.)

Video 4 Gameplay showcasing the usage of learning materials within activities.

(Video Content – Use any PDF viewer that supports embedded videos.)

Video 5 Gameplay showing the usage of universal links.

(Video Content – Use any PDF viewer that supports embedded videos.)

Video 6 Gameplay showing the usage of universal links integrated in SoLeMiO. The slides are indexed with semantic terms such as while-loop and mapped with an URL resource, which points to a related activity.

(Video Content – Use any PDF viewer that supports embedded videos.)

C Simulation study

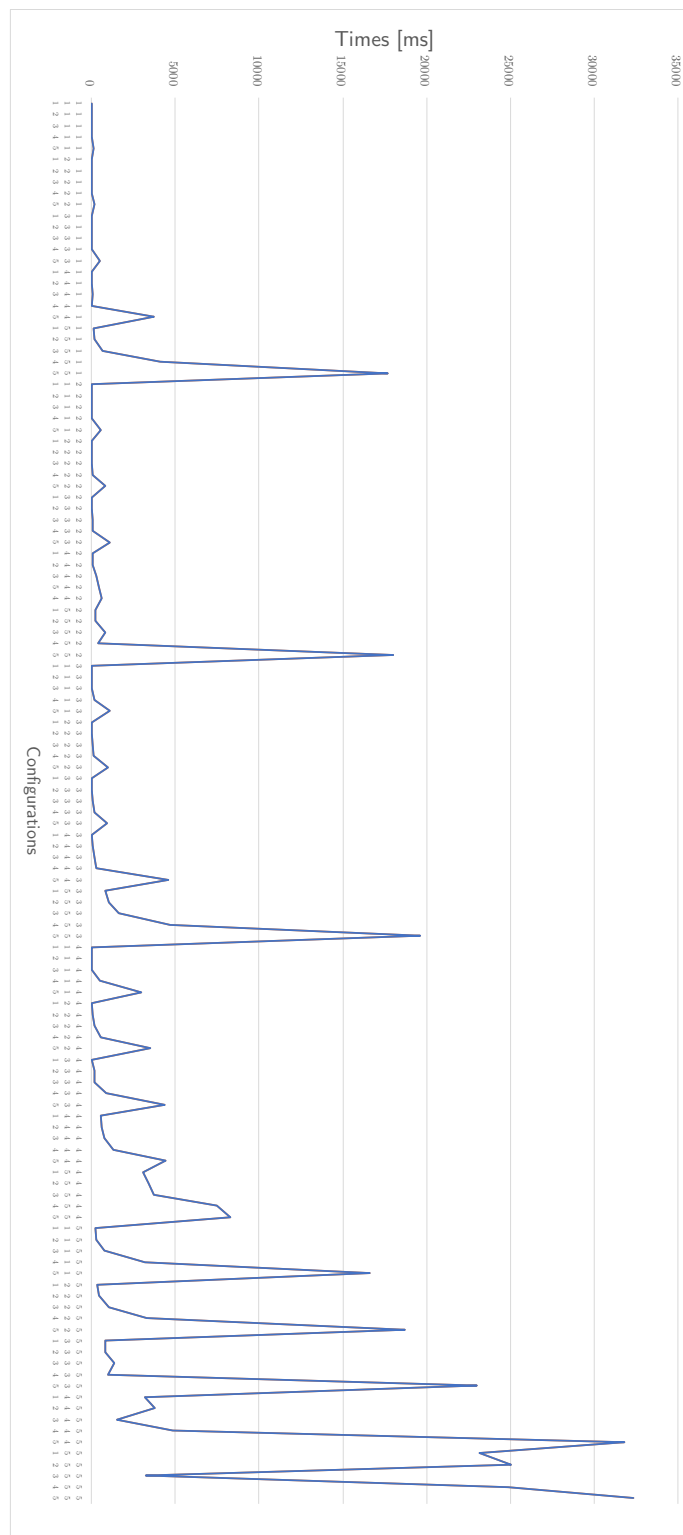


Figure 28: Benchmarking times – Time elapsed ordered in an ascending fashion where the times are mean values computed from 30 independent runs. An iPad Air 2 were used as a test device. The configurations are written as $(\#H, \#O, \#C)^T$ where H is the number of hydrogen, O the number of oxygen and C being the number of carbons. We can observe that the bigger the number of atoms become, the more time it takes to compute. The combination of two or more high numbers of atoms has a significant impact on the performance. The high peaks represent exponential increases

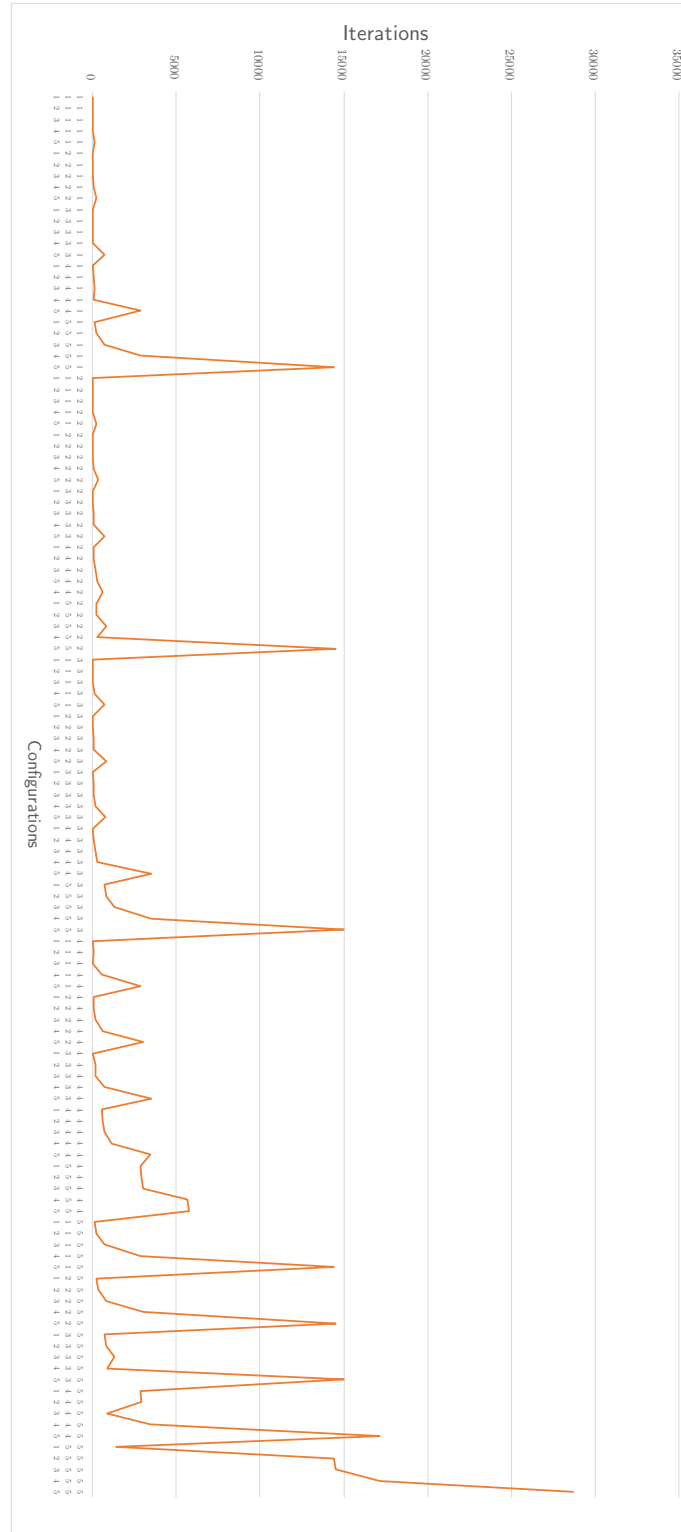


Figure 29: Benchmarking times – Iterations ordered in an ascending fashion where the values are means computed from 30 independent runs. An iPad Air 2 were used as a test device. The configurations are written as $(\#H, \#O, \#C)^T$. One iteration represents one execution of the algorithm 1. The high peaks represent exponential increases.

