Towards Model-Driven Gamified Software Modelling Learning

Alfa Yohannis*, Dimitris Kolovos, Fiona Polack
Department of Computer Science
University of York
York, United Kingdom
Email: *ary506@york.ac.uk

***Abstract*— Backgrounded by the challenges in teaching and learning software modelling, this research harnesses the engaging nature of games, the effectiveness of pedagogy, and the automation of Model-driven Engineering to propose a platform for model-driven gamified software modelling learning. It is an environment for tutors where they can create new and make use of existing software modelling learning activities for teaching and for learners to learn software modelling topics in a gameful way. This paper presents the rationale of the idea behind the initiation of the platform as well as its design considerations which mostly are withdrawn from game, pedagogy, and model-driven domains. An introduction to the concept of substate is statechart modelling demonstrated to show how the platform works, and some identified significant challenges and open issues are also discussed as important lessons for further research and development.**

***Keywords*—model-driven, gameful, software modelling, learning, platform**

I. Introduction

II. Rationale

Software modelling teaching and learning (SMTL) has significant role in model-driven domain, since it establishes the very foundation in Model-driven Engineering (MDE) learners whom in the future will be MDE experts and practitioners. If the process of SMTL can be improved, it will bring benefits not just to learners but also to model-driven domain and wider related domains. So, what are the challenges in SMTL? How they should be addressed?

The first challenge is the abstraction. It is the very nature attribute of modelling since modelling is the process of thinking abstractly about systems [1]. Some writers emphasise the cruciality of abstraction in computer science and software engineering [2], [3], [4] in general as well as model-driven domain [5], [6], [7], [8] in particular.

The nature of abstraction makes the concepts being learned are difficult to access since the concepts are only accessible through tangible representation, such as symbols and diagrams [9]. Even though the abstract concepts have visualized, engagement can still be an issue, that is making learners to actively participate in learning processes. This is the condition where gameful approach comes into play to solve motivational issues. Game elements can be embedded into learning activities to present gameful experience.

Other problem is to posses abstraction skills–ability to perform abstractive operations, such as selecting the appropriate abstraction level, moving through different abstraction levels, and view abstraction from different perspectives. To learn the abstraction skills, Some experts suggest that teaching and learning abstraction should start from familiarizing with real-world objects an then move upwards to abstraction through similarity [5], [10]. However, the process of abstraction could continue to move downwards to reification and then finally to application[10]. Hazzan suggests that in order to grasp abstract concepts, they should be experienced–seen, felt, and used. Therefore, the concepts should be illustrated, reflected, and practised by learners [3]. Supporting tutors to perform these approaches will help them tackling the abstraction challenge.

The second challenge is the contents of SMTL. Some experts suggest the contents should be definitions, semantics, syntaxes, notations of modelling [11], other experts point the core concepts of MDE–modelling, metamodelling, and model transformation–and their applications [1], and others prompt the engineering aspect of software modelling should not be neglected [12]. While main ideas of the contents have been proposed by experts, the detail structures how the contents should be organised is not clear yet. For example, statechart modelling has concepts of states, transitions, decisions, regions, and etc. There might be some structures that are best these concepts for teaching and learning statechart modelling and it is good to follow such structures. However, there are no 'one-for-all' structures that fit to all contexts, creativity of tutors, and learners. So rather than only following the best structures, it is best to support tutors to express organisation of the contents based on their preferences and needs, regardless whether they are inspired by the best structures or not.

The third challenge is the teaching and learning Approaches. One approach in to teach software modelling broadly, throughout, and not deeply[11], [12], so they learn to decide which modelling process is more appropriate than others [13]. Other approach is harnessing modelling tools which are very important to tackle large-scale modelling and ecourage to produce good models [13]. However, failure to provide enough supports (e.g. helpful instructions and expert tools) might overwhelm learners with unnecessary features [14]. These are just two Approaches of many other Approaches proposed by experts in software modelling. On the other side, the field of pedagogy also have employed collections of approaches for teaching and learning in general, for examples, Bloom's taxonomy [15], Kolb's experiential learning [16], and Keller's ARCS motivational model [17]. All of these approaches are worth to be considered in SMTL and should not be neglected. However, considering the same reason that applies to contents there are no 'one-for-all' approaches, it is best to support tutors to design their own approaches in SMTL, whether they follows the existing approaches or their own creativity.

On the other side, MDE provides a convenient way to produce software [18]. The software itself could act as a platform for tutors to create contents and approaches for teaching certain concepts in software modelling in the form of learning activities and also for learners to engage with the learning activities. MDE offers the advantages to speed up the development of the learning activities. First, using the platform, tutors can harness a high-level visual modelling language to design the learning activities at high-level abstraction. Second, the platform automates the generation of runnable instances of the learning activity design, which with the runnable instances learners can perform software modelling learning.

To sum up, in order to improve SMTL, two conditions have to be met. First, SMTL should be engaging for learners. This could be done through embedding game elements into learning activities. Second, SMTL should be creative. Tutors are given supports to create new and reuse existing SMTL contents and approaches, whether inspired by existing good practices or just according to their own creativity. Thus, to satisfy these conditions, a platform for model-driven gamified software modelling learning is proposed. It is an environment for tutors where they can create new and make use of existing software modelling learning activities for teaching and for learners to learn software modelling topics in a gameful way.

Once the platform can be fully utilised, it will empower tutors to express their creativity in producing different patterns of learning activities, whether the learning activities are based on existing best practices or new approaches in teaching certain concepts in SMTL. The platform will also ease learners in learning software modelling since the learning activities are presented a gameful way. As soon as many learning activities are available, the learners can choose learning patterns that is best for them and exercise through many different patterns.

All interaction of tutors and learners with the platform can be logged and logs are the source of data which further could be used to extract knowledge to understand deeper about teaching and learning behaviours in software modelling. Using analytical techniques, such as machine learning and data mining, questions like which part or learning activity that learners often make mistakes, what parts of software modelling that are most challenging for them, what strategies that they prefer to use to construct models for certain problems, and many other questions could be answered and understood. The understanding that could change the way software modelling is taught and learned. Moreover, the platform could also be a place for research and experiment. For example, comparing two learning patterns which one is better in teaching one same concept in software modelling and investigating which shapes that are more appropriate to represent a certain concept in a visual modelling language. With further development, the platform could also support 'multiplayers' which means collaboration in SMTL could also be supported. This also means concepts like collaborative learning and content creation can also be tested and applied into SMTL.

III. Design

A. Gameful Design

B. Pedagogical Design

C. Model-Driven Design

IV. Demonstration

In this section, an example is presented to demonstrate the use of the platform. The example is to show the of teaching the concept of substates in UML statechart modelling. In this demonstration, designers are assumed to have proficient knowledge and skills of statechart diagram while learners are assumed already understood the concept of states, transitions, start state, and final state. The following subsections are the order of the statechart from its definition to be fully used in a game play.

A. Define Statechart Metamodel

In order for the platform to support modelling in statechart diagram, designers have to define its metamodel first. The metamodel is written in Emfatic (https://www.eclipse.org/emfatic) and Eugenia-like annotations [19] are used to define its concrete syntax. For example, the definition of Start State derived from Node class is displayed in Fig. 1. Other basic elements of statechart diagram, such as states, transitions, end states, etc., are also defined but not displayed in the figure.

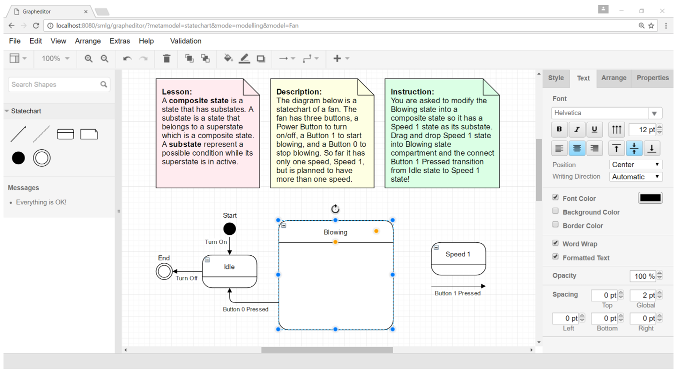
```
abstract class Node extends Entity {
  ref Edge[*]#source outgoing;
  ref Edge[*]#target incoming;
}

@node(label="name", dhape="startState", whiteSpace="wrap",
     html="1", fillColor="#000000", width="30", height="30")
class Start extends Node {
}
```

Fig. 1: A definition of Start State derived from Node class using Emfatic and Eugenia-like annotations.

The annotated metamodel then trasformed using Epsilon [20] and EMF (http://www.eclipse.org/modeling/emf/) to generate the Java codes of the metamodel, as the backbone codes for further model operations, and Javascript codes to display the statechart's elements as visual elements in a palette of an MxGraph-based graphical editor (https://jgraph.github.io/mxgraph/).

B. Create Statechart Model

Using the MxGraph-based graphical editor (Fig. 2), designers can now create statechart models. The editor has a pallete on the left side that contains elements of statechart's elements which they can drag and drop into a drawing area on the centre where they can arrange the elements to construct statechart models. The editor also has a property panel on the right side to modify the attributes and appearance of the models. Models that have been created can be saved and loaded again for further operations. The editor also can be used to create models as input/base models to be embedded in learning activities.



• Everything is OK!

Lesson:
A **composite state** is a state that has substates. A substate is a state that belongs to a superstate which is a composite state. A **substate** represent a possible condition while its superstate is in active.

Description:
The diagram below is a statechart of a fan. The fan has three buttons, a Power Button to turn on/off, a Button 1 to start blowing, and a Button 0 to stop blowing. So far it has only one speed, Speed 1, but is planned to have more than one speed.

Instruction:
You are asked to modify the Blowing state into a composite state so it has a Speed 1 state as its substate. Drag and drop Speed 1 state into Blowing state compartment and the connect Button 1 Pressed transition from Idle state to Speed 1 state!

Font: Helvetica, 12 pt
Position: Center
Writing Direction: Automatic
Opacity: 100 %
Spacing: Top 0 pt, Global 2 pt, Left 0 pt, Bottom 0 pt, Right 0 pt

Fig. 2: The MxGraph-based graphical editor to create statechart models.

C. Create Learning Pattern Model

To teach and learn a concept, designers have to design a learning pattern that consists of ordered learning activities. In this example, a learning pattern to learn the concept of substates is created (Fig. 3). The case that is selected is an electrical fan that has two main states, Idle and Blowing. Learners will be asked to modify the Blowing state so it becomes composite state that has several substates that represent different speed of blowing. The learning pattern has three activities, namely One-Speed Fan, Two-Speed Fan, and Tree-Speed Fan. The activities are arranged *per se* to accommodate Flow state [21] so learners can start learning from the easiest activity to the hardest one. The activities are explained in more detail in the next following subsections.

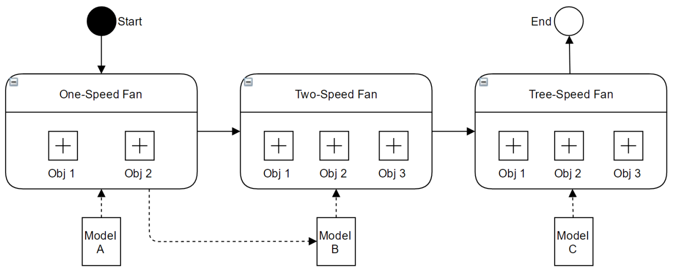
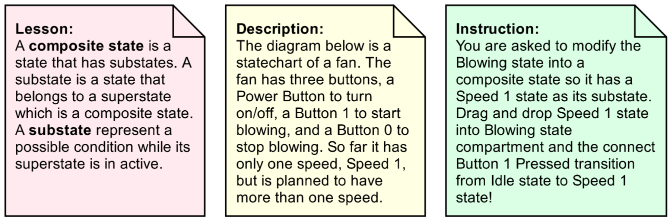


Fig. 3: A learning activity pattern for learning the concept of Substates in state-machine modelling.

Each activity has lesson and instruction properties. Lesson contains explanation about the concepts that are being taught and instruction contains commands or questions that learners need to execute or answer. In the process of satisfying the instruction in each activity, one or more objectives have to be met by learners in order to move to next activity. Also, each activity can consume existing models (Model A, B, and C in Fig. 3) as its base models so learners do not have to create a model from scratch and produce model (Model B in Fig. 3) to be used in its next activity. Each of the models also has a description property to describe itself which is very useful for learners to understand about the model. An example of the lesson, model description, and instruction properties of the One-Speed Fan activity (Fig. 3) is displayed in Fig. 4.



Lesson:
A **composite state** is a state that has substates. A substate is a state that belongs to a superstate which is a composite state. A **substate** represent a possible condition while its superstate is in active.

Description:
The diagram below is a statechart of a fan. The fan has three buttons, a Power Button to turn on/off, a Button 1 to start blowing, and a Button 0 to stop blowing. So far it has only one speed, Speed 1, but is planned to have more than one speed.

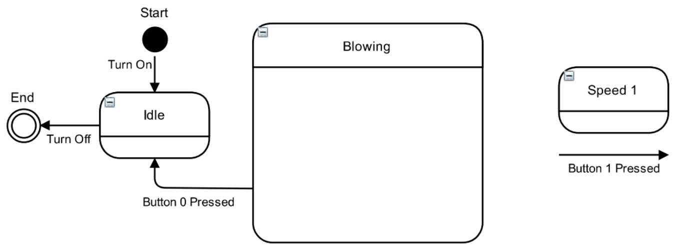
Instruction:
You are asked to modify the Blowing state into a composite state so it has a Speed 1 state as its substate. Drag and drop Speed 1 state into Blowing state compartment and the connect Button 1 Pressed transition from Idle state to Speed 1 state!

Fig. 4: Lesson, model description, and instruction.

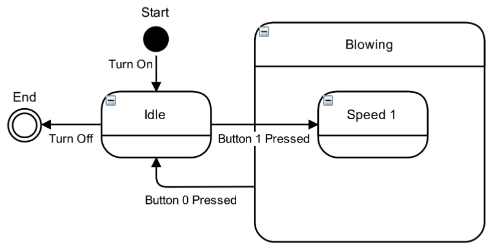
1) One-Speed Fan Activity: In this activity (Fig. 5), learners are introduced to one substate only.The activity start with a base model and learners are required to modify the base model (Fig. 5a) to meet the target model (Fig. 5b). The base model corresponds to Model A in Fig. 3, which refers to an existing model that already created previously and will be loaded once the activity is executed so leaners do not need to create the model from the start.

The example case in this activity is a fan that has three buttons, a Power Button to turn on/off, a Button 1 to start blowing, and a Button 0 to stop blowing. So far it has only one speed, Speed 1, but is planned to have more than one speed. Learners are asked to modify the Blowing state in Fig. 5a into a composite state through moving the Speed 1 state into the Blowing state compartment as well as to connect the Button 1 Pressed transition from the Idle state to the Speed 1 state. Since in Fig. 3 this activity is designed to have only two objectives, the two objectives are adjusted and defined as follow: Objective 1 ”Blowing state contains Speed 1 substate” and Objective 2 ”Button 1 Pressed transition connects Idle state to Speed 1 substate”.

2) Two-Speed Fan Activity: The Two-Speed Fan activity (Fig. 6) consumes the model that has been produced in the first activity (Model B in Fig. 3). Therefore, any model produced in the previous activity will become the base model for modelling in this activity. Inline to the Flow concept [21], this second activity has to be more challenging. Therefore, the activity (Fig. 6) challenge learners with one additional state and three new transitions. Now the case has changed. The fan has an additional button, Button 2, to support 2-speed blowing. When Button 1 is pressed, the fan blows in speed 1. When Button 2 is pressed, the fan blows in speed 2. Thus, Learners are required modify the Blowing state into a composite state so it has two speed states. The fan can move from the Idle state



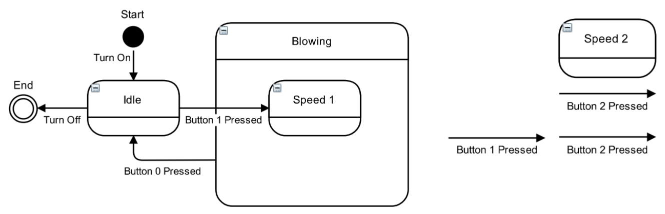
(a) Base model



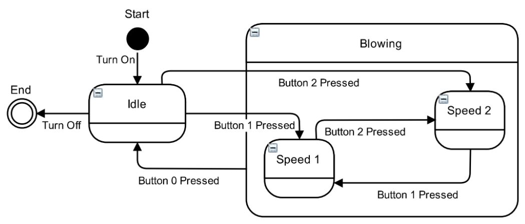
(b) Target model

Fig. 5: The One-Speed Fan example.

to the Speed 1 state, from the Idle state to the Speed 2 state, and from the Speed 1 state to the Speed 2 state or *vice versa*.



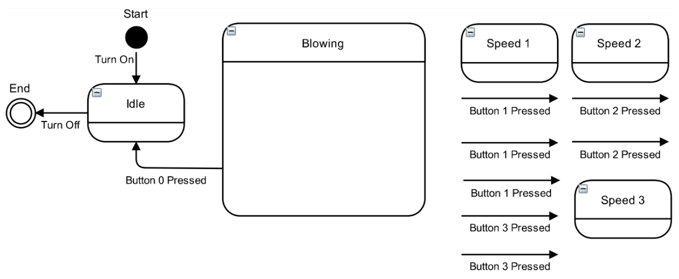
(a) Base model



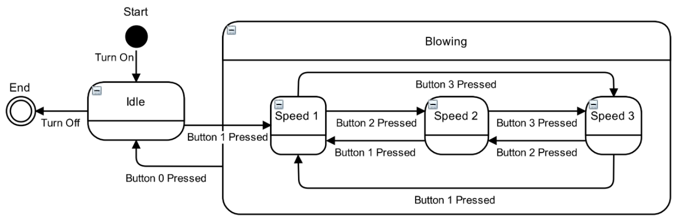
(b) Target model

Fig. 6: The Two-Speed Fan example.

3) Three-Speed Fan Activity: The Three-Speed Fan activity (Fig. 7) should be more difficult than the second activity. The fan now supports 3 speeds of blowing, but it has been modified so it cannot go directly to Speed 2 and Speed 3 without firstly go the state with lower speed. In other words, transition from Idle can only go to Speed 1 for the fan to start blowing–Speed 2 and Speed 3 will not be working if transition comes from Idle state. Thus, starting from the model C as the base model as shown in Fig. 3, learners are required to modify the Blowing state into a composite state so it has 3 speeds of blowing and only allow transition from Idle to Speed 1 and from a speed that is lower from the intended one in order to blow.



(a) Base model



(b) Target model

Fig. 7: The Three-Speed Fan example.

D. Generate Game

After finishing constructing the learning pattern (Fig. 3), designers can now generate a path of levels (or stages) that is usually found in many games. Each generated level corresponds to an activity defined in the learning pattern. The generation also produces an EVL [22] template for each level for its validation to determine whether the level has been completed, all of its objectives have been met, or not. The EVL template is shown in Fig. 8 which can be extended by designers to write code that fits with the level scenarios and objectives. The number of constraints and operations in the template corresponds to the number of objectives defined in the level's learning pattern activity in Fig. 3. As an example implementation of validation, Fig. 9 shows the EVL code of checking whether Blowing state has already Speed 1 substate as intended by Objective 1 in One-Speed activity.

E. Play Game

After generating the learning pattern into a path of ordered levels and defining its levels' validation, Learners can choose the path and play its levels as depicted in Fig. 10. When learners choose the first level, an MxGraph-based editor is displayed. Lesson, model description, instruction, objectives, and base model are presented to learners. Learners then can start to modify the base model to reach the target model.

Every attempt to change the model will trigger an operation to validate the current model using the defined EVL constraints to assess whether the current model has met the current level's objectives. If not, error messages are displayed to learners indicating the objectives that have not been satisfied. For example, in Fig. 11, the Speed 1 substate has not been put into the Blowing state even though the Button 1 Pressed transition has connected the Idle state to the Speed 1 substate. If all objectives have been fulfilled, learners have completed the

```
context Statechart {
    constraint obj_1 {
        check:
            self.obj_1()
        message:
            "FAIL: ob_1"
    }
    constraint obj_2 {
        check:
            self.obj_2()
        message:
            "FAIL: obj_2"
    }
}
operation Statechart obj_1(): Boolean {
    return true;
}
operation Statechart obj_2(): Boolean {
    return true;
}
```

Fig. 8: Validation template for objectives in One-Speed Fan activity/level.

```
context Statechart {
    constraint obj_1 {
        check:
            self.obj_1()
        message:
            "FAIL: Blowing state contains Speed 1 substate"
    }
    ...
}
operation Statechart obj_1(): Boolean {
    for (state in State.allInstances.select(state | state.
         name == "Blowing")) {
        if (state.substates.notEmpty() and state.substates.
             select(substate | substate.name == "Speed 1").
             notEmpty()){
            return true;
        }
    }
    return false;
}
...
```

Fig. 9: Validation realisation for Objective 1 in One-Speed Fan activity/level.

level. A message that congratulate the learners is displayed to give positive reinforcement. Learners then are brought back to the learning path where they can choose the next level to play.

V. Challenges and Open Issues

VI. Related Works

Acknowledgment

This research is part of a doctoral programme supported by *Lembaga Pengelola Dana Pendidikan Indonesia* (Indonesia Endowment Fund for Education).

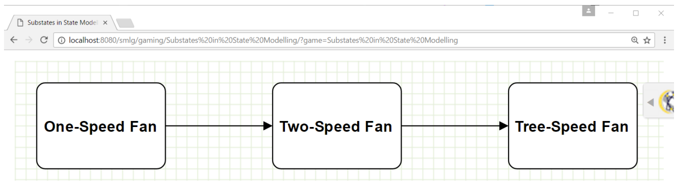
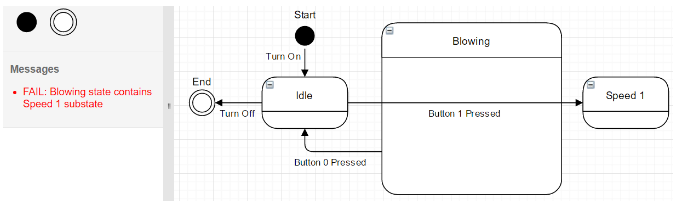


Fig. 10: The path for learning substate concept with its levels.


Messages

- FAIL: Blowing state contains Speed 1 substate

Fig. 11: A message is displayed to learners indicating the objective that has not been fulfilled.

References

[1] J. Bezivin, R. France, M. Gogolla, O. Haugen, G. Taentzer, and D. Varro, "Teaching modeling: why, when, what?" in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2009, pp. 55–62.
[2] J. Kramer, "Is abstraction the key to computing?" *Communications of the ACM*, vol. 50, no. 4, pp. 36–42, 2007.
[3] O. Hazzan, "Reflections on teaching abstraction and other soft ideas," *ACM SIGCSE Bulletin*, vol. 40, no. 2, pp. 40–43, 2008.
[4] L. Saitta and J.-D. Zucker, *Abstraction in artificial intelligence and complex systems*. Springer, 2013, vol. 456.
[5] G. Engels, J. H. Hausmann, M. Lohmann, and S. Sauer, "Teaching uml is teaching software engineering is teaching abstraction," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2005, pp. 306–319.
[6] J. Börstler, L. Kuzniarz, C. Alphonce, W. B. Sanders, and M. Smialek, "Teaching software modeling in computing curricula," in *Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups*. ACM, 2012, pp. 39–50.
[7] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2013, pp. 1–17.
[8] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.
[9] R. Duval, "A cognitive analysis of problems of comprehension in a learning of mathematics," *Educational studies in mathematics*, vol. 61, no. 1-2, pp. 103–131, 2006.
[10] P. White and M. C. Mitchelmore, "Teaching for abstraction: A model," *Mathematical Thinking and Learning*, vol. 12, no. 3, pp. 205–226, 2010.
[11] J. Börstler, L. Kuzniarz, C. Alphonce, W. B. Sanders, and M. Smialek, "Teaching software modeling in computing curricula," in *Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups*. ACM, 2012, pp. 39–50.
[12] R. F. Paige, F. A. Polack, D. S. Kolovos, L. M. Rose, N. Matragkas, and J. R. Williams, "Bad modelling teaching practices," in *Proceedings of the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS14), Valencia, Spain*, 2014.
[13] S. Akayama, B. Demuth, T. C. Lethbridge, M. Scholz, P. Stevens, and D. R. Stikkolorum, "Tool use in software modelling education." in *EduSymp@ MoDELS*, 2013.
[14] G. Liebel, R. Heldal, J.-P. Steghöfer, and M. R. Chaudron, "Ready for prime time,-yes, industrial-grade modelling tools can be used in education," *Research Reports in Software Engineering and Management No. 2015:01*, 2015.
[15] D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," *Theory into practice*, vol. 41, no. 4, pp. 212–218, 2002.
[16] D. A. Kolb, *Experiential learning: Experience as the source of learning and development*. FT press, 2014.
[17] J. M. Keller, *Motivational design for learning and performance: The ARCS model approach*. Springer Science & Business Media, 2010.
[18] T. Stahl and M. Volter, *Model-driven software development*. J. Wiley & Sons, 2006.
[19] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, "Eugenia: towards disciplined and automated development of gmf-based graphical model editors," *Software & Systems Modeling*, pp. 1–27, 2015.

[20] D. Kolovos, L. Rose, R. Paige, and A. Garcıa-Domınguez, “The epsilon book,” *Structure*, vol. 178, pp. 1–10, 2010.

[21] M. Csikszentmihalyi, “Toward a psychology of optimal experience,” in *Flow and the foundations of positive psychology*. Springer, 2014, pp. 209–226.

[22] D. S. Kolovos, R. F. Paige, and F. A. Polack, “Eclipse development tools for epsilon,” in *Eclipse Summit Europe, Eclipse Modeling Symposium*, vol. 20062, 2006, p. 200.