Leveraging Model-Driven Engineering in the Design and Management of Professional Education Programmes

Dennis Wolters, M. Sc.

Supervisor: Prof. Dr. Gregor Engels

Department of Computer Science
Paderborn University

This dissertation is submitted for the degree of
doctor rerum naturalium (Dr. rer. nat.)

December 2023

To my wife and my daughters.

Acknowledgements

In the course of my PhD journey, I have had the pleasure of working with great people to whom I would like to express my gratitude. There are too many to name everybody explicitly. Still, I hope everyone will recognize themselves as part of the groups mentioned.

First and foremost, I would like to thank my supervisor and mentor, Prof. Dr. Gregor Engels. Gregor, this work would not have been possible without your support and advice. I am deeply grateful that you have always trusted me, encouraged me, shared your experiences, and enabled me to find and follow my own path. I would also like to thank my second supervisor, Prof. Dr. Stefan Böttcher. Stefan, you have already accompanied me throughout my studies, and I am very grateful that you were also part of this milestone. I would also like to thank the other members of my examination board: Prof. Dr. Daniel Beverungen, Prof. Dr. Christian Gerth and Dr. Simon Oberthür. Christian and Simon, I am so grateful to you for your guidance, our discussions, and the many things I have learned from you over the years.

I would also like to thank all my current and former colleagues at Paderborn University for the conversations, whether professional or private, the time spent together and the collaboration. In particular, I thank Dr. Stefan Heindorf and Dr. Jonas Kirchhoff, with whom I have worked on many different publications and with whom I had the pleasure of sharing an office. At the same time, I would also like to thank Sonja Saage and Friedhelm Wegener. Your willingness to help and your commitment to others are inspiring. A big thank you also goes to my external colleagues Amanda Bamford, Martin Rolf, Judith Shawcross, Peter Thornton and Ramon van Knippenberg. I have learned so much from you and grown personally and professionally thanks to you. We have achieved great things together, and it has always been fun.

If you want to tackle big projects, you can only do so if you have a solid foundation. I am fortunate to have a wonderful family that always stands behind me and that I can always rely on. From my mother and my sisters, my godfather, and my parents-in-law to the most precious people in my life: My

wife Anne and my daughters. You are always there for me and support me in everything I do. I would not be the person I am today without you. Especially without my wife, I would not have been able to complete this thesis. Anne, you were there every step of the way. You motivated me when I faced setbacks, celebrated every success with me, and gave me the strength to complete this journey successfully. For that, I am forever grateful, and I love you.

Abstract

With the rapid evolution of the IT sector driving Digital Transformation, professional education is more critical than ever. There is a pressing need to upskill and reskill employees since routine tasks are automated, and tech literacy is becoming increasingly important in every role. Professional education helps businesses to adapt to the resulting technological shifts and plays a pivotal role in employee motivation, retention and recruitment. Companies rely on education providers for learning resources as well as entire professional education programmes.

In this thesis, we leverage techniques from Model-Driven Engineering to assist education providers in designing and managing professional education programmes. We provide the basis for remote collaborative design in online whiteboards as well as work and knowledge management with our Professional Education Programme Modelling Language (PEPML). In contrast to other education modelling languages, PEPML is extensible, covers all life cycle phases of such programmes and allows tracking work items relative to programme elements. PEPML does not just have a whiteboard-compatible visual syntax, but we also provide an extension to the online whiteboard Miro for further visual modelling tool support. In particular, we enable extracting PEPML models from Miro boards by using Triple Graph Grammars (TGGs) for model transformation. We combine this with a model merging approach to enable collaboration across more than one Miro board and to support multiple modelling viewpoints. We reduce the effort of writing the required TGGs by introducing an approach to generate them based on mappings between the relevant metamodels. We leverage the same TGG-based technique to enrich PEPML models based on information provided by other tools used to design or manage professional education programmes. Furthermore, we present a knowledge management approach based on PEPML that can provide situation-specific recommendations to education providers to increase programme quality.

Zusammenfassung

Durch die rasante Entwicklung des IT-Sektors und der daraus resultierenden Digitalen Transformation ist berufliche Bildung wichtiger denn je. Es besteht ein dringender Bedarf, Mitarbeitende fort- und weiterzubilden, da routinemäßige Aufgaben automatisiert werden und digitale Kompetenz in jeder Position zunehmend an Bedeutung gewinnt. Berufliche Weiterbildung hilft Unternehmen, sich an die daraus resultierenden technologischen Veränderungen anzupassen und spielt eine entscheidende Rolle bei der Motivation und Bindung von Mitarbeitenden. Unternehmen sind auf Bildungsanbieter angewiesen, um Lernressourcen sowie gesamte Bildungsprogramme bereitzustellen.

Diese Arbeit zeigt wie Techniken des Model-Driven Engineering bei der Konzeption und Organisation von beruflichen Bildungsprogrammen unterstützen können. Die Grundlage ist die Modellierungssprache PEPML, die eine kollaborative Gestaltung solcher Programme auf Online-Whiteboards sowie die Verwaltung von relevantem Wissen und Arbeitsabläufen ermöglicht. Im Gegensatz zu anderen Educational Modelling Languages ist PEPML erweiterbar, adressiert alle Lebenszyklusphasen von Bildungsprogrammen und erlaubt die Verwaltung von Arbeitsaufgaben in Bezug auf einzelne Programmbestandteile. Eine Erweiterung für das Online-Whiteboard Miro unterstützt bei der visuellen Modellierung mit PEPML. Mit Hilfe von Triple-Graph-Grammars (TGGs) und Model Merging wird die Modellierung von Bildungsprogrammen über mehrere Modellierungsperspektiven und Whiteboards hinweg ermöglicht. Der Aufwand zur Erstellung der benötigten TGGs für die Modellextraktion aus Whiteboards wird durch einen Ansatz zur TGG-Generierung auf Basis von Verknüpfungen zwischen den relevanten Metamodellen reduziert. Informationen aus anderen Software-Anwendungen, die für die Gestaltung von Bildungsprogrammen genutzt werden, können mit derselben TGG-basierten Technik in bestehende PEPML-Modelle integriert werden. Basierend auf diesen Modellen können Bildungsanbietern situationsspezifische Empfehlungen zur Steigerung der Programmqualität gemacht werden.

Table of Contents

Chapter 1

Introduction

Education providers must overcome various challenges to design and deliver high-quality, bespoke professional education programmes to their clients. This thesis proposes to apply techniques from Model-Driven Engineering (MDE) to address these challenges and assist education providers in the design and management of such programmes.

Section 1.1 motivates the need for professional education programmes. Section 1.2 describes the challenges of designing and managing such programmes and derives a problem statement. Section 1.3 subdivides the problem statement into four research questions. Section 1.4 presents a solution overview of the approach presented in this thesis. Section 1.5 gives an overview of publications created in the course of this thesis and their relation to our approach. Section 1.6 explains this thesis structure.

1.1 Motivation

There are several different reasons that require today's companies to invest in professional education. A selection of these reasons is visualized on the left of Figure 1.1. We spotlight them in this section to motivate the need for professional education programmes.

The advances in IT lead to Digital Transformation, which promises increased efficiency or new business models via digital solutions [WSE18]. Companies must embrace this transformation to stay competitive [Kre17]. To do so, companies partner with universities and innovation centres and use professional education to prepare their employees and businesses for Digital Transformation [ABC+21].

Digital Transformation

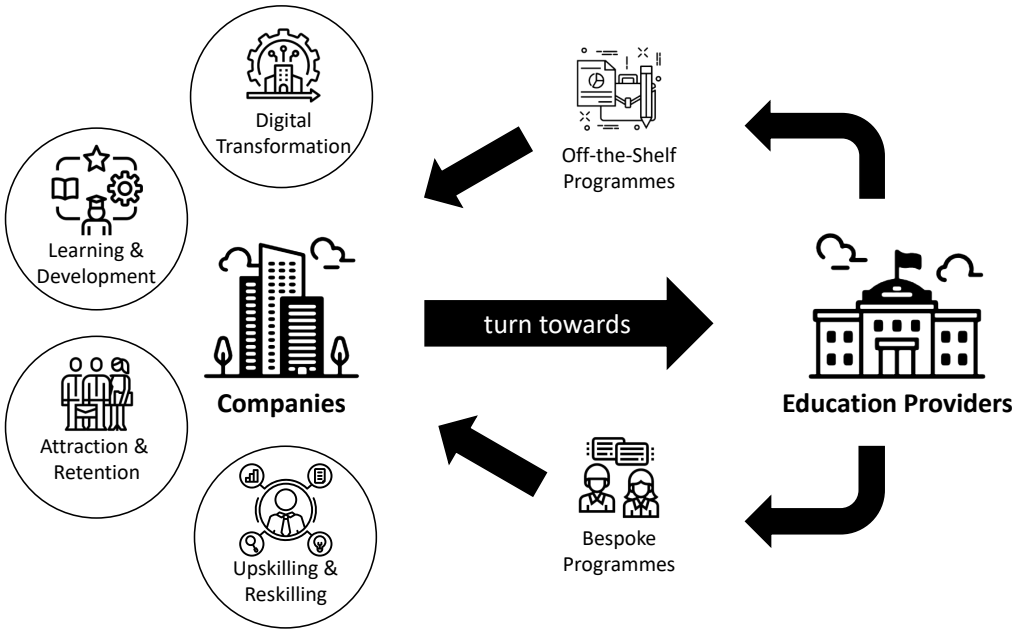

Figure 1.1: Motivation for the need for professional education programmes

Upskilling & Reskilling

Digital Transformation and emerging technologies, like generative AI, impact all jobs. For instance, certain tasks or even complete jobs may be automated. Companies must reskill and upskill their employees through professional education to deal with these changes. Li [Li22] highlights the necessity for upskilling and reskilling in the manufacturing industry. She states that a skill shift is needed for a modern manufacturing workforce, especially towards technological competency. Furthermore, life-long learning should be part of every company's organizational strategy to cope with future changes.

Learning and Development

In general, Learning and Development (L&D) is an essential organizational process in most companies, not only because of Digital Transformation. L&D is concerned with developing employees, individually and collectively, to increase business performance [Har05]. Companies can only address some educational needs internally and rely on external education providers to provide L&D opportunities to a wide range of employees from different areas, e.g. accounting, IT, sales and other areas.

Motivate, Attract and Retain Employees

Professional education is more than just a way to close a skill gap. Professional education offerings can be used to show appreciation and broaden the horizons of employees. This appreciation and investment motivate employees and increase retention [BBH+08, KDMM09]. Retaining people in IT positions is especially crucial due to the shortage of IT professionals [Bit22]. Moreover, showing clear career paths and development opportunities can be a differentiator when attracting new talents [HBVW08].

As mentioned, companies can only address parts of their professional educational needs internally and turn towards education providers to provide professional education. They can request educational content on specific topics or professional education programmes, which structure activities to reach specific educational goals. As depicted in Figure 1.1, education providers can offer off-the-shelf (OTS) or bespoke education programmes. OTS content/programmes are created for a specific market and are the same for all participants, independent of their employer. Bespoke content/programmes are created for a specific company or group of companies and are tailored to their needs. This thesis focuses on the design and management of bespoke professional education programmes. In the subsequent section, we explain the challenges when designing and running such programmes.

off-the-shelf vs. bespoke

1.2 Challenges

In this thesis, we focus on providers of bespoke professional education programmes. This section explains education providers' challenges when designing and managing these programmes. Each challenge presented in this section is given a name in the sidenotes for later referencing.

Focus on Education Providers

Developing bespoke professional education programmes requires education providers to collaborate internally and with their clients to deliver high-quality programmes. Education providers must understand a company's environment, educational needs and the programme's intended goals. Collaboration with companies is broader than gaining information and making decisions. Instead, companies play an active role in designing and delivering programmes. For instance, they handle the promotion of the programme internally, coordinate participant selection or provide educators themselves.

Collaborative Design and Management

If professional education programmes are not developed by an internal department but via an external education provider, spatial separation between them is common. Before the COVID-19 pandemic, physical meetings were dominant, but the pandemic changed the way of working [MF20]. Home office and online meetings are now more common than before the pandemic. Even employees of the same company are not necessarily in the office on the same day, and the willingness to travel for short meetings decreased [SCTR22]. Consequently, education providers must have suitable means to collaborate remotely with their clients on the design and management of a programme.

Challenge CH1: Remote Collaboration

The field of Instructional Design is concerned with the development of educational resources and systems [GWGK05]. Education providers can rely

on Instructional Design methods to design a programme, which define essential activities like describing the target audience or discussing learning objectives. Work must be organized so that education providers can identify tasks related to a specific programme part. Otherwise, getting an overview of the state of a programme is difficult, which can negatively impact a programme's quality.

Challenge CH2: Work Organization

Many educators are involved, especially for more extensive programmes or those with a broad range of topics, which further complicates collaboration and work organization. The type of education provider can also influence how many educators are involved in a programme. For instance, universities are education providers that have access to a high number of educators with deep expertise in specific fields. While transferring knowledge from research into industry is among the objectives of universities and professional education programmes are a way of doing this, universities focus on research and educating students. Consequently, potential educators typically have limited availability for professional education activities, and programmes involve multiple educators to cope with limited availability and, if necessary, to cover a broader range of topics. Involving many educators makes the work organization even more complicated since education providers must ensure that all educators are kept up-to-date about the programme and their tasks.

Challenge CH3: High Number of Educators

While Instructional Design methods support work organization by defining proven roles, activities and tools, there is no one-size-fits-all regarding (development) methods. Method adaption or selection based on the situation is necessary [HSRÅR14]. For example, the choice of a development method may depend on the willingness of the client to experiment. Similarly, delivery methods depend on the situation, e.g. the venue or delivery mode. Using inappropriate methods can reduce programme quality and increase costs.

Challenge CH4: No one-size-fits-all methods

There is no single tool for every purpose when designing and running (professional) education programmes. Instead, a set of tools is necessary, especially when considering the whole life cycle of a programme. For instance, different tools are required to schedule sessions, maintain an overview of work items or prepare specific material. The usage of multiple tools results in the spread of information about programmes across these different tools. Governing these tools and the spread of information is challenging and maintaining a complete overview is difficult.

Challenge CH5: Information Spread Across Tools

Running another iteration of a programme is not just a simple repetition of the same tasks. On the one hand, programmes can change between iterations, e.g. due to participant feedback or changing client needs. On the other hand, education providers gain knowledge with each new iteration and cohort,

which influences future iterations. The knowledge that is gained is broad. For instance, education providers better understand their audience's specific challenges, which can lead to a redesign to improve learning outcomes or the learning experience. It could also be administrative knowledge about running a programme at a specific location or using a particular tool. The longevity of programmes also makes a change of personnel likely. Hence, knowledge has to be appropriately externalized to be preserved over time.

Challenge CH6: Externalizing Knowledge

In the rarest cases, education providers start entirely from scratch when creating a new programme. Existing programmes, bespoke or off-the-shelf, can include valuable parts that can be incorporated into a new programme if the objectives align. However, with an increasing number of programmes and iterations, it is difficult to get an overview of existing elements of education programmes. Furthermore, dependencies within existing content must be considered, e.g. a session from another programme might depend on other sessions within that programme. Proper cataloguing of existing programme elements for later reuse in the design is required.

Extension CH6: Catalogue Existing Programme Elements

Many methods to design and develop educational content and programmes exist within the field of Instructional Design [GWGK05]. Programme designers are not always trained in this field or have practical expertise [Paq04]. So, they do not always know existing methods or best practices. Even experienced designers may not always recall every lesson already learned, externalized or not, due to the workload. Hence, it is not only crucial to externalize knowledge but also to recall knowledge in relevant situations.

Challenge CH7: Recalling Relevant Knowledge

Based on the described challenges, we define our problem statement:

How to support education providers in the situation-specific design and management of bespoke professional education programmes?

Problem Statement

This problem statement reflects our focus on education providers and bespoke professional education programmes. The support of the design of such programmes encompasses a collaborative design that involves clients (CH1), collaborating with multiple educators contributing to a programme (CH3), using appropriate methods (CH4) and incorporating existing knowledge (CH7). Management reflects the aspect of work organizations in all phases (CH2) as well as governing tools and information (CH5) and the externalization of knowledge (CH6).

1.3 Research Questions

This section breaks down the problem statement into several research questions. The relation between the challenges defined in the previous section and the research questions is visualized in Figure 1.2.

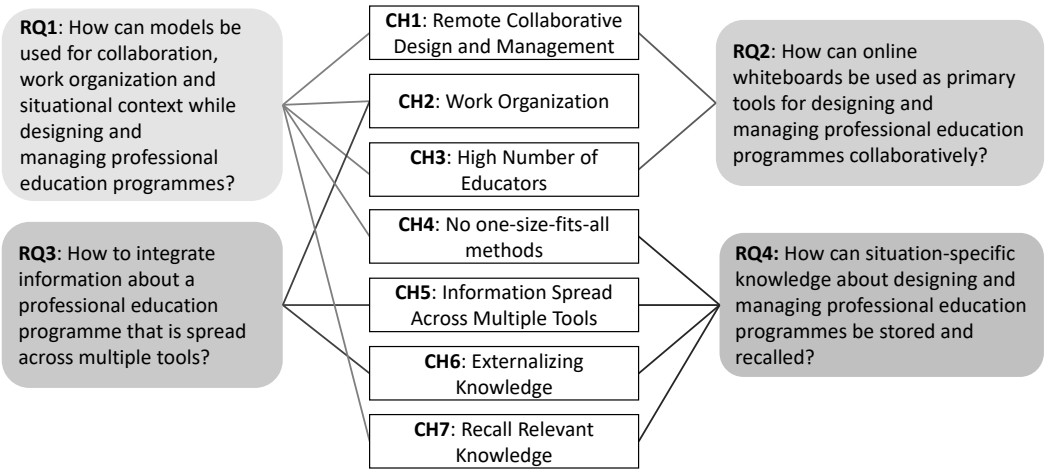

Figure 1.2: Relations between challenges and research questions

Models as mediating artefacts

Challenge CH1 highlights the necessity to collaborate with clients to develop bespoke programmes. Models are known to be good artefacts to facilitate discussions and assist in design [Pet13, Con09]. Several modelling approaches have been presented for Instructional Design [BBCR+08]. As we explain in more detail in upcoming chapters, the design and management processes surrounding education programmes are beyond the scope of existing education modelling languages. Hence, they do not assist in work organization (cf. CH2), and education providers must manage these work items separately, e.g. using project management tools. Since work items can relate to different programme entities, education providers must maintain a manual mapping between existing programme entities and their corresponding work items. In this thesis, a *programme entity refers to any part of a programme*, e.g. a teaching session, an educator or a learning objective. Project management tools support clustering work items in the form of hierarchies and labels, but this has to be set up and maintained manually. Manual mappings are complex to maintain, especially for programmes with a changing structure and many sessions by different educators (cf. CH3). Education providers must generally maintain an overview of a programme to react to a given situation, e.g. by selecting appropriate methods (cf. CH4). A model providing such an overview can serve as a context model for situation-specific assistance (cf. CH7).

Current Models Do Not Assist in Work Orgainzation

Situational Context

The first research question addresses how models of professional education programmes can be used for such purposes:

Research Question 1 (RQ1)

How can models be used for collaboration, work organization and situational context while designing and managing professional education programmes?

The modelling tool support for existing education modelling languages is often outdated and not web-based. Consequently, these tools cannot be used for remote collaborative design with multiple people (cf. CH1 and CH3). Collaborative online whiteboards are valuable tools for remote co-design activities [MVMMV15, BAHT21, LZXL21]. While such online whiteboards might allow modelling with existing (education) modelling languages, they do not enforce that users follow the syntax rules of these languages. From the software engineering domain, we know that modelling tools can be powerful primary design tools. Unfortunately, the models created on online whiteboards are informal and board content can only be exported without machine-interpretable semantics, which hinders further processing. Hence, online whiteboards are currently auxiliary tools for short-term collaborative work. The second research question addresses this issue:

Online Whiteboards are Auxiliary Tools

Research Question 2 (RQ2)

How can online whiteboards be used as primary tools for designing and managing professional education programmes collaboratively?

Modelling professional education programmes can be very beneficial for collaborative design. However, models are an abstraction, and more detailed information is often spread across the tools used to develop and run a programme (cf. CH5). Having an integrated view of information about a programme and knowing where information is stored can assist in work organization (cf. CH2) and externalizing knowledge (cf. CH6). For instance, if the information about a programme's structure can be combined with the information on open tasks in a project management tool, programme designers could better understand which parts still require work. Integrating information about a programme is the core of the third research question:

Information spread across tools needs to be integrated

Research Question 3 (RQ3)

How to integrate information about a professional education programme that is spread across multiple tools?

Challenge CH4 mentions that methods must be selected and composed based on the current situation. In our case, we do not only need methods

for developing professional education programmes but also to manage their delivery. Such methods may be developed during the design of a programme and have to be externalized for later reuse, alongside knowledge about existing learning resources and best practices (cf. CH6 and CH7). Recalling relevant knowledge requires to understand the current situational context. Getting a situational overview is complicated since information about such programmes is spread across multiple tools (cf. CH5). The storing and recalling of situation-specific knowledge for professional education programmes is the subject of the fourth and last research question:

Research Question 4 (RQ4)

How can situation-specific knowledge about designing and managing professional education programmes be stored and recalled?

1.4 Solution Overview

In the following, we explain how we leverage techniques from Model-Driven Engineering to assist education providers in designing and managing professional education programmes. Figure 1.3 shows how the different parts of our solutions are connected.

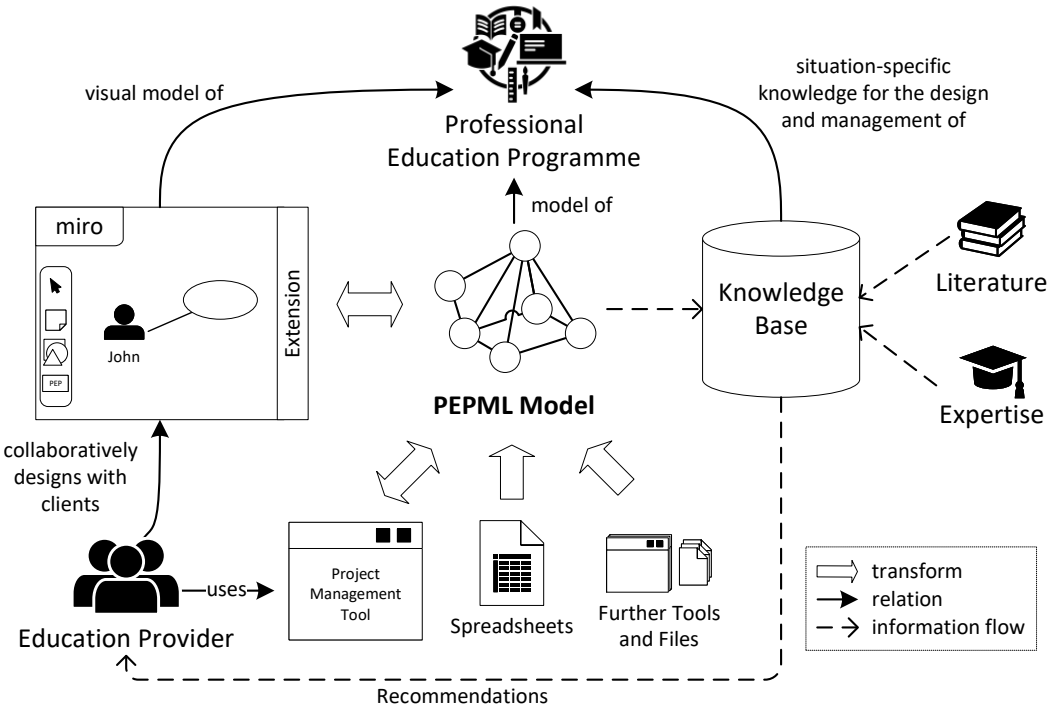

Figure 1.3: Overview of our solution for assisting education providers in the design and management of professional education programmes

The core of the solution presented in this thesis and a response to research question RQ1 is our Professional Education Programme Modelling Language (PEPML). PEPML is an extensible language that can describe professional education programmes on a high abstraction level across their entire lifecycle. In contrast to other education modelling languages, PEPML supports the definition of work items for any programme entity and can serve as a basis for work organization. The language has a visual syntax specifically designed to be used in online whiteboards. Thereby, it serves as a basis for remote collaborative design.

PEPML

We provide an extension to the online whiteboard Miro[1], which provides dedicated modelling support for PEPML. This extension allows the use of Miro beyond informal modelling and makes it a primary tool for modelling professional education programmes, which addresses research question RQ2. In particular, we use model transformations based on Triple Graph Grammars (TGGs) to transform Miro board content to a PEPML model and vice versa. We allow partial modelling of professional education programmes on whiteboards and employ a model merging approach to create a combined model. Thereby, we enable complexity management, support for multiple viewpoints and even collaboration across multiple boards.

Collaborative Visual Modelling

It cannot be expected that education providers visually model all information, especially when it is already present in other tools/files. We utilize the model transformation and merging approach used by our visual modelling tool support to also integrate the information offered by other tools and files containing semantically relevant information. We illustrate our model-driven information integration technique for a project management tool and spreadsheets, but it can be used beyond those two examples and addresses research question RQ3. The focus is on unidirectional information integration since bidirectional transformation between an arbitrary number of tools is too complex. Yet, the integration with the example project management tools is bidirectional to strengthen the work management capabilities of our approach. By enabling information integration, users do not have to switch to new tools.

Information Integration

To answer research question RQ4, we propose a knowledge base to provide situation-specific recommendations for designing and managing professional education programmes. The knowledge base is filled based on PEPML models of existing programmes, expertise of education providers and literature. Knowledge items can be attached to situations in which they apply. A PEPML model of a programme is then used to determine which knowledge is relevant.

Situation-specific Recommendations

[1] https://miro.com

1.5 Publication & Project Overview

This section explains how the publications created in the course of this PhD thesis and the projects the author was involved in are related to the presented approach. Some of these publications are directly related to the approach, while others are indirectly related as they familiarize the author with the techniques used or the context of the thesis.

An overview of PEPML and its integration with project management tooling is given in [WE22a]. Details on PEPML and the development of visual modelling tool support based on Miro are the subject of [WE23]. In [WE22b], we explained the extension of our work to provide situation-specific recommendations.

Experience in creating visual modelling approaches and combining multiple viewpoints was gained in several prior publications. In [BGE14], we presented an extension of the Business Process Model and Notation (BPMN) to denote device usage in processes. Also, this work provided insights into the field of (business) process management, which influenced this thesis's situational assistance approach. We extended UML Use Case Diagrams (UCDs) in [WGE16] to model device usage on a high level of abstraction. We combined the extension of UCDs with the extension of BPMN in [WGE17] to create a visual requirements modelling approach for cross-device systems that provides different perspectives on such systems.

The tooling for the approach presented in this thesis is based on Web technologies and integrations with existing tools. Prior research concerned integrating applications across device boundaries [WKGE16a] and leveraging semantic data in web pages to integrate applications [WHKE17]. The tooling developed for these approaches is also heavily web-based [WKGE16b, WHKE18]. Additionally, we extended the OpenAPI specification to specify HTTP interfaces for command-line applications [WKE19], which led to knowledge in designing, describing and using HTTP APIs.

The didactic background partially stems from completing the certificate programme "Professionelle Lehrkompetenz für die Hochschule". The last stage of this programme included a didactic research project, where the effect of the Peer Instruction method and the use of Classroom Response Systems in a computer science bachelor course was investigated [Wol18]. This research project won the teaching award from Paderborn University in 2017. This work also showed how closely the educational situation must be observed to select the right tools.

The practical domain knowledge in the area of professional education programmes is grounded in the work on the Atos GOLD for Technology Leaders (GfTL) programme and the Miele IT Professionals Program (MIPP). Both are talent development programmes for technology leaders. Atos GfTL is jointly delivered by Paderborn University and IfM Engage[2], a knowledge transfer company of the Institute for Manufacturing (IfM) at the University of Cambridge. The programme won the EFMD Excellence in Practice Silver Award in 2018 in the "Professional Development" category[3].

1.6 Structure of this Thesis

The following gives an overview of the upcoming chapters and how to read this thesis. The structure of this thesis is visualized in Figure 1.4, which relates the chapters to the research questions and highlights the important parts.

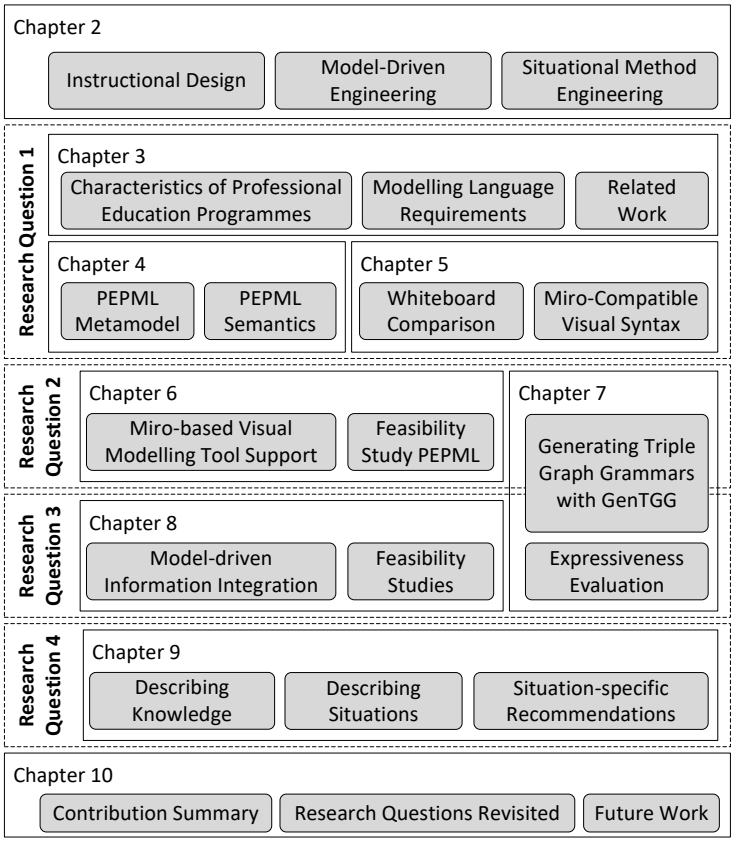

Figure 1.4: Thesis structure overview

[2] https://engage.ifm.eng.cam.ac.uk

[3] https://efmdglobal.org/awards/eip-excellence-in-practice-award/efmd-excellence-in-practice-award-2018-winners

Chapter 2 provides the necessary fundamentals for this thesis. The approach presented in this thesis is located at the intersection of three research areas: Instructional Design, Model-Driven Engineering and Situational Method Engineering. Consequently, the chapter covers the fundamentals of all three areas. Related work is covered separately in the chapters that follow.

The next three chapters introduce our modelling approach for professional education programmes and its tool support. Chapter 3 presents characteristics of professional education programmes, describes requirements for a modelling approach and discusses related work. Chapter 4 introduces the metamodel of PEPML, our modelling language for professional education programmes. Chapter 5 compares online whiteboard solutions and their suitability for visual modelling tool support. We explain why we focus on the online whiteboard Miro in this thesis and define a Miro-compatible visual syntax for PEPML.

Chapter 6 describes our Miro-based visual modelling tool support and how we leverage model transformations based on TGGs combined with a model merging approach to extract PEPML models from Miro boards.

Chapter 7 presents GenTGG, an approach to generate TGG rules based on mappings between two metamodels. GenTGG is used to generate the TGG required for extracting PEPML models from online whiteboards. The chapter also includes an evaluation of the expressiveness of GenTGG to show its applicability beyond the use cases of this thesis.

Our whiteboard-based modelling approach can be used to build PEPML models, but several other tools are used to design and manage professional education programmes. Chapter 8 explains how we leverage the technical infrastructure used to obtain PEPML models from online whiteboards and GenTGG to build up such models based on information provided by other tools. We show the technical feasibility by integrating information from a project management tool and spreadsheets.

Chapter 9 presents our approach to providing situation-specific recommendations while designing and managing professional education programmes. It covers how knowledge can be described and stored with respect to the situations in which it is applicable. Furthermore, we explain how PEPML models can be used as situational context to identify if knowledge is relevant.

Chapter 10 concludes this thesis by summarising its contributions and revisiting the research questions. Finally, we give an outlook on future work.

Chapter 2

Fundamentals

This thesis is concerned with providing education providers with situation-specific assistance while designing and managing professional education programmes. The creation of education programmes is one of the subjects within the field of Instructional Design. The techniques we apply within our approach stem from the fields of Model-Driven Engineering (MDE) and Situational Method Engineering (SME). Consequently, this thesis is placed in the intersection of these three fields (cf. Figure 2.1), and the fundamentals of each field are necessary to understand this thesis.

Section 2.1 describes the basics of Instructional Design by introducing important terms, common assumptions about the field and well-known Instructional Design processes. Section 2.2 focuses on MDE and explains what modelling is, how to engineer modelling languages and how to use model transformations for model manipulation. Section 2.3 covers the fundamentals of SME by clarifying the terminology, explaining which types of SME approaches exist and focusing on assembly-based method engineering in particular.

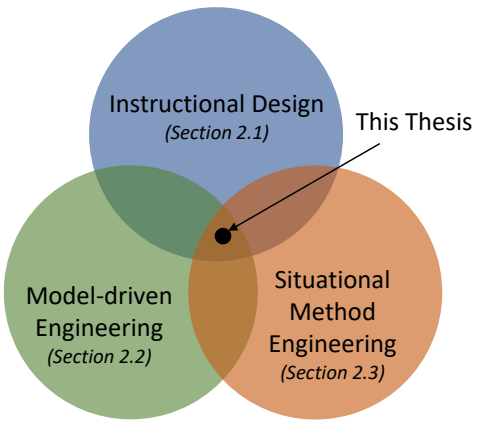

Figure 2.1: Placement of this thesis in existing research fields

2.1 Instructional Design

This section provides an introduction to Instructional Design and is split into three subsections: Section 2.1.1 explains background information and relevant terminology. Section 2.1.2 presents common assumptions about Instructional Design. Section 2.1.3 covers Instructional Design processes that are concerned with the creation of education programmes.

2.1.1 Terminology

Instructional Design is the discipline and research field of creating instructions for intentional learning [GWGK05]. Intentional learning is about active and conscious participation in a goal-oriented learning process. For instance, taking part in an onboarding programme that aims to familiarise participants with corporate procedures would be an example of intentional learning. In that scenario, the learner is actively aware of the learning intentions and is given instructions that help to reach the learning goals. Instructions are the planned events that support intentional learning, e.g. listening to a lecture, reading a book or participating in group work. In contrast, an example of unintentional learning would be that one might learn about corporate procedures by listening to colleagues during a coffee break.

Instructional Design

The field of Instructional Design offers a broad range of models, processes, methods and principles, e.g. to describe how people learn [GWGK05], distinguish different levels of evaluations [Kir98] or to design instructional systems [BK14]. The latter is the most common conception of Instructional Design. Partially, the term Instructional Systems Design is used to emphasise the focus on the design of instructional systems. While we mainly focus on Instructional Systems Design in this thesis, we use the term Instructional Design throughout this thesis to acknowledge the use of other models in the system's design.

Instructional Systems Design

Alternative terms for Instructional Design exist, like Learning Design or Education Design. While some use them as synonyms, others try to differentiate between these terms. A recent comparison from Begüm et al. [BBE21] showed the overlap of these terms and concluded that Instructional Design takes a theory-driven approach utilizing models and frameworks. In contrast, Learning Design shifted towards focusing on the learner's experience. However, some theory-driven approaches exist that are labelled as Learning Design approaches. One prominent example is the IMS Learning Design (IMS-LD)

Instructional vs. Learning vs. Education Design

specification [IMS03]. To emphasize the focus on the learner's experience and to be different from previous terms, the term learner experience design can also be found [TGGLH21]. Another reason for the difference in terminology may be a geographical one because, as Goddard et al. [GGM15] point out, Instructional Design is rather used in North America, and the term Learning Design is more dominant in Europe. Since a clear distinction between these terms is not commonly defined, **this thesis treats Instructional Design, Learning Design and Education Design as synonyms**. We use the term Instructional Design for our own work, which is fitting in the sense that we are utilizing models for designing and managing professional education programmes. When discussing related work, we use the terms used in the respective work.

As mentioned, Instructional Design is concerned with the creation of instructional systems. Gagné et al. [GWGK05] define an instructional system as "an arrangement of resources and procedures to facilitate learning". The public school system is one example of a large-scale instructional system. The instructional systems we are concerned with are education programmes, which the International Standard Classification of Education (ISCED) [UNE11] defines as "a coherent set or sequence of educational activities designed and organized to achieve pre-determined learning objectives or accomplish a specific set of educational tasks over a sustained period." Education programmes are a subtype of instructional systems. We provide further information on (professional) education programmes in Section 3.1.1.

Instructional Systems and Education Programmes

An education provider is an entity that develops and runs an instructional system. The ISCED [UNE11] defines an education provider as "an organization that provides education, either as a main or ancillary objective. This can be a public educational institution, as well as a private enterprise, non-governmental organization or non-educational public body." This definition highlights that education providers are not limited to public educational institutions like schools or universities but that private companies can also be education providers. In primary and secondary education, public education providers are the most dominant and private education providers are mostly suppliers, e.g. of learning material. Private education providers play a more dominant role in professional education. Several companies provide education programmes on their products, such as Google's education programmes on their Cloud offering. Other private education providers provide education programmes for specific topics, such as personal development. Universities mainly focus on educating students but also transfer knowledge from research into practice via professional education programmes [ABC+21].

Education Providers

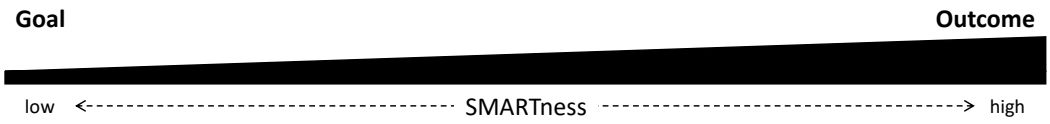

Figure 2.2: Spectrum of objectives

Different terms refer to the objectives of an education programme, e.g. goals, aims or outcomes. Airasian et al. [AKA+01] provide a thorough description of objectives in educational settings. They acknowledge the differences in terminology and use specificity as a dimension to identify different objectives. In this thesis, we use the term "objectives" as a general term for any kind of objective and define a spectrum of objectives for this thesis (see Figure 2.2). We position goals and outcomes at opposite ends of this spectrum and use SMARTness as a basis for our spectrum. SMART is an acronym for Specific, Measurable, Assignable, Realistic and Time-related [Dor81] and is widely used for defining objectives. If the SMARTness is low, meaning the objective is broad, we call it a goal. A basic level of SMARTness is needed even for goals as completely unspecific, non-measurable and unrealistic goals do not provide any value. On the other end of the spectrum are outcomes, which have a higher specificity and should be measurable.

Spectrum of Objectives

2.1.2 Common Assumptions

While the individual perception of Instructional Design may differ, certain assumptions are shared throughout the field. In their seminal book on Instructional Design, Gagné et al. [GWGK05] describe a set of common assumptions that we summarize and relate to other concepts of the field in the following.

Instructional Design entails various models for designing instructional systems. However, Gagné et al. [GWGK05] note: *"It would be a mistake to think that there is a single best model for Instructional Design. In actuality, there are as many models as there are designers and design situations. Each designer brings to the process his and her understanding of the principles and events that affect learning, and how to best structure instructions."* This quote is central to this thesis. It underlines that there is no single best process/model for Instructional Design. Furthermore, the expertise of designers and the situation influence the Instructional Design process and the instructions necessary. With the approach described in later chapters of this thesis, we provide a framework that allows education providers to externalize parts of their expertise and assists them in finding design and delivery processes that fit their situations.

No single best model

Instructional Design is not just about teaching but focuses on learning. Teaching is only one part of the learning process. In addition, it is necessary to understand the learning process, the learners and their environment to provide effective instructions. Furthermore, learners shall be actively involved in the learning process rather than just passively receiving instructions, e.g. listening to a lecture. A focus on learning over teaching does not mean that teaching is not important. It is merely one part of the learning process.

Focus on Learning over Teaching

One of the reasons why there cannot be a single best model for creating instructions is that learning is complex and affected by many interdependent variables. For instance, the quality of the instructions, the time given to learners, their prior knowledge and motivations influence the learning outcome. A single model can only cover some variables, and an optimal solution is unlikely. Hence, there will always be a trade-off.

Learning is Complex

Since learning is complex, creating good instructions takes time and effort. Gagné et al. [GWGK05] write that Instructional Design is not about creating perfect instructions but more about perfecting instructions over time. Consequently, Instructional Design has to be an iterative process, with the goal of improving instructions with each iteration. Constant evaluation of the teaching and learning process is necessary to measure the impact of changes.

Iterative Process

While there is no single best process, essential sub-processes and their interrelations are known, e.g. conducting a need analysis to determine programme goals. Instructional Design models exist to structure the overarching process and those defining specific sub-processes. For instance, a need analysis is a regular part of the analysis phase of an Instructional Design process.

Known Sub-Processes

Instructional Design models can also be applied at different levels or focus on specific settings. An Instructional Design process may be used to develop a single lecture or a whole programme. Similarly, Instructional Design processes exist that focus on a specific setting, e.g. the design of e-learning instructions.

Different Levels of Application

With each instructional system, specific objectives shall be reached. These objectives influence the necessary instructions. It is common practice to do a Backward Design by determining the desired long-term objectives and then working backwards to design an instructional system to reach these goals.

Backward Design

Kirkpatrick's model of evaluation [Kir98] can help with a proper Backward Design. It describes four levels on which the effectiveness of instructions can be evaluated: (1) reaction, (2) learning, (3) behaviour and (4) results. The first level, reaction, provides feedback about how the learners experience the learning process and the quality of the instructions. The second level, learning, assesses the knowledge and skills that learners have acquired. Typically, this

Kirkpatrick's model of evaluation

is done via a form of assessment, e.g. exam, test or interview. The first two levels are still within the scope of the instructional system. The third and fourth levels are observed afterwards, e.g. in the workplace. The third level, behaviour, assesses whether the learners apply the knowledge and skills in their work. The fourth level, results, assesses the impact of the learning on the organization. During Backward Design, one should start with the results that shall be achieved in the long term and then work backwards to the behaviours that are likely to lead to these results. The instructional system then needs to be designed in such a way that it fosters the intended behaviours.

Figure 2.3 visualizes Backward Design by showing Kirkpatrick's model of evaluation in combination with Constructive Alignment [Big96]. The latter emphasises that learning objectives, instructions and assessments must be aligned. Let us use driving school as an example to illustrate Backward Design and constructive alignment. The intended result of attending driving school is that students can safely participate in traffic later. The desired behaviours are that students can drive a car and can interpret street signs. Learners are likely to focus on the assessment, and if this is not aligned with the instructions and objectives, the objectives may not be reached. For instance, if the final driving test in driving school would only be an interview with the driving instructor, it is not strictly necessary to be able to drive. Similarly, the instructions and the assessments require alignment. Suppose the instructions do not prepare the learners for the assessment. In that case, they are likely to fail, e.g. theoretical lessons about driving are insufficient to prepare someone for driving a car during a driving test.

Constructive Alignment

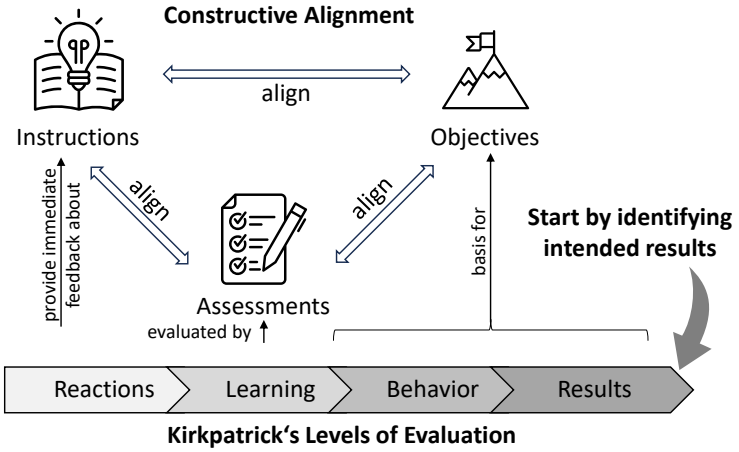

Figure 2.3: Backward Design based on Kirkpatrick's model of evaluation and Constructive Alignment

2.1.3 Instructional System Design Processes

The previously presented common assumptions are incorporated into Instructional Design processes for developing educational programmes. This subsection features three prominent examples of such processes.

ADDIE

ADDIE is the most well-known process for Instructional Design. It is an acronym for Analysis, Design, Development, Implementation and Evaluation. The client's requirements for an education programme are captured in the *Analysis* phase. The analysis is not just centred around the topics and the objectives but covers the whole context of the programme. For instance, it is crucial to understand a programme's audience and their current and desired future knowledge and skills (c.f. Backward Design in the prior subsection). As a result, learning objectives might only be relevant for a subset of the audience, or specific topics are not relevant due to existing prior knowledge. The *Design* phase is concerned with refining the objectives and planning educational activities to reach them. During the *Development* phase, the programme materials and infrastructure are developed. *Implementation* entails the preparation of educators and participants and running the education programme. The *Evaluation* phase ensures that the right programme is being designed/developed and that it leads to the intended goals.

Analysis, Design, Development, Implementation, Evaluation

The ADDIE process has been adapted multiple times, often sticking to the same name [Bra09, US 17]. Hence, concrete versions of ADDIE differ between organizations. While comprehensive descriptions exist for the ADDIE phases resulting in actionable processes [Bra09], others suggest viewing ADDIE not as a process but as a framework outlining the essential phases [GWGK05]. This framework view makes ADDIE compatible with nearly every existing process for Instructional Design. In this thesis, we favour the framework interpretation but may refer to a specific version of an ADDIE process when necessary.

Many Versions of ADDIE

Of course, different parts of a programme can be in different phases simultaneously. Consider a programme with three weeks of teaching distributed over six months: When the second week is running (Implementation), the feedback for the first week can already be analysed (Evaluation), and the third week may still be in Development.

Different Parts Different Phases

Allen and Sites [AS12] note that different perceptions of the ordering of ADDIE phases exist. Figure 2.4 shows three common visualizations of ADDIE: The left shows the waterfall-like interpretation, where all phases are ordered

Sequence of ADDIE Phases

sequentially. The middle shows the iterative version that emphasises that after the evaluation, the process continues from the beginning. The right shows that evaluation affects every other phase of ADDIE. We follow the interpretation on the right and map the steps of the two processes presented in the following to ADDIE by using the colouring of Figure 2.4.

Figure 2.4: Different visualizations of the ADDIE process model

Dick & Carey Model

The Dick & Carey Model [DCC05] is another well-known process for Instructional Design. It describes a nine-step process, which is shown in Figure 2.5. On the highest level, the Dick & Carey model is more detailed than ADDIE, but, as the colouring indicates in the figure, the steps of the process can be mapped to ADDIE phases. The first three steps relate to ADDIE's Analysis phase and are about goal identification and understanding the context. The next three steps are part of the Design phase. The seventh step maps to the Development phase. None of the steps explicitly target the implementation phase of ADDIE. Instead, this phase is implicitly part of the evaluation steps. Depending on the evaluation results, instructions may need to be revised.

Nine Step Process

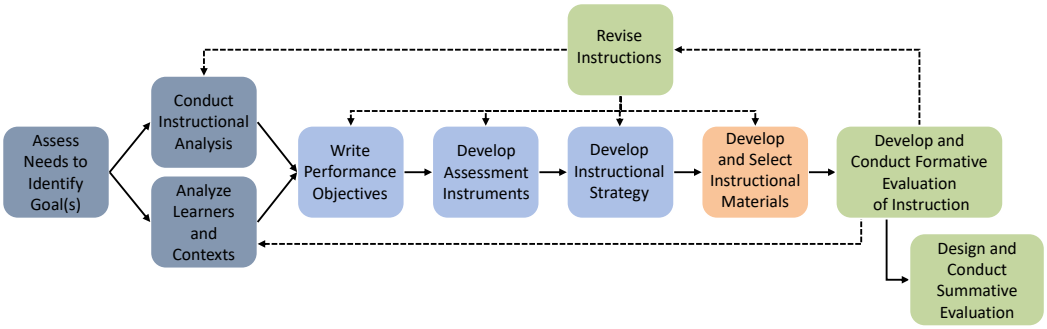

Figure 2.5: Dick & Carey Model for Instructional Design (based on [DCC05])

The Dick & Carey Model makes a differentiation between formative and summative evaluation. Formative evaluation occurs during the learning process, while summative evaluation occurs at the process' end. Both types of evaluation can be oriented in two directions. The evaluation can provide learners with feedback about their learning, e.g. progress and end result, or learners can give feedback on the learning process, e.g. how satisfied they are with the instructions. Formative evaluations of the instructions can be used to revise instruction during the learning process. Summative evaluation results can only influence the next iteration.

Formative and Summative Evluation

Successive Approximation Model

The Successive Approximation Model (SAM) is an agile process to develop educational programmes [AS12]. It applies principles from Agile software development [BBvB+01] to Instructional Design. Two versions of SAM exist: The first version is for smaller educational programmes and consists of a loop of three steps: evaluation, design and development. The second version is for more extensive educational programmes and entails the three phases depicted in Figure 2.6: In the preparation phase, information about a programme's context is gathered. Based on this information, the programme is designed iteratively in the second phase through prototyping and reviewing. Once a workable design is found, the iterative development phase starts in which the programme is developed in different maturity grades (alpha, beta, gold). In the final phase, the programme is rolled out. SAM has a focus on e-learning, which is why the process ends with the rollout.

Agile Instructional Design Process

The colouring indicates how SAM can be mapped to ADDIE phases. In contrast to ADDIE, SAM has a stronger emphasis on faster iterations and prototyping, especially in the early development of a programme.

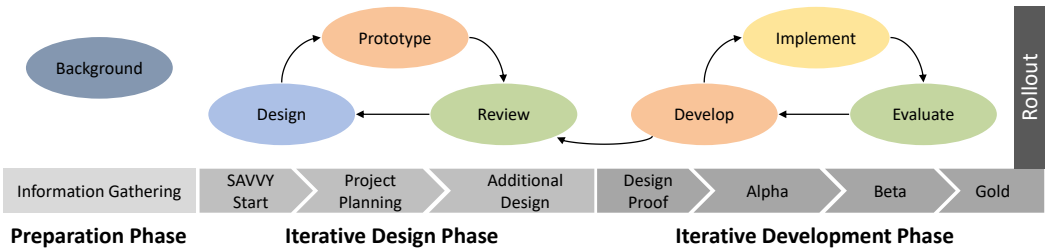

Figure 2.6: Successive Approximation Model (SAM) (based on [AS12])

2.1.4 Summary

In this section, we presented the basics of Instructional Design. In particular, we explained that the design of instructional systems like professional education programmes is within the focus of this field. We explained relevant terminology and common assumptions about Instructional Design. Additionally, we showed three well-known Instructional Design processes for designing instructional systems. Out of these three processes, ADDIE has a unique role since it can be used as a framework describing the major life-cycle phases of such programmes. Even though other processes are structurally different, their steps can still be mapped to ADDIE phases. In this thesis, we use ADDIE to refer to the phases of a programme but do not limit our approach to a specific ADDIE-based process. The common assumptions and processes provide essential information to understand the significant steps while designing and managing education programmes. Furthermore, they provide insights into which information can be relevant within models of such programmes.

2.2 Model-Driven Engineering

This section covers the basics of MDE relevant to this thesis. Section 2.2.1 explains essential terms and their relation. Section 2.2.2 discusses the engineering of modelling languages. Section 2.2.3 covers the fundamentals of model transformations, focusing on Triple Graph Grammars (TGGs).

2.2.1 Terminology

The term "model" is widely used, and what is considered a model varies significantly based on the domain. For architects, a blueprint or small-scale replica of a building is a model. For software architects, it is rather a visual diagram consisting of boxes and lines (with well-defined semantics) used to express a software system's structure or behaviour. While the concrete types models are different, on an abstract level, they all fit Stachowiak's definition of a Model [Sta73], which is based on three properties: (1) there is a correspondence of the model elements to the real world (mapping/homomorphism) (2) they abstract from irrelevant details (reduction/abstraction), and (3) the model must serve a particular goal, i.e. it can be used instead of the original for analysis (pragmatism). From which details models abstract depends on the goal of the model.

Model Definition of Stachowiak

A model may serve multiple goals, but not all are relevant for every stakeholder. Therefore, a model can be viewed from different viewpoints. A business analyst may look at a software model from a financial viewpoint to determine the overall cost of the system and an operator from a performance viewpoint to judge the system's scalability. A view is a model shown from a particular viewpoint. Viewing a software system model from a performance viewpoint would hide information about the person-hours needed to build the actual system and focus on performance aspects like the required disc space.

Viewpoints and Views

When engineering complex systems, it has become standard practice to model a system to a certain extent before implementing it. The degree to which a system is modelled and how the models are utilized varies. Models may only be used informally to sketch the system and to communicate with others. Such informal models can be powerful vehicles to improve communication. However, due to the lack of standard interpretation rules, an external person may not derive the same information from such a model.

Informal Modelling

When models are used systematically, it is called *Model-based Engineering*. Those models are based on well-known modelling languages, such as the Unified Modelling Language (UML) [Obj17], and are used systematically throughout the development process. For instance, models are used to document requirements or the system architecture. At a certain point, the system is implemented based on these models. However, the transition from models to the actual system is mainly manual in Model-based Engineering. This manual transition often leads to the problem that the model and the system diverge after some time, and the models no longer reflect the implemented system.

Model-based

In MDE, models are primary artefacts used to generate parts of a system automatically or that are interpreted at runtime [BCW17]. Typical practice examples are data models, which are transformed into database schemes or process models used to steer processes within workflow engines. MDE is a specific subtype of model-based engineering. Since models are primary artefacts, they should stay in sync with the implementation. Changes in the system are either the result of generating system parts based on a changed model (top-down) or the changes in the system are reflected in the model through reversibility (bottom-up) [POB00].

Model-Driven

The use of models in engineering methods differs significantly. The MISA method for Instructional Design heavily relies on the MOT+ modelling language [PdlTL+05]. Other methods merely suggest that certain modelling languages can be utilized, e.g. the V-Modell XT [FKSH09] used in software engineering recommends the usage of the UML but does not prescribe it.

Models and Engineering Methods

Agile methods tend to avoid mentioning modelling directly, which is often falsely interpreted as modelling and Agile methods contradicting each other. While the Agile Manifesto [BBvB+01] for software development states that working software is favoured over comprehensive documentation, it neither forbids nor prescribes the use of models. Especially when models are executed at runtime or used to generate code, they are as good as code. Furthermore, documentation is still crucial, and models can be used to explain complex components, e.g. their structure or life cycle. When dealing with many teams, the role of models as communication vehicles becomes more important. Hence, the Scaled Agile Framework (SAFe) [Sca21] deliberately mentions the usage of model-based systems engineering.

Modelling and Agile Methods

Different means exist to express models, e.g. prototypes or mathematical equations. In this thesis, we focus on models expressed using modelling languages. Such modelling languages define the syntactical rules models must follow and the semantics of the concepts that can be used. Modelling languages are available for a multitude of purposes. Several modelling languages exist to express educational scenarios, such as IMS-LD [IMS03] or E^2ML [Bot08]. The UML is the most prominent modelling language in software engineering and integrates multiple languages for structural and behavioural modelling. The Business Model and Notation (BPMN) [Obj11] has become the de-facto standard for (business) process modelling. The Software Process Engineering Metamodel (SPEM) [Obj08] allows formal modelling development methods such as Scrum, RUP and others. The System Modelling Language (SysML) [Obj19] is one of the major modelling languages for systems engineering.

Modelling Languages

Modelling languages can be General-Purpose Languages (GPLs) or Domain-specific Languages (DSLs). It is a matter of perspective if a language is a GPL or a DSL. The UML is referred to as a GPL, but if the interpretation domain is stretched to cover the broad software engineering domain, it could also be a DSL. Volter et al. [VBD+13] give guidance to distinguish a GPL from a DSL. They describe that a GPL is often standardized, has a long lifespan, evolves slowly and addresses large/complex domains. For instance, these criteria fit languages like UML, SPEM, BPMN or SysML, standardized by the Object Management Group (OMG), or IMS-LD, which 1EdTech[1] maintains. In contrast, a DSL addresses a smaller and well-defined domain, has a shorter lifespan and can evolve faster. In this thesis, we develop a DSL for modelling professional education programmes. How to develop modelling languages, especially DSLs, is the subject of the next section.

GPL vs DSL

[1] https://www.imsglobal.org/learningdesign

2.2.2 Engineering Modelling Languages

Previously, we discussed what models are and how they are used. This section focuses on how to create modelling languages for defining models. For this, we describe how to define the syntax and semantics of such languages and present a process for language development.

Syntax

The basis for any modelling language is its abstract syntax [BCW17]. It defines the structure of the language by specifying its concepts and their relationships. One approach to defining the abstract syntax of a modelling language is metamodelling. Thereby, a metamodelling language is used to define a modelling language. An example of a metamodelling language is the Meta-Object Facility (MOF) [Obj15] provided by the OMG. MOF is based on a subset of the UML, class diagrams, and is used to define the metamodels for languages like UML, BPMN or SPEM. In this thesis, we directly use UML class diagrams for metamodelling.

Abstract Syntax

A concrete syntax provides a representation of elements of the abstract syntax of a modelling language. There must be at least one concrete syntax for the language to be usable. A concrete syntax can be visual, textual or a mix of both. Visual concrete syntaxes are defined by providing visual notation elements, and they map them to the concepts they represent. This mapping is often described as a legend in the diagram or via a series of examples. Language engineers usually define the grammar or use a schema language like XML or JSON Schema to define a concrete textual syntax. We present a new modelling language for professional education programmes in this thesis and use TGGs to map its abstract syntax to concrete visual syntaxes.

Concrete Syntax

The mapping of notational elements of the concrete syntax to the elements of the abstract syntax can be incomplete, meaning there can be elements in the abstract syntax that are not covered by the concrete syntax and vice versa. In his seminal work on concrete visual notations, Moody [Moo09] refers to these two cases as *symbol deficit* and *symbol excess*, respectively.

Incomplete Mapping between Abstract and Concrete Syntax

In the case of a symbol deficit, a concrete syntax does not provide a representation for a particular element of the abstract syntax. However, another concrete syntax may define a representation for this element. For instance, the extension mechanism of BPMN does not have a visual representation in the concrete visual syntax. Still, the extension elements can be expressed in the XML-based textual syntax.

Symbol Deficit

Symbol excess refers to the case that the concrete syntax contains a representation that does not map to any element of the abstract syntax. Moody lists comments in the UML as an example of symbol excess and stresses that notational elements not corresponding to any element of the abstract syntax add unnecessary complexity.

Symbol Excess

Also, the mapping between elements of the abstract and concrete syntax does not necessarily need to be a one-to-one mapping. Moody [Moo09] speaks of *symbol redundancy* if multiple different elements of the concrete syntax can represent the same element of the abstract syntax. While this provides flexibility for the modellers, it can negatively impact comprehensibility since seemingly different diagrams may be semantically equal. In the case of *symbol overload*, the same element of a concrete syntax represents multiple abstract syntax elements. Hence, two seemingly equal diagrams can differ semantically.

Symbol Redundancy

Symbol Overload

Audiences must know that a model's concrete depiction may not show the complete model. A depiction can be incomplete due to symbol deficits or because the concrete representation shows a specific view of the model and intentionally omits elements. In modelling tools such as Enterprise Architect[2], users can add elements to a model but choose not to show them on a diagram. Elements not shown in a diagram may be needed for semantical integrity and can be displayed on other diagrams of the same model.

Model vs. Diagram

Semantics

The semantics of a modelling language describe the meaning of its elements. Semantics can be defined informally using natural language. To a large extent, the semantics of the UML is defined informally. Informal definitions are often sufficient but leave room for different interpretations and ambiguities. Especially when quality aspects of models need to be checked automatically or models are executed, a formal definition of semantics is required. Formal semantics can be subdivided into static and dynamic semantics.

Formal vs. Informal Semantics

Static semantics describe constraints that can be evaluated without executing a model. Partially, such constraints can already be defined as part of the abstract syntax by defining multiplicities. The capabilities of the MOF (or UML Class Diagrams) to express static semantics are limited, and an additional language called the Object Constraint Language (OCL) [Obj14] can be used to define more extensive static semantics.

Static Semantics

[2] https://sparxsystems.de

For formal dynamic semantics, we can distinguish between translational and operational semantics. Translational semantics define a mapping to an existing language for which the semantics are already specified, e.g. the BPMN specification [Obj11] provides a mapping to BPEL for execution. Operational semantics [Plo04] define in which states a model can be and rules for state changes. For instance, Dynamic Meta Modelling (DMM) [EHHS00] enriches a language's metamodel with elements needed to describe the operational semantics. The result is a runtime metamodel over which graph transformations are defined that describe possible state changes. Alternatively, a model interpreter can be developed to define operational semantics, e.g. process engines implement the operational semantics of process modelling languages.

Dynamic Semantics

Language Engineering Process

Brambilla et al. [BCW17] define an iterative three-step process for developing modelling languages, which is depicted in Figure 2.7. Each step of this process is discussed in the following.

In the first step, the domain addressed by the new modelling language has to be analysed. For this, the language engineers must define the purpose, the realisation and the content of the language [KKP+09]. The purpose states the modelling goal of the new language, i.e. for what the models shall be utilized. Regarding language realisation, designers must decide if a language shall be created from scratch or as an extension/alteration of an existing one. Based on the language's purpose, the content of the new language must be described. This content analysis can be done by synthesising the content via a series of examples or by analysing the design dimensions of the domain.

Modelling Domain Analysis

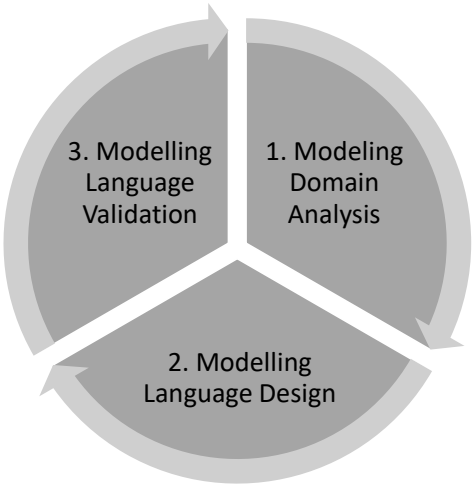

Figure 2.7: Language engineering process (based on [BCW17])

Modelling Language Design

In the second step, the new modelling language is designed by defining the abstract syntax, the semantics and at least one concrete syntax. The previously identified content of the modelling language allows for deriving requirements towards the language and its expressivity.

Modelling Language Validation

In the third step, the newly developed modelling language is validated. This validation can take different forms. For instance, the language can be matched against the purpose and content defined in the first step, or the expressiveness can be evaluated by modelling domain examples. Also, general quality requirements for modelling languages can be evaluated, such as separation of concerns, simplicity, scalability, or supportability [POB00]. Furthermore, a user study could analyse a language's ease of use or whether the correct meaning can be inferred from language instances.

Iterative Process

As visualized by Figure 2.7, the process is iterative and not finished after going through all steps once. The validation can uncover domain analysis or language design flaws that lead to further iterations. Furthermore, domains change over time, and a new language version may be needed.

Designing a Modelling Language from Scratch

When starting language development from scratch, language engineers have to define a new metamodel describing the abstract syntax, e.g. using the MOF or another technology space [IBA02], define the semantics and provide a concrete syntax. This freedom is beneficial if the new modelling language differs from existing ones. Depending on the technology space, tool support can be generated to a certain extent. For instance, Xtext[3] can generate editors with syntax checks and auto-completion for textual languages defined using Ecore, the MOF variant of the Eclipse Modelling Framework (EMF) [SBPM08].

Language Extension

Instead of starting from scratch, language engineers can base their language on an existing one. We differentiate between an uncontrolled and a controlled extension of an existing language. In the case of the former, language engineers reuse parts of existing languages, e.g. excerpts of the metamodel or a concrete synax, and freely adapt them as they wish. Such changes have the consequence that the tool support for the base language cannot be reused for the new language or must be adapted. Several languages provide a controlled extension mechanism to maintain tool support. For instance, any MOF-based language could be extended by using profiles, but in practice, profiles are mainly supported for the UML. Other languages like BPMN or OpenAPI, a JSON-based API describing language, also provide extension mechanisms. While the tool support remains intact for the base language, the tools usually only have limited knowledge about the extension, resulting in limited support for

[3] https://eclipse.org/Xtext

tool assistance for the extension, like a lack of the before-mentioned syntax checks or auto-completion. It largely depends on the language that shall be created and for which purposes it shall be used if a creation from scratch or an extension/alteration is preferable.

2.2.3 Model Transformation

Model transformations play a crucial role in MDE. They can be used to maintain consistency between different models or to transform one model into another. In the following, we discuss different types of model transformations and mainly focus on Triple Graph Grammars.

Types of Model Transformations

Endogenous and Exogenous Transformations

Czarnecki and Helsen [CH06] provide a comprehensive overview of possible features of model transformation approaches. For this thesis, the distinction between endogenous and exogenous model transformation is relevant since this thesis focuses on the latter. For endogenous model transformation, the transformed model is still an instance of the same metamodel as the original model. In contrast, exogenous model transformations transform a source model m_{src} belonging to a source metamodel M_{src} into a target model m_{tgt} that is an instance of a target metamodel M_{tgt}.

Focus on Exogenous Model Transformations

In this thesis, we leverage exogenous model transformation to provide tool support for our visual modelling language (see Chapter 6) and to map information stored in other tools to instances of our modelling language (see Chapter 8). The basic concepts of exogenous model transformations are shown in Figure 2.8. On the topmost layer, a transformation language is defined between a source and a target metamodelling language. These metamodelling languages can be different but are often the same, e.g. EMF Ecore [SBPM08] or MOF [Obj15]. Based on a transformation language, rules can be defined to transform source metamodel instances into target metamodel instances. The transformation language is not only used to describe transformation rules but also a correspondence metamodel, which describes how classes of the source correspond to classes of the target metamodel. When a source model is transformed into a target model or vice versa, these correspondences are created and utilized for pattern matching. Patterns describe the existence of specific elements or relations in a model, e.g. a BPMN model pattern could describe a task immediately following a start node. Change operations on the source or target model are performed if a pattern matches.

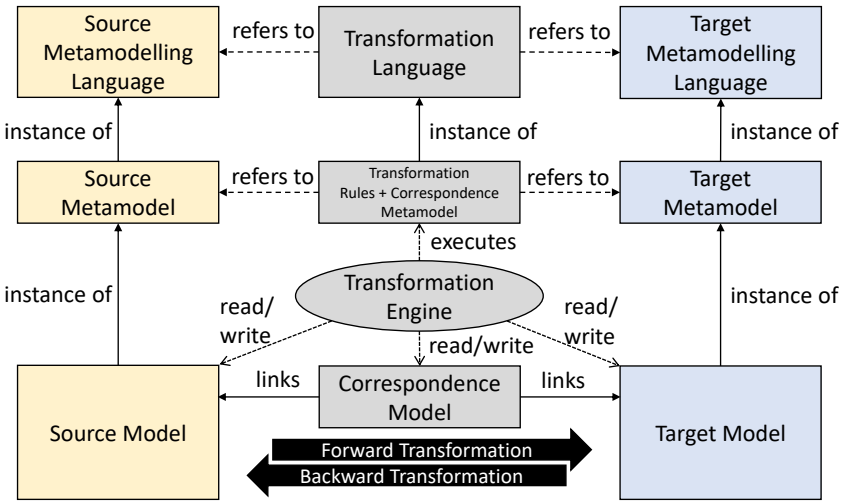

Figure 2.8: Basic concepts of exogenous model transformations (extended from [CH06])

Triple Graph Grammars

Triple Graph Grammars

We use Triple Graph Grammars (TGGs) in our approach for exogenous model transformation [Sch94]. The "triple" in TGG refers to the source, correspondence and target graph. TGG rules are declarative and describe consistency relations between a source and a target metamodel [Anj18]. A source and a target model are considered consistent if a sequence of TGG rules can be found to generate this pair of models from scratch. TGG rules can be operationalised for different purposes, which are explained in the following.

Operationalization

When using TGGs for consistency checks [LAS17], we can distinguish between two operationalizations: *Check Only* requires existing correspondence nodes between the source and the target models and determines if the given triple graph is consistent without manipulating it. *Correspondence Creation* takes a source and target model as input, checks their consistency and creates the correspondences between consistent parts.

Consistency Checks

Forward Transformation (FWD) [KLKS10] transforms a source into a target model. FWD builds up the entire target model and does not extend or adapt an existing one. *Backward Transformation* (BWD) does the same in the reverse direction. *Model Synchronisation* can be unidirectional [GW06] or bidirectional [GW09, Wei22]. In contrast to FWD, unidirectional forward model synchronisation takes an existing target model into account and extends/adapts it accordingly. Concurrent model synchronisation can deal with changes in both the source and target model and tries to reach a consistent state. Detected conflicts are resolved based on heuristics or manually by users.

Forward/Backward Transformation

Model Synchronisation

eMoflon Modelling Specification Language (eMSL)

eMSL to specify Metamodels and TGGs

As a transformation and metamodelling language, we use the eMoflon Modelling Specification Language (eMSL) in this thesis. The language has been developed as part of the eMoflon::Neo TGG engine [WA21]. Aside from specifying transformation rules and metamodels, eMSL can also express endogenous model transformation and models confirming to metamodels. In the following, we briefly introduce eMSL and focus on using it to define metamodels and TGGs. A comprehensive eMSL introduction is given in [Sch19].

eMSL vs. UML Class Diagrams

Defining metamodels with eMSL is similar to defining them with UML class diagrams. The significant differences are: In eMSL, all associations, including aggregations and compositions, are directed. The association name is simultaneously the role name to refer to the association's target end. Associations classes for binary associations do not need to be represented separately since properties can be added to any association directly. Multiplicities cannot be defined for properties. For associations, the multiplicity can only be defined for the target's end of the association.

eMSL's syntax

eMSL has both a visual and a textual syntax. However, not all information that can be specified textually is represented visually, e.g. properties of associations are not shown in eMSL's current visualization. As a consequence, we use UML class diagrams for visualising eMSL metamodels. In Figure 2.9, we show an example of a UML class diagram and its mapping to a textual eMSL representation. In contrast to the eMSL's visualization, we depict associations with properties as association classes and use the stereotype «association» to indicate that they are translated to an association in eMSL.

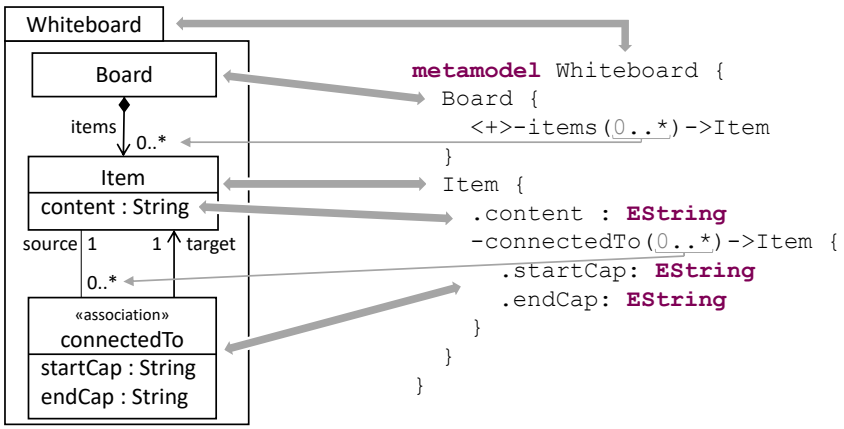

Figure 2.9: Example mapping between a UML class diagram and eMSL's textual representation

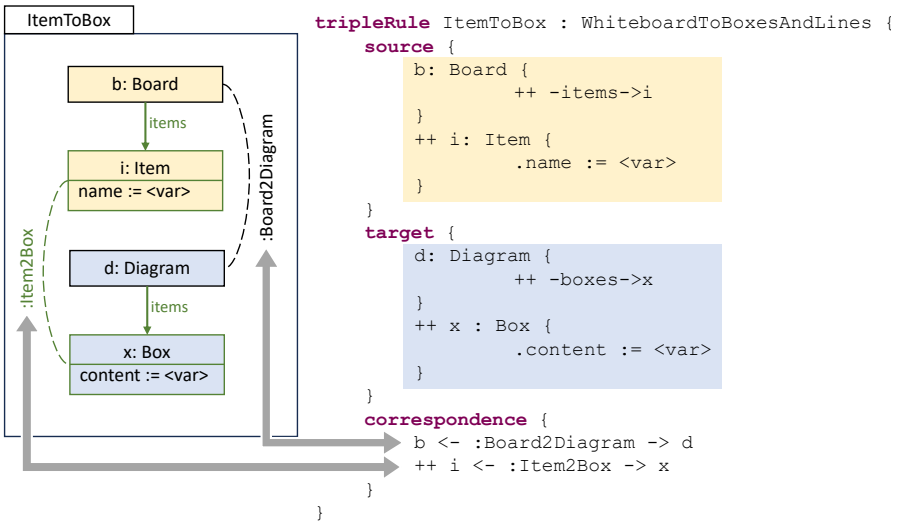

Figure 2.10: Comparison of a TGG rule's visual and textual representation

Visual and Textual Example of a TGG Rule

Figure 2.10 shows an example of a TGG rule in its visual and textual representation. A rule defines objects of the source and target metamodels and uses correspondences and variables to set them into relation. The rule in the example has the name `ItemToBox` and is part of the TGG `Whiteboard-ToBoxesAndLines`. The example rule maps items on a whiteboard to boxes in a diagram. In this rule, the whiteboard object in the source model and the diagram object in the target model must exist, whereas the item object in the source model and box object in the target model are created. The created elements are coloured green in the visual notation and marked with a "`++`" operator in the textual representation. If a rule is all green, meaning all elements are created, it is called an *axiom* [Anj18]. The dashed lines between objects represent correspondences. Variables, like "`<var>`" in the example, are used to map values of properties of the source and target model. In addition to the rules, a TGG specification (not depicted) is needed that consists of a list of all rules, available correspondence types and references metamodels.

2.2.4 Summary

In this section, we explained essential MDE terminology and how modelling languages can be engineered. Furthermore, we covered the basics of model transformations with TGGs and showed how eMSL can be used to describe metamodels and TGGs. In later chapters, we introduce a new modelling language for professional education programmes and use TGGs to couple the abstract syntax of this language to a whiteboard-compatible visual syntax.

2.3 Situational Method Engineering

The success of applying a method always depends on the situation in which it is applied. SME aims to consider the situation during method selection and development and produce a method specifically suited to the situation. Section 2.3.1 defines relevant terminology. Section 2.3.2 explains different approaches to engineering situation-specific methods. Section 2.3.3 focuses on assembly-based method engineering, which is the most relevant to this thesis.

2.3.1 Terminology

Unfortunately, there is no predominant terminology in the field of SME. Consequently, we describe how terms must be interpreted in this thesis' scope.

It is important to understand what a method is and how it differs from a process to understand SME. The relation between the terms "method" and "process" is not intuitive or commonly agreed upon. According to the Cambridge Dictionary, a method is "a particular way of doing something"[4], and a process is "a series of actions that are needed in order to do something or achieve a result"[5]. These two definitions overlap because a "particular way" also implies a series of actions. Nevertheless, distinctions between these two terms can be made. Engels and Sauer [ES10] state that a process focuses on a series of actions, while a method goes beyond that and entails the broader context, like roles, artefacts, or tools. Nonetheless, there is a close relation between these two terms because a process is part of a method, and a process can combine a set of methods. We visualized the relation of these two terms for the scope of this thesis in Figure 2.11. Method and process are intentionally on the same level in this visualization to emphasize that it is a matter of perspective, which term is more appropriate. If the emphasis is on the series of tasks, the term "process" is more appropriate. If the focus is on the broader context, the term "method" is more appropriate. The visualization in Figure 2.11 is greatly simplified. Aspects like control flow in processes or additional interrelations between the constituents of methods are omitted.

Process vs. Method

Matter of Perspective

Another distinction regarding methods and processes is whether we refer to a metamodel for their specification, a concrete specification, or the enactment of a method/process. These three levels are visualized in Figure 2.12 alongside an example of each level. For instance, the Software & Systems Process Engineering Metamodel Specification (SPEM) [Obj08] is an example of a

[4] https://dictionary.cambridge.org/dictionary/english/method
[5] https://dictionary.cambridge.org/dictionary/english/process

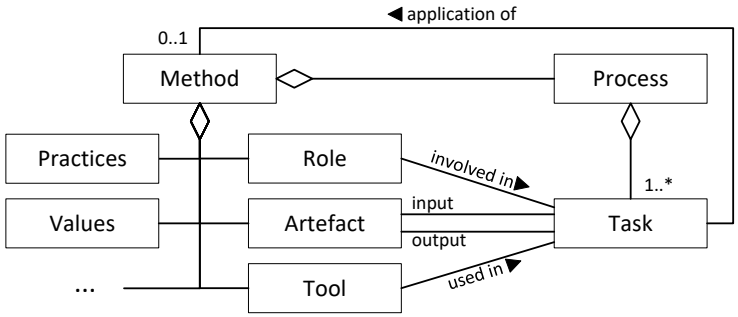

Figure 2.11: Simplified visualization of the relationship between method and process visualized as a UML class diagram

Metamodel, Specification, Enactment

metamodel to specify engineering methods/processes. SPEM can be used to specify a method like Scrum, which is located at the specification level. The enactment level refers to the actual use of a method/process. As Henderson-Sellers and Ralyté [HR10] point out, the term "process" is inconsistently used to refer to the specification or enactment level or both. In this thesis, we use "method" and "process" to refer to their specification. We use prefixes like "enacted", "used", or "applied" to refer to the methods/processes in use.

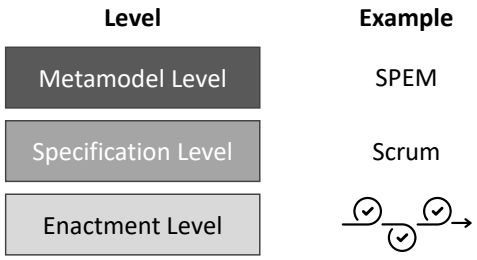

Figure 2.12: Levels at which we can talk about methods and processes

Method Scale

Unfortunately, there is not just one type of method (or process). For instance, an engineering method is significantly different from a feedback method for a workshop, e.g. in length, complexity, or quality requirements. With "engineering method", we refer to a systematic approach to solving problems and designing appropriate solutions. Engineering methods are required in various domains, like software engineering, mechanical engineering, or Instructional Design. They usually address a large part or even the whole project life cycle. If an engineering method is chosen wrongly, it can significantly affect the overall project's success. In contrast, a feedback method is an example of a small-scale method that is relatively short, involves a few steps, and the impact is (probably) neglectable when chosen wrongly. SME often focuses on creating situation-specific large-scale engineering methods. Smaller scale methods can become a part of the situation-specific engineering method.

Different terms refer to parts of a method: The term "method fragment" refers to individual parts of a method, like a tool or a role [HSRÅR14]. A combination of fragments useful for a specific purpose is called a "method chunk". Such chunks do not necessarily need to be self-contained, meaning a method chunk may not represent a complete (smaller scale) method. The terms "method component" or "method service" refer to self-contained, reusable methods that can be used within engineering methods [FB16]. For instance, a daily stand-up meeting is a method component used in Scrum and could be used in other methods. A key differentiator between method components and services is that the latter also considers the execution of the method. This thesis uses "method part" to refer to any sub-part of an engineering method, i.e., a fragment, chunk, or component.

Fragment, Chunks, Components and Services

The definition mentioned above of process from the Cambridge Dictionary used the term "action" to describe the steps of a process. Other terms for process steps are frequently used, such as "activity" or "task". The Cambridge Dictionary provides the following definitions for these terms: A task is "a piece of work that needs to be done, especially one that is a regular part of someone's job"[6]. An activity is "the work of a person, group, or organization to achieve something, especially to make money."[7] An action is "something that you do, especially in order to deal with a problem or difficult situation"[8]. Again, there is a significant overlap in these definitions. The BPMN [Obj11] uses "activity" to denote a process step and defines a task as a subtype of activity. The UML [Obj17] uses "activity" as a synonym for process and uses "activity node" for a step within a process. In the UML, an action is a more specific type of activity node. These differences in the terminology of two well-known process modelling languages maintained by the same organization show that there is no predominant terminology. For the scope of this thesis, we use all three terms as synonyms for work to be done as part of a process or a method.

Task vs. Activity vs. Action

The situation refers to all relevant information for an engineering project, e.g. its contexts, the people involved, or the product/service being developed. The situation of all engineering projects is different. For instance, a slightly (different) product is developed, or more/less/other people are involved. These differences in situations affect the success of an engineering method because no method can address all situations.

Situation

[6] https://dictionary.cambridge.org/dictionary/english/task
[7] https://dictionary.cambridge.org/dictionary/english/activity
[8] https://dictionary.cambridge.org/dictionary/english/action

Describing a situation with all its relevant information is infeasible and hard to keep up-to-date. Instead, so-called situational factors refer to key aspects of the situation, like the number of people involved or their experience level. The type of value expressed by a situational factor can be very different. For instance, the number of people involved can be expressed by a number or a range. Other situational factors, like the native language of the people working on a project, might be a list of string values. Similarly, a situational factor could express if it is an international project, where people from different countries contributed or not, which can be reflected using a boolean value. Unfortunately, there is no definitive set of situational factors. Bekkers et al. [BvdWBM08] identified a set of situational factors for software products, e.g. product size, customer involvement, development team size or development philosophy. However, situational factors are based on experience, and each domain has factors that are not necessarily relevant to other domains.

Situational Factors

A method is situation-specific if it is created, selected, or adapted to fit a given situation by considering situational factors. For instance, if the customer wants to be involved, the company has an Agile development philosophy and there is just one team working on the product, Scrum may be a good choice. If a project is spread across multiple teams, Scrum may not be the ideal choice since it does not address proper inter-team communication. For such large-scale projects, another method like the Scaled Agile Framework (SAFe) [Sca21] is probably the better fit since it is designed to deal with multiple teams.

Situation-specific

2.3.2 Paths Towards Situation-specific Methods

Different approaches to SME exist. Harmsen et al. [HBO94] use the degree of controlled flexibility to categorize SME approaches. Flexibility refers to the ability to adapt a method to different situations, and control to the ability to guide the method adaptation. Harmsen et al. [HBO94] position fixed methods on the low end of controlled flexibility and modular construction of methods on the high end of the spectrum. In Figure 2.13, we show an adapted version of this spectrum created by Fazal-Baqaie [FB16]. He refers to the modular construction of methods as assembly-based approaches and extends the flexibility spectrum to the right by adding creation-based approaches and free (unguided) method creation. Furthermore, he neglects the control aspect and instead introduces a second dimension reflecting the effort required to ensure a workable (“unflawed”) method as an outcome.

Flexibility vs. Effort

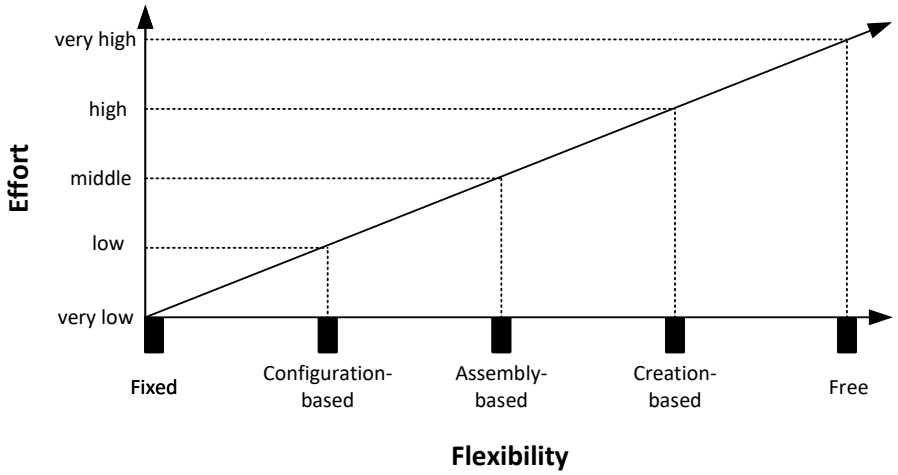

Figure 2.13: Categorization of SME approaches based on the degree of flexibility to adapt to a situation and effort required to create a high-quality method (adapted from [FB16])

Fixed methods do not provide explicit means for method adaption. This means that method users cannot rely on the guidance of the method developers to adapt the method to their situation. Ideally, at least a selection between different fixed methods is possible to select the most appropriate fixed method for the current situation. The effort required to ensure an enactable method as a result of an SME endeavour increases with the degree of flexibility. The benefit of fixed methods is that they have been tested and showed that they can be enacted and add value. Consequently, the effort required to get an enactable method is mainly determined by selecting the best, fixed method for the current situation.

Fixed Methods

Some methods provide configuration options. For instance, the software engineering method V-Modell XT [FKSH09] allows defining which deliverables are necessary. For such configurable methods, the number of configuration options determines the effort. Consistency constraints can ensure that invalid configurations are avoided. Since the result is just a method variation, the effort required to get an enactable method is relatively low.

Configuration-based Approaches

Assembly-based approaches provide an even higher degree of flexibility than configurable methods. In this case, method parts can be selected from a repository based on the current situation, and construction guidelines help to assemble a situation-specific method. Assembly-based approaches represent a balanced trade-off between effort and flexibility. The method repository allows the reuse of existing method parts, and guidance is provided in selecting appropriate ones.

Assembly-based Approaches

Creation-based

Creation-based approaches provide a metamodel like SPEM for specifying methods. The advantage is that the resulting methods are described in a standardized format, and tool support for consistency checks or enactment can be provided. Since the method content is up to the method developer, the effort for method creation is relatively high.

Unguided

Free method creation does not guide how to describe a situation-specific method. The benefit is that method developers are not constrained by any metamodel and can freely design their method. For instance, they can use their own visualizations to communicate how their method works. The effort required to design and enact a freely designed method is very high, e.g. no prior knowledge about the notation may exist, the semantics of visualizations must be understood, and tool support must be developed.

Abstraction Level of a Method

One aspect not covered by the SME spectrum defined by Harmsen et al. [HBO94] is the degree to which a method abstracts from a specific situation. For instance, the Agile method Scrum could be considered a fixed method. It even defines a dedicated role, the Scrum Master, to ensure the method is followed. However, Scrum abstracts from the situation in which it can be applied, e.g. it does not prescribe activities for software engineering even though it originated in that domain. The result is that Scrum is agnostic to the domain in which it is being used and can be applied in various situations. Due to the level of abstraction, certain parts of Scrum, like the work organization within a sprint, are underspecified, and experience is needed to deal with such underspecified areas. Assembly-based approaches can assist in such situations since methods addressing the underspecified parts can be obtained or constructed based on the method repository.

2.3.3 Assembly-based Situational Method Engineering

The idea behind assembly-based method engineering is similar to component-based software engineering. Reusable method parts are defined, together with information on when they are applied and how they connect to other method parts, e.g. input and outputs. The core tasks of such approaches are clustered into different layers and shown in Figure 2.14.

Method Base

The core of an assembly-based method engineering approach is the method base (sometimes also called a method repository), which contains reusable methods or method parts alongside information about situations in which they apply. A method base does not necessarily have to be a digital database or coupled to a specific SME approach. For instance, many books describe

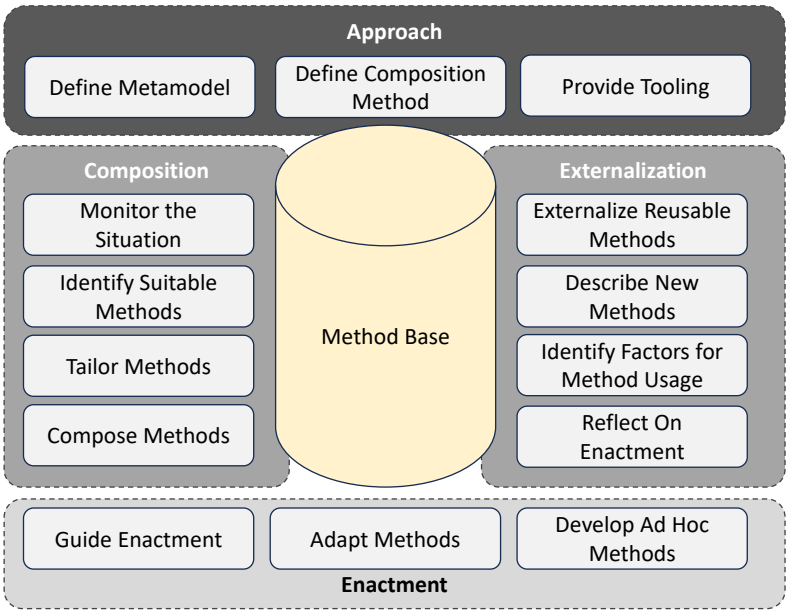

Figure 2.14: Overview of tasks in assembly-based method engineering (adapted from [FB16])

reusable methods for specific purposes like digital innovation [Dar17] or games to illustrate Agile principles [WB19]. The DIN SPEC 33453 for developing digital service systems [PHW+19] can also be considered an assembly-based SME approach for a specific domain. It defines an iterative, agile process consisting of three main phases (analysis, design and implementation), and decision points determine which phase follows next. The standard does not prescribe methods to use in the respective steps but provides a structured tabular description of methods, a method base, in the appendix of the standard.

Method Bases in Practice

The developers of an SME approach specify the metamodel behind their method base, define how methods can be composed using their approach and provide tooling. Tooling typically supports tasks on the other layers, e.g. filling the method base, retrieving information, or guiding method enactment.

Approach Definition

A method engineer uses the SME approach to compose a situation-specific method. For this, the method engineer has to monitor the situation and can use the method base to identify suitable method parts. These method parts are then used to compose a method for the given situation. Existing method parts may be adapted (tailored) before they are used to fit the situation better.

Method Composition

The enactment layer describes the actual execution of the composed method. The project team is responsible for this layer and for guiding the enactment. Tools for project management or process engines can also assist in guiding the enactment. It may be necessary to adapt the method during enactment,

Method Enactment

e.g. the situation changes, or the initially composed method is no longer appropriate. Especially for underspecified parts or new situations, it may be necessary to define new methods in an ad hoc fashion.

Externalizing Method Knowledge

The externalization layer is concerned with filling the method base. This can be done by externalizing knowledge about reusable methods, e.g. from literature or based on experience, or by specifying completely new methods. Ad hoc methods developed during enactment may be refined and added to the method base as part of externalization work. Reflecting on previous enactments is necessary, especially when defining (adapted/new) methods for reuse based on experience. This externalization work is typically done by experienced method engineers, which is why Fazal-Baqaie [FB16] refers to the people performing these duties as "senior method engineers".

Methods can be incomplete

Please note that composition, enactment, and externalization are not sequential phases. For products with a long lifespan, the method cannot necessarily be defined completely upfront or just on an abstract level. Hence, method composition may not be completed when starting to enact a method. Similarly, externalization does not have to happen after enactment but can also be done parallel to composition and enactment.

2.3.4 Summary

In this section, we introduced the basics of SME. We discussed the relation of essential terms, like method and process, and presented a classification of SME approaches. We focused on assembly-based method engineering since we use the same concept to give situation-specific recommendations to education providers while designing and managing professional education programmes. These recommendations are based on a knowledge base that includes method parts and possible content items for such programmes.

Chapter 3

Towards Modelling Professional Education Programmes

Throughout the next four chapters, we present a modelling approach for professional education programmes. The development of our modelling approach follows the modelling language engineering process from Brambilla et al. [BCW17], which we introduced in Section 2.2.2. This chapter covers the first phase of this process, the *Modelling Domain Analysis*, which entails defining the modelling language's purpose, content and realisation.

Section 3.1 describes the characteristics of professional education programmes to provide general context information for our modelling approach and as a basis to define the language's content. Section 3.2 states the language's purpose and derives requirements for the modelling language. Section 3.3 analyses to which degree existing modelling approaches already address these requirements. Section 3.4 provides a chapter summary and explains the realisation strategy of our modelling approach.

3.1 Characteristics

In this section, we explain the main characteristics of professional education programmes by focusing on the 3Ps: the *product* itself, the *people* involved and the *processes*. Afterwards, we briefly mention characteristics beyond the 3Ps and differences to other education programmes. This section builds upon the introduction of Instructional Design given in Section 2.1.

3.1.1 Product

The products we are concerned with in this thesis are professional education programmes. As defined by the ISCED [UNE11], an education programme is "a coherent set or sequence of educational activities designed and organized to achieve pre-determined learning objectives or accomplish a specific set of educational tasks over a sustained period." The prefix "professional" in front of an education programme narrows the target audience. It refers to "a person who has the type of job that needs a high level of education and training"[1]. Hence, professional education programmes are not intended for undergrads or school students but for people already working in a profession.

Professional Education Programmes

As briefly mentioned in Section 1.1, education providers can offer off-the-shelf (OTS) or bespoke education programmes. OTS programmes are not tailored for a specific client and run as open programmes, meaning anyone can enrol independently of their affiliation. OTS programmes can also run as in-house events for specific companies if the demand within the company is high enough. Other than the potential change of venue, OTS programmes remain the same whether they run as open courses or as in-house events. E-learning offerings such as Massive Open Online Courses (MOOCs) are among OTS programmes. However, bespoke programmes are needed if companies require programmes focussing on their specific needs [Mac06]. These can be programmes developed from scratch or tailoring/mixing the content of existing programmes. Similarly, bespoke programmes can be generalised and offered as OTS programmes.

Off-the-shelf vs. Bespoke

Regarding the delivery of education programmes, we can distinguish between spatial and temporal flexibility [PP21]. The spatial flexibility determines the degree of freedom participants have when they want to attend a part of a programme. Traditional face-to-face teaching requires participants to be in the same room and does not offer spatial flexibility. In contrast, spatial restrictions usually do not apply to online delivery (e-learning). Temporal flexibility describes if the participants can only attend a part of a programme at a specific time (synchronous) or can do it anytime (asynchronous). In combination, spatial and temporal flexibility requirements determine which delivery mode is appropriate for a programme. During the COVID-19 pandemic, traditional face-to-face teaching at universities often shifted towards a synchronous form of online teaching. While students could attend from anywhere, they were bound to a specific time. Self-paced online learning programmes like MOOCs

Delivery Modes

[1] https://dictionary.cambridge.org/dictionary/english/professional

often do not restrict participants concerning time and space. However, this flexibility can negatively impact a programme's efficacy, as it can be less personalised and engaging [PP21]. The spatial and temporal flexibility can differ between different parts of an education programme.

Professional education programmes can have a limited lifespan or run indefinitely. Programmes intended to introduce a new way of working, e.g. Agile methods or new software, often have a limited lifespan. Such transition programmes typically end after all relevant persons passed the education programme. Onboarding, talent or executive programmes usually run indefinitely because new employees join the company. If programmes run indefinitely, it does not mean they stay static or never end. Programmes are adjusted to meet clients' and participants' needs, which is discussed further in the process part of this section (see Section 3.1.3). At some point, programmes may end due to cost reasons or new programmes are introduced.

Lifespan of Programmes (Perspective of Companies)

Professional education programmes can be iterative or continuous or contain aspects of both. Iterative programmes run multiple times, each iteration with a new cohort of participants. For continuous programmes, there is no clear separation between cohorts. This missing separation can mean that participants are not structured into cohorts, e.g. because the programme focuses on individual learning or that cohorts are mixed. Most schools operate iteratively, i.e. each year, a class advances to the next grade and covers the same material as the class before them did. MOOCs are examples of continuous programmes where participants can join at any time. University study programmes contain aspects of both. Students enrol each semester, and courses are mixed with students from prior semesters.

Continuous vs. Iterative

The duration of an education programme for participants can vary depending on the style of the programme. For self-paced programmes, there can be an upper limit concerning the duration, but the actual time depends on the individual. Other programmes have a fixed duration for all participants. In that case, the pace for all participants is defined by the programme. In principle, a one-day workshop could already be declared an education programme. However, professional education programmes typically last a few days to a few months. Study programmes at universities last multiple years, but we argue that such long durations are less common for professional education because companies operate at different paces.

Duration for Participants

Content-wise, we can distinguish between breadth and depth. Programmes that focus on breadth cover various topics, but on a high level, but none in-depth. In contrast, programmes exist that focus on individual topics.

Breadth vs. Depth

Certification programmes for a specific Cloud infrastructure are examples of in-depth programmes. Onboarding or talent programmes are often rather focused on a wide variety of topics. Depending on the duration of a programme, both breadth and depth can be achieved, e.g. university study programmes.

Linear vs. Non-Linear

The learning path outlined by a programme can be linear or non-linear. In a linear programme, the sequence of events is predefined, and learners must follow this learning path independent of their preferences. Non-linear programmes give learners control to structure the learning process, e.g. by selecting topics which interest them.

Programme & Component Level

The described characteristics are summarized in Figure 3.1. The figure is split into two parts: Figure 3.1.a shows features relevant to the programmes in general, and Figure 3.1.b shows dimensions applicable to the education components of an education programme. We use "education component" to refer to any teaching or learning format within an education programme. Such components can be composed of other education components. Since an education programme is composed of education components, the dimensions applicable to education components also apply to programmes.

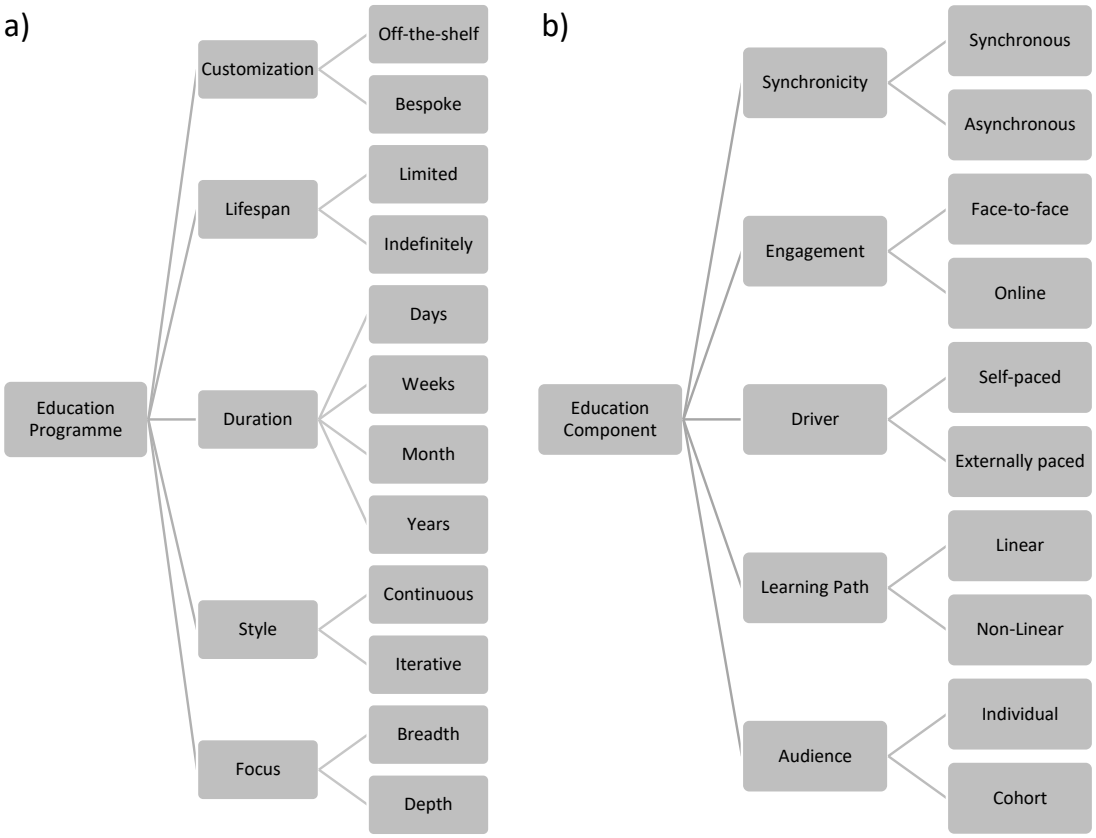

Figure 3.1: Features to classify (a) education programmes and (b) education components

3.1.2 People

As depicted in Figure 3.2, we distinguish between two main groups of people involved in bespoke professional education programmes: (1) companies that request education programmes for their employees and (2) education providers that provide such programmes. Each group can be subdivided into different roles. Meaning we do not refer to specific individuals but rather a role a person can have. A person may have multiple roles simultaneously, and throughout the lifetime of a programme, it is typical that the persons having specific roles change. Also, one role may be occupied by multiple persons jointly. Examples follow of the relation between persons and roles and how it can change as we discuss the different roles.

Roles

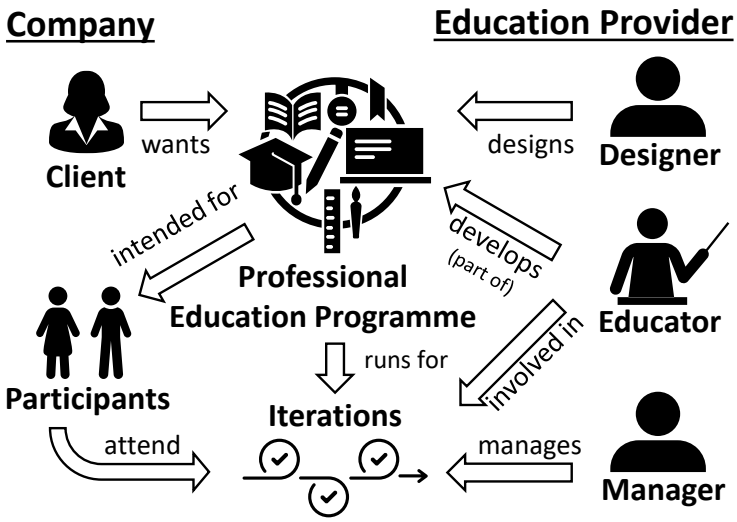

Figure 3.2: Visualization of people involved in a professional education programme (based on [WE22b])

On the company's side, we can differentiate between clients and participants. Clients represent the interests of the company regarding the requested education programme. They work together with education providers to create and run a programme. Typically, the client role is occupied by more than one person, each of which may focus on different interests like learning goals or costs. The client role could be subdivided into further roles like human resources, procurement, work council or accounting. However, we do not make this distinction as it is irrelevant to our approach since our focus is assisting education providers in design and management activities. In the future, a more fine-grained distinction of the client role may be beneficial to assist in dealing with different sub-roles.

Clients represent company interests

Participants / Audience

Participants are the target audience (the learners) of an education programme. Knowledge about participants is crucial for the design of a programme. For instance, accessibility needs or differences in prior knowledge impact a programme's design. Depending on the programme, participants can enrol on their own or are selected for a programme. Aside from entry requirements, MOOCs are open for everyone to enrol. Participants for talent development programmes are usually selected based on their achievements. In onboarding programmes, the participants automatically qualify when joining a company.

Designers, Educators and Managers

On the side of education providers, we can distinguish between designers, educators and managers. Designers are responsible for designing professional education programmes that address the needs of a company. For this, designers closely work together with clients. Once the learning goals of a programme, the intended audience and the client's environment are understood, designers involve educators. Together, educators and designers develop the educational activities that help reach these learning goals, are suited for the audience and fit the client's environment. Different types of educators exist, e.g. instructors, coaches or subject matter experts, and they are involved in developing activities and their execution. While educators are involved in executing individual activities, programme managers are responsible for delivering a programme. For instance, managers schedule the activities, book rooms, provide participants and educators with information and collect feedback.

Relation of Designer/Manager

Especially during the early iterations of a programme, the roles of designer and manager are often filled by the same person. Otherwise, a close collaboration between them is necessary as the lines between designing and managing a programme can be blurry. Commonly, designers or managers are also educators who contribute to or run activities in a programme.

Figure 3.3 relates the roles to the ADDIE phases and indicates which tasks are performed and tools are used in these phases. It illustrates how the involvement of the different roles changes over time. Here, we focus on the involvement of roles. Tool usage is further discussed in Section 3.1.4.

Role Involvement

The analysis mainly involves designers and clients. During the design of a programme, educators and managers are getting more involved. The development resides on the side of the education provider, and less involvement of clients is needed. During the implementation, managers and educators play the dominant roles in delivering a programme. Of course, participants are involved as learners, and clients may contribute as well. Managers conduct evaluations, and results are discussed jointly with designers and clients. Participants or educators are in evaluations involved when necessary.

ADDIE Phase	Analysis	Design	Development	Implementation	Evaluation
Roles	Designer, Client	Designer, Client Educator, Manager	Educator, Designer, Manager	Manager, Educator Client, Participants	Manager, Designer, Client Educator, Participants
Tasks	Identify Goals, Understand Target Audience, Gather Options, ...	Structure Content, Align with Objectives, Find Educators, Plan Delivery, ...	Preprare Instruction, Setup Learning Environment, Gather Materials...	Prepare Educators and Participants, Deliver Programme, ...	Gather Feedback, Identify Need for Change, Plan Actions, ...
Tools	(Digital) Whiteboards, Office Applications	(Digital) Whiteboards, Office Applications, PMT, Authoring Tools	PMT, Authoring Tools, Office Applications, LMS	LMS, PMT	Survey Tools, LMS, PMT

Figure 3.3: Example process journey of a professional education programme

3.1.3 Processes

Some Instructional Design processes entail all life cycle phases of education programmes, like concrete ADDIE-based processes [Bra09] or ASSURE [HMRS02]. Other processes like SAM [AS12] or Dick & Carey [DCC05] exclude the delivery of a programme. This exclusion is understandable since the processes for delivering a programme are partially the output of a design process. We visualize this in Figure 3.4. **For the remainder of this thesis, we distinguish between design and delivery processes.**

Design vs. Delivery

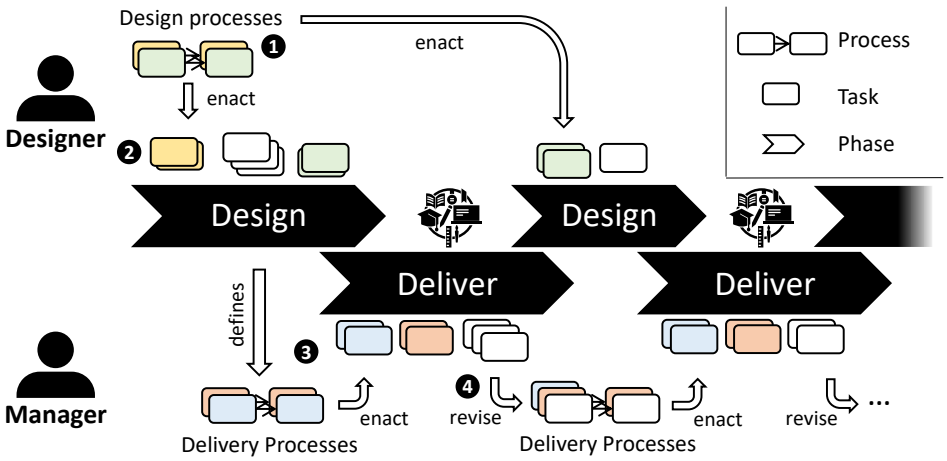

Figure 3.4: Visualization of the design and delivery of professional education programmes (based on [WE22b])

Design processes (see ❶) are typically based on a designer's expertise or Instructional Design process models from literature, such as ADDIE [GWGK05], SAM [AS12], Dick & Carey [DCC05] or ASSURE [HMRS02]. Design processes are concerned with analysing what is needed, designing a programme addressing the needs and developing the actual content. Furthermore, design

Design Processes

processes also cover the adaption of a programme based on evaluations. When enacted, these processes lead to tasks that are performed by the parties involved in a programme (see ❷). Managing the design processes and the related tasks is the duty of the programme designer.

Delivery Processes

Delivery processes are concerned with managing running programmes and are the programme manager's responsibility. Delivery processes can be developed during the design of a programme (see ❸), exist for specific locations or event types, or are based on a manager's expertise. For instance, conference hotels have a set of delivery processes to host conferences, e.g. the reception of delegates or the organization of banquets.

Overlapping Processes

As shown in Figure 3.4, design processes often come to a halt during delivery and continue afterwardss. However, these processes can run in parallel to some degree. For instance, assume a programme consists of two one-week modules. When the first module runs, the second may still be under development. For iterative programmes, the design and delivery processes are nearly cyclic. The feedback and assessment results collected when delivering a programme and changing client needs are the input for continuing the design after delivery. During the design after delivery, the delivery processes are also revised to improve the delivery of the next iteration (see ❹). Also, programmes can run for multiple iterations simultaneously, leading to multiple parallel delivery process instances.

3.1.4 Further Characteristics

Beyond products, people, and processes, there exist other characteristics like the used tools, the portfolio of an education provider, partners or the usage of platforms. We briefly cover these characteristics in the following.

Tool Usage

In addition to the involvement of the different roles, Figure 3.3 also relates tools to specific phases. Office applications like Word or Excel are typically present in every phase [CR19]. Especially in early phases, (online) whiteboards are used to gather analysis results or sketch and communicate designs. After analysis, project management tools like OpenProject[2] or Jira[3] play a significant role in managing the work that has to be done. During development, authoring tools like Articulate Storyline[4] or Adobe Captivate[5] can assist in creating e-learning content. For other types of content, dedicated tools exist as well,

[2] https://openproject.org
[3] https://atlassian.com/software/jira
[4] https://articulate.com/360/storyline
[5] https://adobe.com/products/captivate

but using applications like PowerPoint, Keynote or other office applications is also common. Learning management systems (LMSs) like Moodle[6] or Ilias[7] can be used to provide content, manage participants, track progress or perform evaluations. Companies and education providers may run their own LMSs or obtain them as a service. Please note that the feature sets of such LMSs differ [CZ14], and it cannot be assumed that a specific LMS can be used, e.g. a client may demand a particular LMS because it is already used internally. Dedicated tools may be used for evaluation when an LMS does not adequately address this functionality. While not listed explicitly, communication tools like Slack or Microsoft Teams are relevant in every phase.

Education programmes do not have to be developed from scratch. Instead, education providers have a portfolio of educational content that can be adapted and reused in new programmes. The portfolio can contain OTS programmes and individual parts of (bespoke) programmes, like content on a specific topic. The portfolio of an education provider can be a differentiator when clients select an education provider. The better the portfolio, the faster (and ideally cheaper) a programme can be developed.

Portfolio

Education providers cannot always address all relevant topics for a programme. In that case, education providers leverage relationships with partners to fill gaps in their expertise or when they need more resources to provide additional services. For instance, a university might employ external educators for topics they do not address.

Partners

Especially for e-learning material, education platforms are a source from which education providers can obtain additional content. On education platforms, education providers can offer their content to other companies (incl. other education providers) or individuals. The provider of an education platform can be an education provider or a third party. edX[8] or Udemy[9] are examples of dedicated providers of education platforms that provide the infrastructure and not the content. Companies or individuals can get subscriptions to these platforms and gain access to specific content. Instead of accessing content on the education platform, standardized formats like SCORM [Rus23b] or xAPI [Rus23a] allow the content to be integrated into other environments.

Platforms

[6] https://moodle.com
[7] https://ilias.de
[8] https://edx.org
[9] https://udemy.com

3.1.5 Other Types of Education Programmes

Difference to School/University Programmes

The characteristics described in the previous sections focus on professional education programmes. However, there are several other types of education programmes, like primary and secondary education programmes at schools and study programmes at universities. While the general characteristics of education programmes are similar, there are differences in the details. For instance, for university study programmes, there does not exist a client requesting a programme and instead, accreditation agencies play a significant role. For primary school programmes, spatial and temporal flexibility in the delivery of programmes is less relevant since students are required to attend school. The approach for designing and managing professional education programmes presented in this thesis can be helpful for other types of education programmes. However, specific parts can be less relevant or require adaptation.

3.2 Purpose and Requirements

The purpose of our modelling approach is derived from Research Question RQ1, which tries to determine how modelling professional education programmes can (1) facilitate collaboration, (2) assist in work organization and (3) provide information about the situational context. Based on the purposes, we define the requirements a modelling approach has to fulfil. We split these requirements into language and expressiveness requirements. The former focuses on the design, usability and tool support of the language. Expressiveness requirements determine what shall be expressible by the language.

3.2.1 General Language Requirements

The following describes general Language Requirements (LR). For each requirement, we provide a rationale for its relevance.

LR1 **High-Level Perspective**: *The modelling language shall focus on the high-level perspective of professional education programmes.*

The characterisation of professional education programmes in Section 3.1 shows that programmes can vary in multiple dimensions. While there are commonalities, the planning, development, documentation and enactment can differ depending on a concrete programme's characteristics. For instance, a self-paced e-learning programme is very different to a one-week face-to-face programme. Consequently, there is a trade-off

between being able to describe every scenario in detail and the language's complexity and comprehensibility. We distinguish between the high-level and low-level perspectives of a programme. The high-level perspective is concerned with the programme from an abstract, conceptual point of view, e.g. what is required (learning goals, content items), who the audience is, and the programme's structure necessary to reach intended goals. In contrast, the low-level perspective entails how a programme is represented in detail, e.g. the representation of a programme within an LMS. This high-level perspective is mostly documented informally [CR19]. Consequently, the comparability between programmes and the identification of reusable parts is negatively impacted due to the differences in their documentation. The low-level perspective becomes more important, starting with the development of the learning resources and during programme implementation. Dedicated tools exist to define programmes on a low level, e.g. the education design assistant MyScripting [ME22]. It may not even be necessary to address the low-level perspective since certain parts of a programme may be addressed by (off-the-shelf) content provided by partners or platforms.

LR2 **Programme-Work-Mapping**: *The modelling language shall provide a mapping between entities of a programme and the work items (tasks) related to these entities to assist in work organization.*

Work can be organized differently, e.g. using imperative processes that define the sequence of tasks explicitly or rather declaratively by stating the tasks that have to be done and their dependencies [FLM+09]. For the former, languages like BPMN can be used to express the processes. For the latter, checklists or declarative process modelling languages can be used. We cannot assume that an imperative process-driven approach suits every company or project. However, focusing on tasks and their interdependence is compatible with any type of work management. A mapping of work items to entities of a programme is necessary to get an overview of pending work items concerning individual entities.

LR3 **All Phases**: *The modelling language shall cover all phases of professional education programmes.*

Since the models of professional education programmes shall be used for work organization and to determine the situational context, it is vital that programmes can be modelled across their entire life cycle.

LR4 **Lightweight**: *The modelling language should be lightweight.*

Adopting a new modelling approach comes at a cost since all parties involved need to be able to use the language and make sense of the resulting models. Consequently, creating a complex modelling approach may hinder its adoption, and a lightweight approach should be preferred. According to [Bot08], a modelling language is lightweight when it is easy to learn, allows brainstorming, enables quick design sketches and serves as a starting point for more detailed planning.

LR5 **Formal Metamodel**: *The modelling language shall have a metamodel.*

Not all modelling languages have a formal metamodel. Some are explained informally through natural language texts and examples. For instance, E^2ML [Bot06] and the first version of the UML are described without a formal metamodel. With a formal metamodel, programmatically accessing information or syntax validation is simplified.

LR6 **Scalability**: *The modelling language shall support the definition of professional education programmes of different scales.*

A professional education programme can last a few days up to years and consist of various modules. A modelling approach must provide means to manage complexity when used for larger-scale programmes.

LR7 **Extensibility**: *The modelling language shall be extensible to capture information beyond what is specified in the initial language design.*

We already stated that creating a modelling language that covers every aspect of professional education programmes is not feasible. All programme designers and managers have their own experience and might prioritise certain information differently, even on a high level of abstraction. Hence, the language has to be extensible to capture information beyond what is specified in the initial language design.

LR8 **Online Whiteboard-compatible Visual Syntax**: *The modelling language shall have a visual syntax usable in collaborative online whiteboards.*

Designing bespoke professional education programmes is not the effort of a single person but rather a collaboration of designers, managers, educators and clients. Models can be a focus point and communication vehicle for all parties involved. In remote settings, working on models collaboratively on online whiteboards is beneficial for collaborative design efforts [LZXL21].

3.2.2 Expressiveness Requirements

In the following, we specify a set of Expressiveness Requirements (ER), which are derived based on the language requirements, the fundamentals described in Chapter 2 and the characteristics explained in Section 3.1. These requirements state what shall be expressible by the modelling approach.

ER1 **Objectives**: *Objectives and their level of application must be specifiable.*

Section 2.1.2 explains that objectives are a good starting point for educational endeavours. Objectives exist on different levels of granularity, e.g. they relate to the overall programme or just to an individual activity in a single session.

ER2 **Stakeholders**: *The stakeholders of a programme shall be specifiable.*

We introduced several internal stakeholder roles in 3.1.2, e.g. designer, manager or participant. Additionally, there can be external stakeholders like partners or platforms which can influence a programme.

ER3 **Environment**: *The environment of a programme shall be specifiable.*

The environment of a professional education programme significantly impacts a programme's design. Hence, education providers must be able to specify information about the environment, like different participant types or resources that could be used.

ER4 **Comments**: *Comments regarding model elements shall be specifiable.*

As Karsai et al. [KKP+09] state, it is helpful for any modelling language to permit comments. Comments help clarify complex model elements or express things that go beyond the expressiveness of the language.

ER5 **Decisions**:*Design decisions shall be specifiable.*

Often, there is no clear winner between design alternatives, and trade-offs are necessary. For instance, the appropriate length of synchronous online teaching days is debatable. Documenting design decisions creates clarity, and people can revisit decisions.

ER6 **Tasks**: *Tasks and their relation to a programme must be specifiable.*

Tasks to design and deliver a programme have to be performed by designers, managers, educators and clients. Keeping track of these tasks is crucial to deliver a high-quality programme (cf. LR2).

ER7 **Education Component**: *Education components representing teaching and learning formats and content shall be specifiable.*

An education programme is not a monolith but rather composed of several individual components that contribute to reaching the objectives, like a lecture or a tutorial. Such components are not uniform and can follow different teaching and learning formats.

ER8 **Hierarchies**: *Subparts of complex elements shall be specifiable.*

It shall be possible to break down complex elements like education components or objectives into smaller elements.

ER9 **Temporal Order**: *The temporal order of components shall be specifiable.*

Components of a programme typically have a temporal order. Such orders define when a specified component is held, which requires the definition of date, time and duration.

ER10 **Dependencies**: *Dependencies of programme parts shall be specifiable.*

Aside from hierarchies and temporal order, other dependencies exist between the parts of an education programme. For instance, a component can be a prerequisite to another component.

ER11 **Delivery Modes**: *Delivery modes shall be specifiable.*

Programmes and their subcomponents can have different delivery modes. For instance, in a blended learning programme, some parts can be asynchronous and digital, while others are face-to-face.

ER12 **Resource Usage**: *Resource usage shall be specifiable.*

Teaching and learning activities use resources like exercise sheets, flipcharts or software. A modelling approach must be able to state which resources are required and when.

ER13 **Location**: *The location of the teaching/learning shall be specifiable.*

In the case of no spatial flexibility, it must be possible to denote where the teaching/learning occurs. This can be a physical or digital location.

ER14 **Evaluation**: *Evaluation strategies and results shall be specifiable.*

We discussed the role and different types of evaluation in Section 2.1. Evaluations must be planned, and the results provide valuable insights into a programme's quality.

3.3 Related Work

In the following, we analyse related work to see to which degree the existing Instructional Design modelling approaches cover the requirements stated in the previous section. The focus while discussing related work is on the language requirements. Table 3.1 summarizes the coverage of the language requirement of the approaches presented in the following. We summarize the coverage of expressiveness requirements at the end of this section.

Table 3.1: Language requirements coverage of related modelling approaches

Approach	LR1	LR2	LR3	LR4	LR5	LR6	LR7	LR8
IMS-LD [IMS03]	o	-	-	-	+	+	+	-
Learn-CIAN [MJdlC+09]	+	-	-	+	+	-	-	o
MOT+(LD) [PLLC11]	+	-	o	+	+	o	o	+
E^{2}ML [Bot06]	+	-	-	+	-	o	+	+
HLAN [NS14]	+	-	-	+	o	o	-	+
STOPS [APH14]	+	-	-	+	+	+	-	-
PoEML [CLA14]	o	-	o	-	+	+	-	-
MyScripting [ME22]	o	-	-	o	-	-	-	o
UML [Obj17]	+	-	o	-	+	+	+	o
PCM [Dou08]	+	-	-	+	+	-	+	+
coUML [DMP08]	+	-	o	-	+	+	+	o

IMS-LD [IMS03] is an XML-based language for describing online learning scenarios. The language uses a theatre metaphor to describe learning designs. A learning design is described as a play consisting of acts in which roles perform activities in an environment. For instance, one activity in an act could be a reflection by the learner on the purpose of an exercise. The language is divided into three levels (A, B, C). Level A, the top-most level, introduces concepts to describe learning designs based on the theatre metaphor. Level B provides concepts to express conditions, and Level C introduces notifications. IMS-LD is expressive enough to describe learning designs that are executable with engines like CopperCope[10].

IMS-LD

IMS-LD itself does not provide a visual notation. However, UML diagrams such as Activity Diagrams can be used to visualize parts of the specification, and several languages compile to IMS-LD-compliant designs [PL06, MJdlC+09]. Due to its verbose textual syntax and focus on executability, the approach is relatively low-level and heavy-weight. It mainly addresses the design, development and implementation phases. Since IMS-LD is XML-based, it can be extended using custom tags belonging to a different namespace.

Visualization of IMS-LD

[10] https://coppercore.sourceforge.net

Learn-CIAN [MJdlC+09] is a visual modelling language for describing learning processes using a graph-based notation. Learning activities in a Learn-CIAN design are represented as nodes. Each node has a fixed set of compartments for different information: the name of the activity, involved roles, modality, activity type and manipulation of learning objects. Control flow is defined using edges and control flow nodes, e.g. to define the start, end or concurrent activities. The visual syntax is complex, which makes the use of Learn-CIAN within online whiteboards challenging. The approach is high-level and lightweight but focuses on activities and cannot describe more extensive programmes. The language is described using its visual syntax, and a formal metamodel is not described. Nevertheless, the tooling is based on Eclipse and the Eclipse Modelling Framework (EMF), which implies that a formal metamodel exists. Extension capabilities are not mentioned. A prior version, CIAN [JMG+08], can also be translated into IMS-LD.

LEARN-CIAN

MOT+[PLLC11] is a visual language heavily used as part of the MISA method for Instructional Design. The language follows the idea of concept maps and has a two-level metamodel able to describe classes of objects (concepts, principles and procedures) as well as concrete instances (examples, statements and traces). The approach is high-level and lightweight, but the associations between elements all use the same visual notation and the semantics are only described by a single-letter label. For novice users, it can be challenging to distinguish and interpret these associations. A specialized version called MOT+LD [PL06] has been developed for describing learning designs that can be translated into IMS-LD. Theoretically, MOT+ can be used in all phases due to its generality, but it does not offer any modelling concepts that are specific to any of these phases. The specialized version, MOT+LD, adds a few constructs like roles, activities and resources specifically useful for the design and development phase. MOT+ covers most of our requirements, but mainly due to its generality. While MOT+LD shows potential extensibility of the language, no extension mechanism is described. Creating subtypes of the metamodel classes seems possible, but no addition of new associations.

MOT+(LD)

E^{2}ML [Bot06] is a semi-formal approach with three different perspectives to describe learning designs. The activity diagram uses a graph-based approach similar to flow charts or activity diagrams to express sequences of actions. An action is any activity in a teaching/learning process. Sequential and parallel ordering of actions is possible, and timings can be specified informally as side comments. The second perspective, the dependency diagram, allows for defining dependencies between actions, e.g. learning prerequisites. The

E^{2}ML

third perspective is a template to describe actions by defining the initial state (requirements, preconditions, inputs), the final state (outcomes, side-effects, outputs) and the action performance (procedures, durations, tools, locations). E^2ML is a high-level, lightweight approach focusing mainly on the design phase. It is only informally specified but allows flexible extension since no metamodel constraints can be violated. The language has been defined for sketching designs on traditional whiteboards or paper.

HLAN [NS14] is an approach based on extended Petri nets for modelling instructional designs. It can be used to describe a temporal ordering of learning activities and required resources. These designs can be used in the evaluation phase for learning analytics [TdMFS+17]. The syntax is whiteboard-compatible, but due to the dominance of process modelling languages like BPMN or UML Activity Diagrams, an approach based on Petri nets is less intuitive for novices. Since it is based on Petri nets, a metamodel can be inferred, even though it is not explicitly provided. Extension capabilities are not mentioned. The Authoring Tool for Instructional Design (ATID) [BN19] allows modelling with HLAN, and it enables integration with the LMS Moodle.

HLAN

STOPS [APH14] is a graph-based tool for study planning that can also be used for curriculum design. It has a formal metamodel that allows specifying outcomes, courses and dependencies between them. The approach is high-level, lightweight and focuses on the design phase. The visual syntax is coupled to the tooling and can only partially be reflected in whiteboards. The authors do not mention any extensibility features.

STOPS

PoEML [CLA14] is a multi-perspective educational modelling language. Its expressivity is comparable to IMS-LD, but in addition, PoEML has a visual syntax. The language cannot be considered lightweight due to its large metamodel consisting of 13 packages and complex visual syntax. The latter also limits its use for visual modelling of education programmes in collaborative online whiteboards. It can only be partially considered high-level since it is meant to describe executable designs.

PoEML

MyScripting [ME22] is an educational design assistant tool. Education designs are described as scripts, which are composed of metadata and a tabular description with learning phases as columns and topics as rows. For each cell of this table, a linear sequence of teaching/learning activities can be specified. The tabular description could also be used in online whiteboards, but the definition of metadata, like learning objectives, is tool-bound. The tool focuses on the phases from design to implementation and is on the edge between a high-level and low-level specification of a programme's design. Scripts can be

MyScripting

exported as Word documents or to LMSs such as Moodle, Ilias or the MOOC platform edX. While not explicitly linked to a specific design process, the tool requires a certain approach to describing educational designs and can only be considered somewhat lightweight. A large variety of teaching/learning formats can be used within scripts, but there is no feature to add additional formats. Moreover, a definition of an underlying metamodel is missing.

The UML [Obj17] is a modelling language that originated in the field of software engineering. Since it is a general-purpose language (GPL), it can be used to describe Instructional Design projects across all phases and to varying degrees of detail. However, no domain-specific concepts exist. Hence, language users have to decide how to represent certain programme elements, e.g. a learning objective. Furthermore, the UML is not one language but a family of languages with a complex unified metamodel and, as a whole, cannot be considered lightweight. Whiteboard-compatibility differs depending on the diagram type. For instance, Use Case Diagrams can easily be used in online whiteboards, but representing sequence diagrams can be more difficult. In general, using a dedicated modelling tool for the UML is the better choice. Like MOT+, the UML covers many of our requirements due to its generality.

UML

Performance Case Modelling (PCM) [Dou08] is an approach that uses UML Use Case Diagrams for performance analysis in the early stages of an Instructional Design project. PCM illustrates how to utilize a specific UML diagram type for Instructional Design. Use Case Diagrams alone are lightweight but insufficient to cover the entire life cycle of education programmes.

PCM

Domain-specific constructs can be added to the UML using UML Profiles [FFVM04]. A UML profile is mainly comprised of stereotypes that classify UML elements as domain-specific entities. coUML [DMP08] is such a UML profile for Instructional Design and adds stereotypes like «goal» or «document». Furthermore, coUML specifies how UML diagrams can be used to model Instructional Design projects. Due to its utilization of many UML diagram types, coUML is not lightweight.

coUML

In Table 3.2, we provide an overview of the expressiveness requirement coverage of the presented approaches. A detailed version is provided in Appendix A. Most approaches can express objectives, components, activities, temporal orders and dependencies. Only limited support for comments exists, and no approach has a notation of tasks or decisions. PoEML, coUML and MyScripting have the highest degree of expressiveness requirements coverage. PoEML and coUML also have good coverage of the language requirements but are too heavy-weight for our purposes.

Coverage of Expressiveness Requirements

Table 3.2: Summarized coverage of expressiveness requirements

Approach	Fulfilled	Partially Fulfilled	Not Fulfilled
IMS-LD [IMS03]	10	2	2
Learn-CIAN [MJdlC+09]	4	1	9
MOT+(LD) [PLLC11]	6	3	5
E^{2}ML [Bot06]	6	4	4
HLAN [NS14]	4	0	10
STOPS [APH14]	4	0	10
PoEML [CLA14]	10	1	3
MyScripting [ME22]	9	2	3
UML [Obj17]	5	5	4
PCM [Dou08]	5	2	7
coUML [DMP08]	10	1	3

3.4 Summary & Language Realisation

In this chapter, we presented characteristics of professional education programmes by mainly focusing on the 3Ps: product, people and process. Based on the characteristics and the Research Question RQ1, we defined requirements towards a modelling approach for such programmes. We divided these into language and expressiveness requirements and used them to analyse related work. Our analysis shows that current education modelling languages are insufficient for our purposes. Therefore, we present a new modelling language for describing professional education programmes in this thesis.

Regarding the language's realisation, we decided not to base our language on any existing education modelling language since they are partially too complex, do not cover the whole life cycle or are too generic. While creating a new modelling language comes at the cost of requiring new tool support, we mitigate this problem by basing our tooling on existing tools and focusing on their integration. For instance, we use collaborative online whiteboards for visual modelling tool support, which also relates to Research Question RQ2. Moreover, reusing or adapting tool support from other educational modelling languages would not have been feasible because tooling is mainly outdated or has not been made public as open-source.

The metamodel of our modelling language is presented in the next chapter. Visual modelling with online whiteboards and our language's visual syntax is covered in Chapter 5. The tool support for our modelling approach is the subject of Chapter 6.

Chapter 4

Professional Education Programme Modelling Language (PEPML)

This chapter presents the metamodel of PEPML, our extensible modelling language for describing professional education programmes. As mentioned in the previous chapter, the development of this language is based on the language engineering process from Brambilla et al. [BCW17], which we explained in Section 2.2.2. The previous chapter covered the first step, the *Modelling Domain Analysis*, and defined requirements towards a modelling approach for professional education programmes. This chapter, together with Chapter 5, covers the second step, the *Modelling Language Design*.

Section 4.1 provides an introduction to the metamodel by describing the metamodel's structure, conventions and stereotypes used for representing the metamodel and a running example to illustrate the language's use. Section 4.2 explains PEPML's metamodel, package by package. Section 4.3 focuses on the semantics of hierarchies, additional metamodel constraints and PEPML's extensibility. Throughout the chapter, we explain how the language and expressiveness requirements are addressed. The visual syntax of our language is covered in Chapter 5. The third phase of the language engineering process, *Modelling Language Validation*, is spread across Chapters 6 to 9.

4.1 Overview

Before explaining PEPML's metamodel, we provide instructions on how to interpret it. We explain the metamodel's structure and introduce stereotypes and conventions to reduce complexity. Furthermore, we present a running example to illustrate the use of the language.

4.1.1 Metamodel Structure

As shown in Figure 4.1, PEPML's metamodel is divided into seven packages. The main reason for splitting it into multiple packages is to improve comprehensibility and to be able to refer to specific parts of the metamodel.

The topmost package, *Foundation*, introduces basic language concepts used by all other packages. The *Work* package adds the possibility to define work items and statuses for programme elements (cf. LR2). The five packages below the *Foundation* package are named after the ADDIE phases. We use the same ADDIE colouring as in Section 2.1.3 for these packages and their classes.

PACKAGES

Naming packages after ADDIE phases shall illustrate that PEPML addresses every life cycle phase of a programme (cf. LR3). However, education providers are not required to follow ADDIE-based processes and can use concepts of the respective packages as needed.

NOT LIMITED TO ADDIE

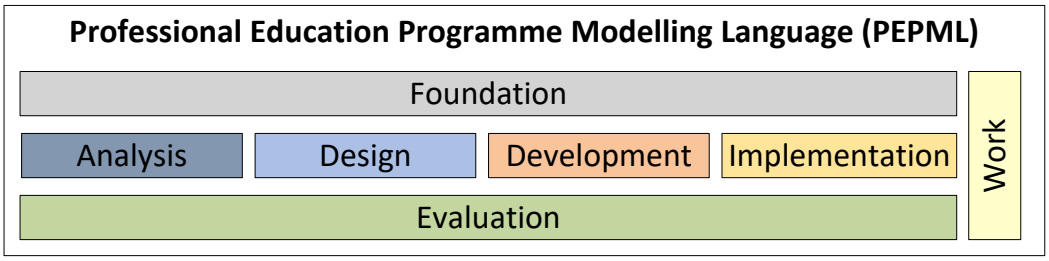

Figure 4.1: Overview of PEPML packages

4.1.2 Previous Versions of the Metamodel

In [WE22a], we first introduced PEPML. It was a short description of PEPML's features, visual syntax and tooling. The first tooling was static, and an extension of PEPML was difficult. Furthermore, the packages for development, implementation and evaluation were underspecified.

FIRST VERSION OF PEPML

In [WE23], we introduced the second version of PEPML. It included concrete extension capabilities, which were limited to adding new types of education components. The metamodel presented in that work is largely the same as the one presented in this thesis. The significant differences are further predefined entity types, a more powerful extension mechanism and a better specification of temporal orders. The latter was underspecified in [WE23]. Especially the tooling around PEPML, which we present in Chapter 6, changed significantly compared to our previous work. The metamodel is no longer hardcoded but instead an input for the tooling, which allows extensions of the metamodel and using the tooling for other modelling languages.

SECOND VERSION OF PEPML

4.1.3 Metamodel Interpretation Guidelines

Before explaining PEPML's metamodel, we introduce some guidelines required to understand the metamodel. In particular, we explain the default values we assume for multiplicities, discuss how we deal with repeating content and briefly mention the data types used. Furthermore, we present two stereotypes meant to reduce the complexity of the metamodel.

DEFAULT MULTIPLICITIES

The UML does not define a default multiplicity. For UML class diagrams, we assume 0..∗ as the default multiplicity for regular association ends. Furthermore, we imply a 0..1 for aggregations and a 1..1 for compositions. We list multiplicities explicitly when we diverge from the our default values. Furthermore, we imply a 0..1 multiplicity for any class property, which allows users to leave a property undefined.

PROPERTIES

A complete list of properties for a specific class is available in the package description to which a class belongs. When we refer to classes of other packages, we only list properties necessary to understand the package currently explained. In object diagrams, we only list properties with a relevant value, e.g. if a value of a string property is empty, it is omitted.

DATA TYPES

We use `boolean`, `number` and `string` as basic data types. As in JavaScript, the data type `number` allows integers and floating point numbers. In addition, we support the data types `Date`, `Time`, `DateTime`, `URI` and `Range`. The first three are ISO 8601-compatible strings. `URI` can be used to refer to external data. `Range` expresses a number range with a minimum and maximum value.

STEREOTYPE «ENTITY»

In Section 4.2.1, we introduce `Entity`, which is the superclass for most classes in PEPML's metamodel. Explicitly showing the inheritance for all relevant classes would introduce unnecessary complexity. We add the stereotype «Entity» to classes where the inheritance association is missing. Figure 4.2 shows how this stereotype has to be interpreted.

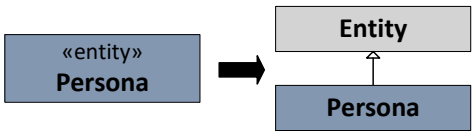

Figure 4.2: Translation of the Stereotype «entity»

STEREOTYPE «RELATION»

Instances of `Entity` or its subclasses can be connected via an instance of `Relation` (see Section 4.2.1). We use the stereotype «relation» to indicate that an association between `Entity` subclasses is a refinement of `Relation`. Figure 4.3 shows how this stereotype is translated. The grey-coloured associations indicate the implied associations enforced by the OCL constraint.

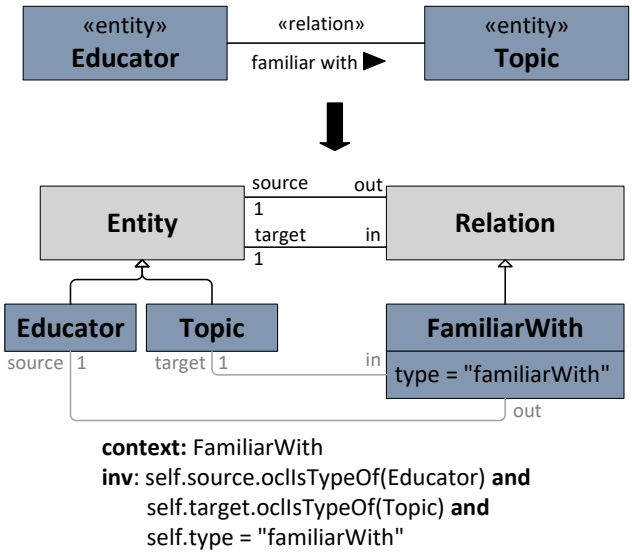

Figure 4.3: Translation of the Stereotype «relation»

4.1.4 Running Example

We use a running example throughout the following chapters to illustrate the design and management of professional education programmes and how our approach can assist education providers. Our running example is an extended version of the scenario we used in [WE23].

Running Example: Agile Transformation Programme

An IT company is starting its shift from traditional processes to applying Scrum. The company approaches an education provider to provide a professional education programme that supports its Agile transformation endeavour. The goal of the programme is to familiarize participants with the idea behind Agile processes and particularly prepare them to apply Scrum. The working title for the education programme is the "Agile Transformation Programme". Based on these initial goals, the client and the education provider start designing the programme collaboratively. Remote collaboration is required since the clients' offices are not located conveniently.

4.2 Metamodel

As described in the previous section, the metamodel is clustered into different packages, which are one by one explained in this section. The colours of the metamodel classes match those of the packages shown in the package overview in Figure 4.1.

4.2.1 Foundation Package

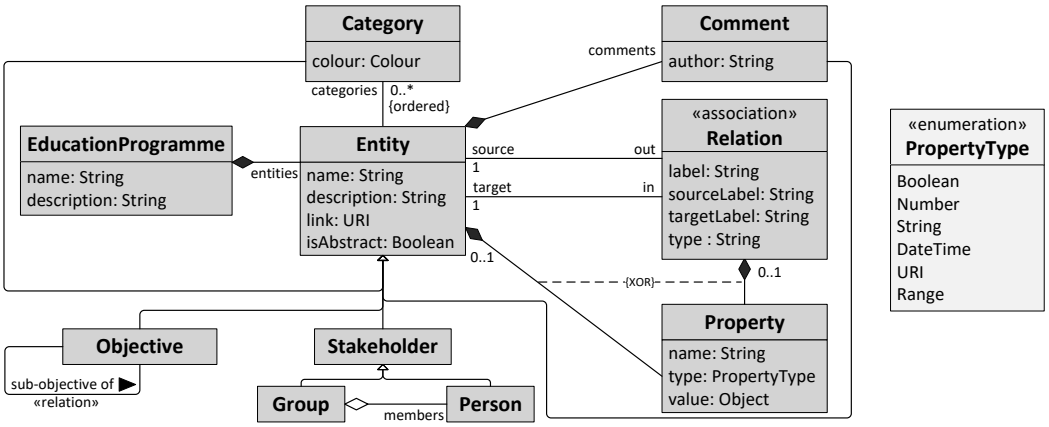

Figure 4.4: *Foundation* package of PEPML

The *Foundation* package is shown in Figure 4.4. It is the basis for all other packages and defines that an education programme is composed of entities. Consequently, `EducationProgramme` is the root class for PEPML models, meaning that every instance of `Entity` is linked to an education programme. With a few exceptions, most classes of the metamodel are subclasses of `Entity`, which makes it the base class of the metamodel. Objects of these classes can be treated uniformly as entities when necessary and have common properties such as `name` and `description`.

ENTITY

The class `Relation` allows linking two entities. The relation expressed by this class is intentionally generic since we can only foresee relevant relations to a certain degree. We refine this class in the other packages to describe relations we believe to be necessary and allow users to use the generic class to denote further relations which they identify as relevant (cf. LR7). The `type` property allows to cluster relations of the same type. Relation labels are used within the concrete syntax to provide further information on the relation's semantics. Similar to an entity, a relation can have arbitrary properties.

RELATION

Based on the class `Property`, we can define custom properties for `Entity` and `Relation`, which is essential for extensibility (cf. LR7). Education providers can use custom properties to reflect information that does not correspond to any of the existing properties. We support a fixed set of property types. The type `URI` is noteworthy since it can refer to the external tools in which an entity may be described in greater detail. For instance, URIs can refer to additional documents on Cloud file storage solutions. The default link for an entity can be specified using the `link` property of `Entity`.

PROPERTY

OBJECTIVE

`Objective` is the base class for all objectives (cf. ER1). In other packages, we add further associations to this class and define subclasses. In the *Foundation* package, the class can be used to denote programme objectives.

STAKEHOLDER, PERSON, GROUP

`Stakeholder` is a base class for stakeholders involved in a programme (cf. ER2). We introduce two subclasses for denoting persons and groups. The property `isAbstract` states if an object expresses a concrete or an abstract stakeholder. A concrete person could be a specific instructor, and an abstract stakeholder would instead be the role instructor. We introduce further subclasses for more specific types of stakeholders in other packages.

CATEGORY

Entities can be clustered into categories. Each category has a colour to identify entities of the same category. The categories of each entity are ordered. If an entity belongs to multiple categories, the colour of the first category is used in the concrete syntax. Grouping entities into categories partially addresses ER10.

COMMENT

It is generally a good practice to allow comments in models [KKP+09], which is why comments are listed among the expressiveness requirements (cf. ER4). Comments can be attached to any entity. In addition to a name and a description (both inherited), a comment includes an author, which is represented by a `String`. Not necessarily every comment author has to be represented as a person within an education programme, which is why we decided against an association with `Person`.

4.2.2 Work Package

The *Work* package is depicted in Figure 4.5. It contains classes to denote tasks, decisions and statuses for programme entities (cf. LR2). Furthermore, constraints can be defined that must hold if tasks have a certain status.

The information about tasks is kept slim in the metamodel, and additional information is maintained outside the model, e.g. in project management tools. The rationale behind this is that project management tools are specifically designed to manage tasks but are usually agnostic to the type of projects being managed. Hence, such tools are unaware of the structure of an education programme. Consequently, designers and managers must manually define how the tasks contained in a project management tool relate to entities of an education programme. The *Work* package allows specifying task-entity-mapping in PEPML models. This information can be used to automatically reflect this mapping in project management tools, e.g. via a task hierarchy or labels. We provide more information on this in Section 8.2.2.

RELATION TO PROJECT MANAGEMENT TOOLS

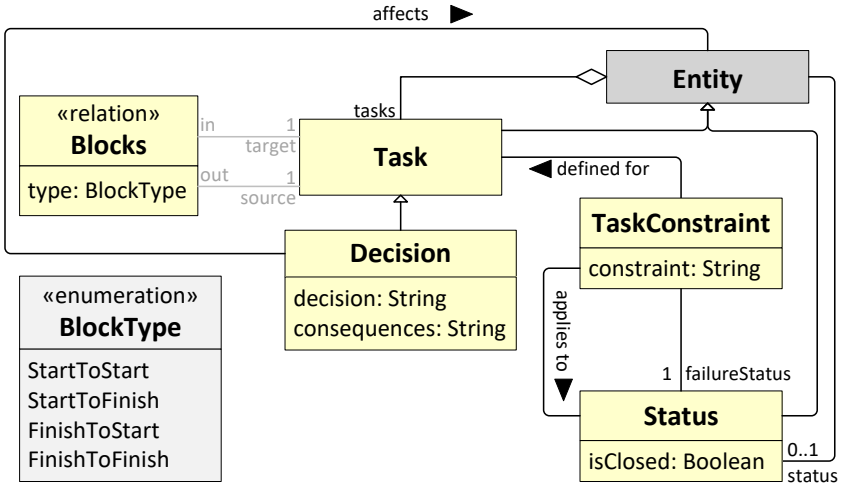

Figure 4.5: *Work* package of PEPML

Tasks can have a status due to the inherited association from `Entity`. Users define which statuses exist, e.g. "Open", "In Progress", "Done" or "Obsolete". The semantics of these different statuses are largely out of the scope of our language. We only differentiate if a status marks a task as closed or not. If a task is in a closed status, no further work by a user is required any longer, either because the work has already been done or because the task is no longer relevant. "Done" and "Obsolete" are typical examples of closed statuses.

STATUS

We decided against introducing a control flow mechanism allowing the definition of imperative process models [FLM+09]. Instead, we allow a lightweight form of declarative process modelling that enables the definition of task dependencies. Since a class is also an entity, it can itself have tasks, which enables the definition of task hierarchies. Furthermore, we allow to define that a task blocks another task and support specifying common blocking styles [Tsi18]. For instance, `FinishToStart` means that the blocked task can only start when the blocking task is finished. Further task dependencies can be expressed via `Relation` but have pure annotation purposes.

TASK DEPENDENCIES

`Decision` is a special form of `Task` that can be used to document design decisions (cf. ER5), which is helpful for collaborative design. The context of the decision can be expressed in the description (inherited property from `Entity`) of the decision. Two further properties allow denoting the decision itself and the consequences, similar to Architecture Decision Records in software engineering [Kee22]. A status can be used to classify if a decision has been made or if it is still pending.

DECISION

As the metamodel suggests, a status can be assigned to any entity. Since we do not prescribe any statuses, the semantics of statuses are defined by

Using Statuses to Classify Entities

language users. In the left part of Figure 4.6, we use the three statuses "Option", "Selected" or "Rejected" to classify entities. The latter two are closed statuses. In particular, the example uses these three statuses to classify the programme objectives. Especially during the inception of a new programme, several possible objectives may exist, and the example illustrates how an entity status can be used to distinguish between possible, selected and rejected options. This technique to classify entities with statuses plus the subtypes of entities, which are introduced in the other packages, allows the description of an education programme's environment (cf. ER3).

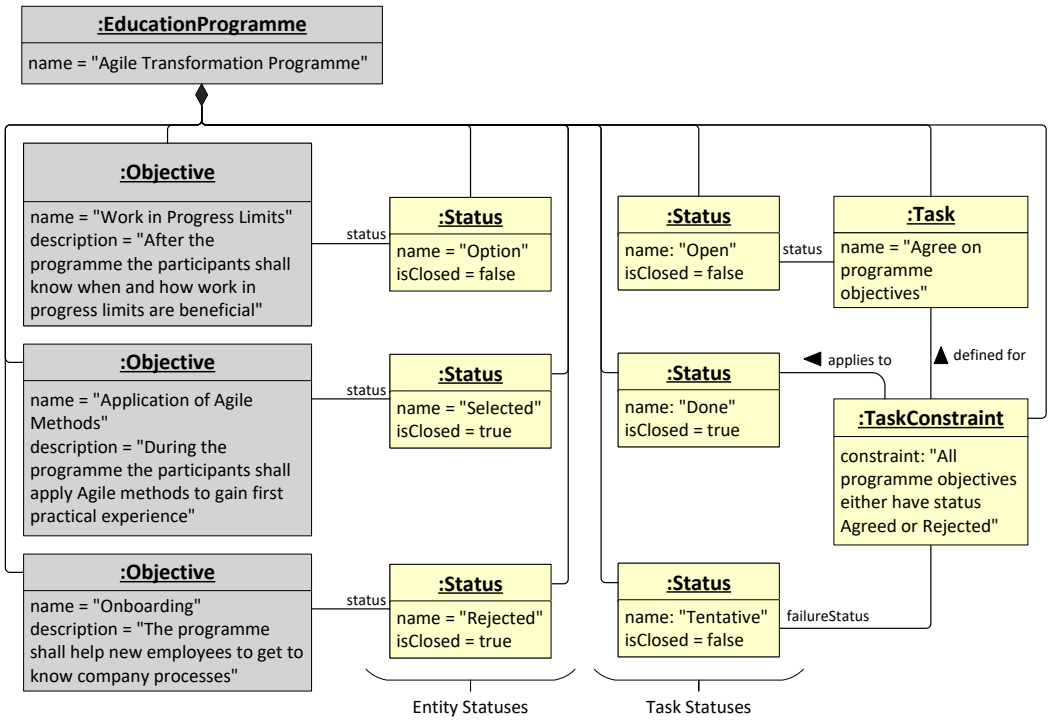

Figure 4.6: Example of assigning statuses to entities and tasks

TaskConstraint

A typical problem in modelling is that models and reality get out of sync. This can be problematic if a model shall serve as the situational context for providing recommendations (cf. RQ1). To address this problem, we allow to define constraints, ensuring that status changes of tasks are only possible if certain conditions in the model are met. In the right part of Figure 4.6, we define a task constraint for "Agree on programme objectives". This constraint applies to the "Done" status. If a task gets a status for which a constraint is defined, the constraint is evaluated, and only if it yields true can it remain in this status. Otherwise, the task is put into the failure status defined by the constraint. The constraint in the example is specified informally. We explain in Section 8.2.2 how executable constraints can be defined.

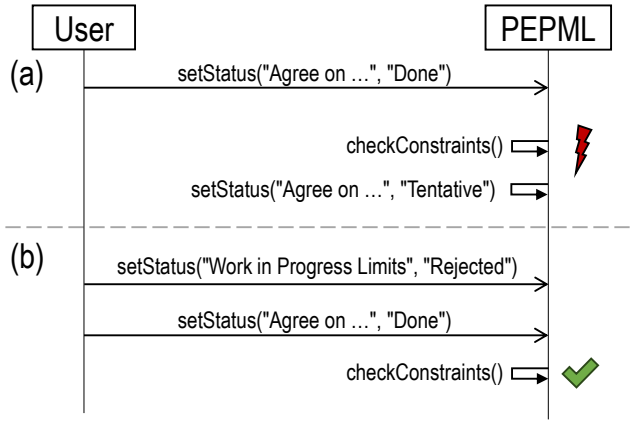

Figure 4.7: Example of task constraint validation

Figure 4.7 illustrates the constraint evaluation for the example of Figure 4.6. In part (a) of the figure, a user changes the status of the task "Agree on programme objectives" to "Done". Since the objective "Work in Progress Limits" is still in status "Option", the constraint is not satisfied. As a consequence, the task gets the status "Tentative". In part (b) of the figure, the user changes the status of the objective to "Rejected" and can successfully close the task.

Constraint Validation Example

4.2.3 Analysis Package

The *Analysis* package shown in Figure 4.8 contains classes to document findings from the analysis phase. Please note that during the analysis phase, it may be necessary to use entity types defined in other packages, e.g. to document possible locations or existing software.

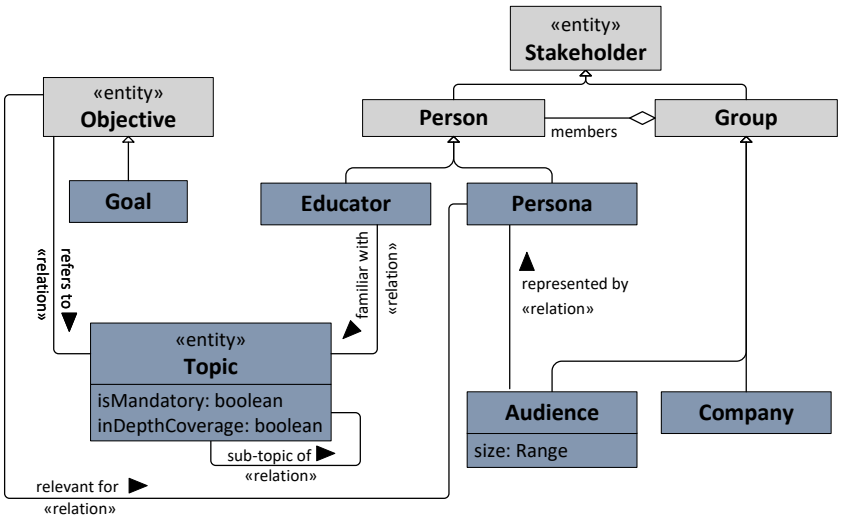

Figure 4.8: *Analysis* package of PEPML

We add several subclasses of `Stakeholder` to describe abstract stakeholders (cf. ER2), like personas representing stereotypical participants, the company requesting a professional education programme and the audience of a programme as a whole. The latter two are examples of stakeholder groups which can have multiple members. For instance, participants are the members of the audience. We introduce a class to represent participants in the *Implementation* package in Section 4.2.6, which covers runtime information about a programme.

Persona, Audience, Company

Dedicated classes representing the company or the audience are vital to model the environment (cf. ER3). Relevant information can be attached to objects of these classes to model situational factors, e.g. the size of the audience is typically an essential situational factor that influences possible teaching formats. We do not predefine additional properties since they can be added as needed as custom properties. For instance, a large variety of canvas models to define personas exists. Most of them have slightly different properties, and due to the extensibility of PEPML, users can choose which template they want to use and store the information in custom properties.

Modelling the Environment

While the *Foundation* package already describes the base class for objectives, the *Analysis* package adds the subclass `Goal`. We explained our spectrum of objectives in Section 2.1.1, and goals are typically defined during analysis to document a programme's purpose. Goals can be defined for the programme but also relate to a topic or persona. For instance, a programme could target both new and existing employees. Different personas could reflect each employee type. We can then attach separate goals to each persona.

Goal

As part of the analysis, it is identified which topics are of interest for the professional education programme. These topics can be structured in hierarchies in PEPML. For our running example, "Agile" is one of the topics that shall be covered and "Scrum" is a relevant subtopic. Decomposing topics into subtopics can help to define more fine-grained objectives. The property `inDepthCoverage` allows denoting which topics shall be covered in detail.

Topic

PEPML uses the class `Educator` to represent any educator in a professional education programme. Educator is a generic term, and further subtypes could be identified, e.g. subject matter expert, coach, instructor or facilitator. However, these subtypes are roles that educators take in specific contexts. We explain how this can be specified in the next subsection.

Educator

4.2.4 Design Package

The *Design* package depicted in Figure 4.9 introduces another core class of PEPML, the class `EducationComponent` (cf. ER7). This class is used to represent any learning/teaching format in an education programme. A broad range of learning/teaching formats exists, such as instructor-led sessions, project-based learning or assessments. We define subclasses for a few formats, and subclasses representing further ones can be added as needed.

EducationComponent

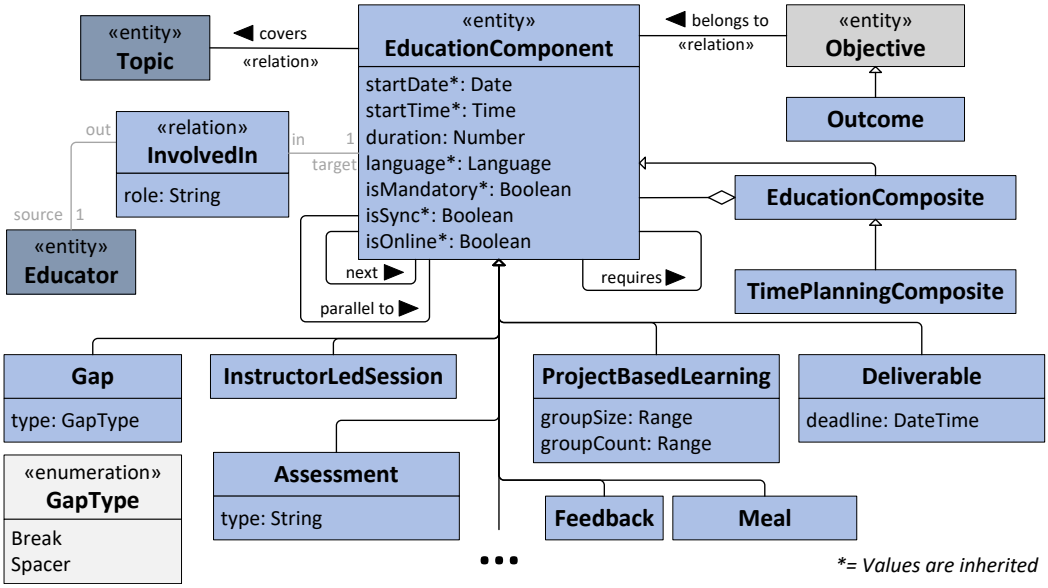

Figure 4.9: *Design* package of PEPML

The class properties of `EducationComponent` specify a few basic characteristics. The property `duration` specifies a component's duration in minutes. Anything below a minute is too fine-grained for PEPML. We decided against using the ISO 8601 format to specify duration since such values cannot be compared using regular mathematical operators. The property `language` can be used to specify the delivery language of the component. In a monolingual programme, this might not be directly important. However, when PEPML is used to create a portfolio based on existing programmes (see Chapter 9), the delivery language can differ between programmes. The delivery mode is determined by the properties `isOnline` and `isSync` (cf. ER11). The former denotes if the component is provided online or not, e.g. an online or a face-to-face session. The latter determines the allowed temporal flexibility. Furthermore, it can be declared if a component is mandatory or not. For composites, the values of properties marked with an asterisks are inherited to sub-components (see Section 4.3.1).

Properties of EducationComponent

Educator Involvement

For an education component, we can specify which educators are involved. In this case, we explicitly list the subclass of `Relation` to also denote a property that enables the definition of an educator's role in the context of this component. Additionally, topics that are covered and objectives can be specified for an education component.

Component Hierarchy

Education components can be aggregated in composites. Composite is the language-internal name for a component that contains other components. In the scope of an actual programme, composites may be referred to as courses, modules, units or subjects [UNE11].

Temporal Order

The temporal order of components is specified using the `next` and `parallel to` associations. Figure 4.10 illustrates the component hierarchy and temporal order of a possible programme design for the running example in the abstract syntax[1]. The `next` and `parallel to` associations define the temporal order for components within the same composite. Two components, C_A and C_B, contained by the same composite have a sequential order if there is a path of `next` links from C_A to C_B. For instance, the top-level objects `dp`, `c1` and `d3` are all in sequential order. If there is no path of `next` links from C_A to C_B, then those components are parallel. In Figure 4.10, example pairs of parallel components are $(c2, c3)$, $(e1, e2)$ and $(a1, a2)$. We explicitly denote this parallelism using the `parallel to` association to define a visual order, which we explain in further detail in Section 5.4.

requires

The `requires` association allows the definition of prerequisites of components (cf. ER10). If there is a `requires` link from a component C_A to C_B, then C_B is a prerequisite of C_A and must occur earlier in the programme. In the case both belong to the same composite, this means that there must be a path of `next` links from C_B to C_A. Figure 4.10 illustrates this with the objects `s4` and `p`, which have a transitive path of `next` links via object `m2`. The `requires` association can be used between components of different composites, e.g. the objects `a1` and `a2` in Figure 4.10 require the objects `e1` and `e2`, respectively.

TimePlanningComposite

The time-planning composite allows the structuring of components into days. If a time-planning composite has a duration of more than one day, it must contain separate time-planning composites for each day. In Figure 4.10, the time-planning composite `dp` has a duration of 2 days and contains the two time-planning composites, `d1` and `d2`, to specify each day. Selected property values in component hierarchies are inherited (see Section 4.3.1), e.g. `d1` and `d2` inherit the `startTime` from `dp`.

[1] Please note that we provided readable values for the `duration` properties instead of giving the internal representation (minutes).

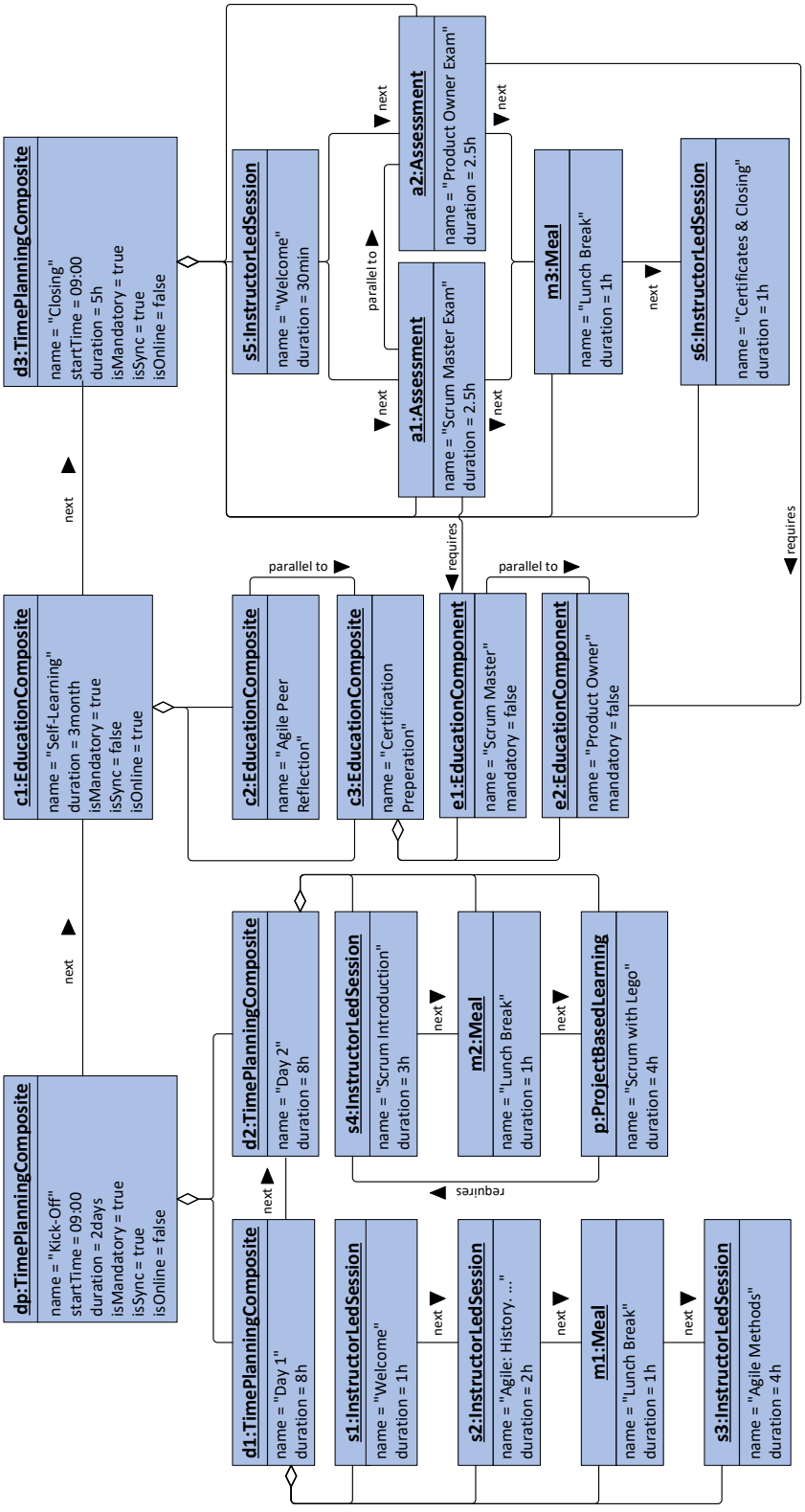

Figure 4.10: Example of a component hierarchy and temporal order in the abstract syntax

4.2.5 Development Package

The *Development* package depicted in Figure 4.11 allows associating education components with resources and locations (cf. ER12 and ER13). Resources can be physical or digital. Some resources may be developed digitally and become physical resources later. For instance, a handout is often prepared digitally, printed at some point and given to participants. Also, we differentiate between physical and digital locations. A digital location requires software like a video conference application.

Resource, Location

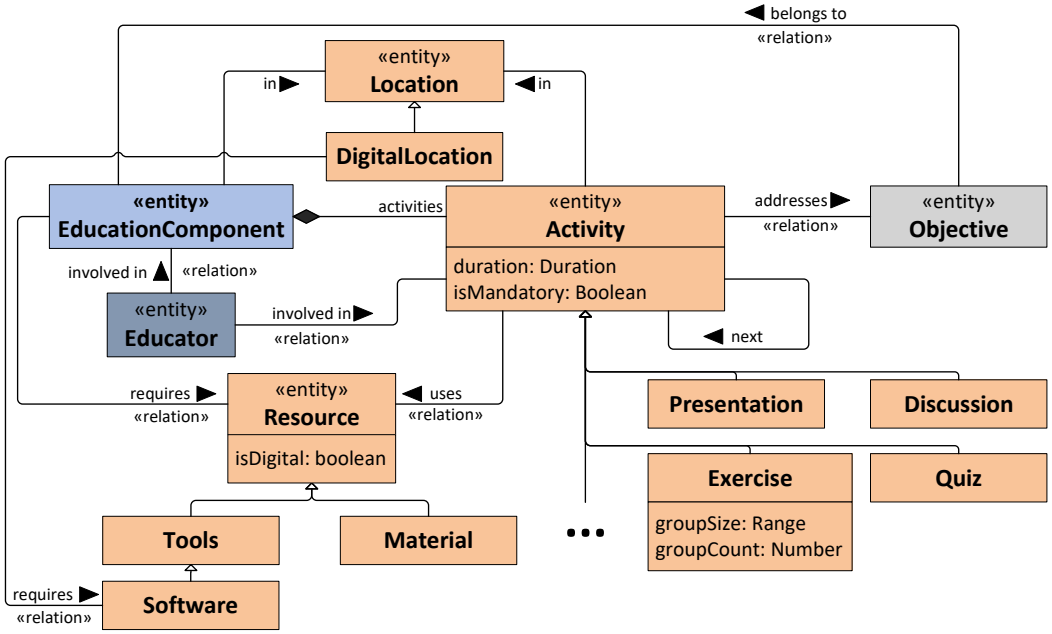

Figure 4.11: *Development* package of PEPML

Earlier Use Of Entity Types

During the analysis phase of an education programme, possible locations or resources may already be gathered. During development, it is defined which resources are being used. We illustrated in Section 4.2.2 how statuses can distinguish between possible and selected options.

Activity

Education components can be refined by specifying the activities carried out as part of this component. The class `Activity` is similar to `EducationComponent` but addresses more fine-grained planning of teaching and learning (cf. ER8). While an education component defines the required resources, the activities specify when they are used. Also, the activities address objectives belonging to the component. This information can be used to determine if all objectives are addressed and if all resources are actually required. The involvement of educators can also be broken down into their involvement in individual activities.

Requirement LR1 states that PEPML shall focus on the high-level perspective. The *Development* package provides the highest degree of detail within PEPML since it allows the decomposition of components into activities. However, it merely states what activities happen and which tools/resources are required. This information is essential to deliver a programme. It does not include how these activities are performed but can refer to tools and documents that provide those details. For off-the-shelf content, like courses on an e-learning platform, it may not even be necessary to refine a component into activities.

Abstraction Level

4.2.6 Implementation Package

The *Implementation* package depicted in Figure 4.12 allows to describe information about running programmes. It introduces `Participant`, a subclass of `Person`, to represent the participants of a programme. It is possible to capture if participants attended a particular education component. Of course, attendance may only make sense for some types of education components, e.g. a face-to-face session. Participants can be part of groups.

Participant

As part of the project-based learning component type, groups or individual participants work on projects. This can be the same project for everybody, or different groups can work on different projects. Furthermore, individual participants and groups can hand in submissions or participate in exams.

Submission, Project, AssessmentResult

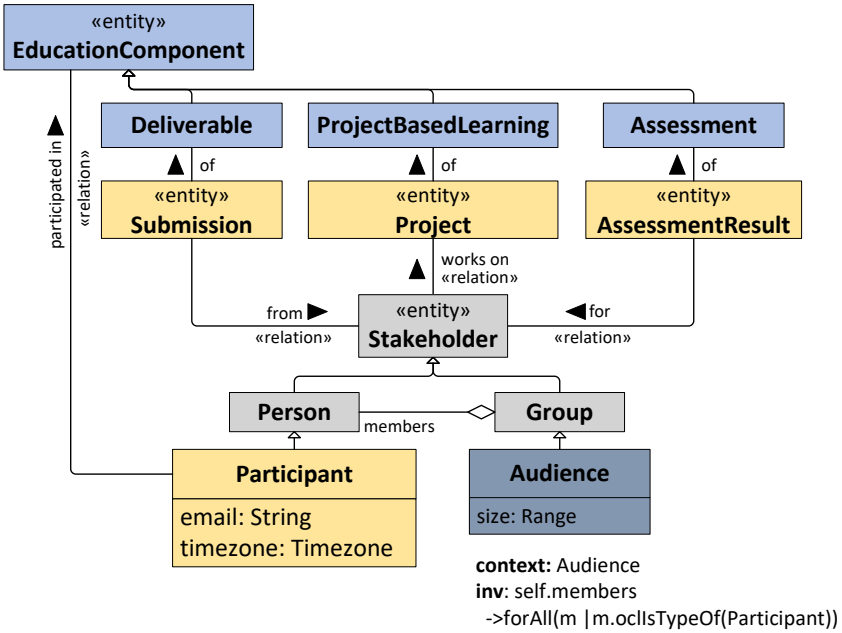

Figure 4.12: *Implementation* package of PEPML

4.2.7 Evaluation Package

The *Evaluation* package depicted in Figure 4.13 introduces the classes `EvaluationStrategy` and `EvaluationResult` (cf. ER14), with the latter being the result of the former. Evaluation strategies can be very different and be performed at different points in time. As mentioned in Section 2.1.2, the Kirkpatrick Evaluation Model [Kir98] defines four levels of evaluation: (1) reactions, (2) learning, (3) behaviour and (4) results. Which strategies are used at which level and even which levels are evaluated depends on the programme. PEPML allows the representation of evaluation strategies in PEPML models to document them or point to their documentation. Performing the actual evaluation is beyond the scope of PEPML and can be supported by tools, e.g. LMSs.

PROGRAMME EVALUATION

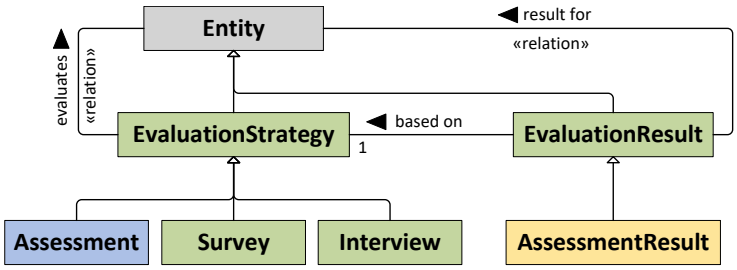

Figure 4.13: *Evaluation* package of PEPML

An evaluation strategy can be attached to any entity of a programme, e.g. to learning outcomes to determine if these have been reached. The evaluation of learning outcomes refers to level (2) of the Kirkpatrick Evaluation Model. Assessments are a particular type of evaluation strategy that can be used to evaluate if learning outcomes are reached. Assessments are a type of education component introduced in Section 4.2.4.

EVALUATIONSTRATEGY

The class `EvaluationResult` is not meant to store full evaluation results but rather as a pointer to the results. For instance, it may contain a link to a document containing the results. This is necessary since the structure of evaluation results is far too diverse to prescribe in a metamodel.

EVALUATIONRESULT

4.3 Additional Semantics

The semantics of the packages and classes of PEPML have been explained in the previous section. This section adds additional semantics for hierarchies and defines constraints PEPML models must fulfil.

4.3.1 Hierarchies

The PEPML metamodel contains various hierarchies: (1) Composites and components, (2) components and activities, (3) topics and sub-topics, and (4) objectives and sub-objectives. Such hierarchies can be used to inherit property values to sub-elements. Currently, we only define property value inheritance for the properties `startTime`, `startDate`, `language`, `isSync` and `isOnline` of `EducationComponent` (see Figure 4.9). This inheritance is why it is sufficient in Figure 4.10 to define values for properties `isSync` and `isOnline` of the top-level composites. Inheritance of property values for other hierarchies can be added when relevant.

Property Value Inheritance

For all hierarchies, categories assigned to a super element are inherited by sub-elements. This means that if a composite belongs to a category, all subcomponents of that composite belong to this category as well, unless the subcomponents specifically define that they belong to other categories.

Inheritance of Categories

4.3.2 Constraints

The UML has limited capabilities to express static semantics. In addition to the constraints imposed by the class diagrams depicting the metamodel, we define the following constraints:

- No circular dependencies in hierarchies.
- No transitive containment of components. Consequently, if a component C_A contains a component C_B, there shall not be a component C_X, which is contained by C_A and contains C_B.
- No transitive `next` links. Meaning if a component C_A is followed by a component C_B, there shall not be a component C_X which follows C_A and is followed by C_B.
- A `next` or `parallel to` link is only allowed between components belonging to the same composite or between top-level components that do not belong to any composite.
- Subgraphs based on `next` links are acyclic. Otherwise, it is not possible to determine a sequential order.
- No reflexive links, e.g. a component cannot require itself.

- If the duration of a composite is set, there should not be a path of sequential components having an overall duration exceeding the composite's duration.
- Activities can only be defined for components and not composites.
- Resources used or objectives addressed by an activity must belong to the education component containing the activity or to one of its ancestors.
- The failure status of a constraint must be different from the status to which the constraint applies. If both point to the same status and the constraint is violated, it would result in an infinite loop.
- The name of a property has to be unique concerning the entity to which it belongs, and it has to be different to any of the entity's property names.
- The semantics of an instance of `Relation`, which is not an instance of a more specific subclass, must be specified using a label. Without a label, the semantics would be too generic and not necessarily convey the intended message.

4.3.3 Extensibility of the Metamodel

The *Foundation* package of PEPML allows the description of an education programme based on entities and relations. The additional packages provide more specific subtypes with a concrete semantical meaning. Furthermore, we provide the possibility of extending PEPML.

PEPML Extensions

While the semantics of the classes and relations defined by the PEPML packages are fixed, we allow adding additional subclasses to refine the entities further. For instance, subclasses of `EducationComponent` can be added to reflect more learning/teaching formats or new subclasses of `Material` can narrow the type of material that is modelled. Similarly, new relation types can be defined by new subclasses of `Relation`. Any addition to the PEPML metamodel must be defined as a package with a clear intent and a description of the semantics of the new entity and relation types.

Custom Properties

In addition to adding subclasses, arbitrary properties can be added for any PEPML entity. This extension capability is required since the current properties only cover basic scenarios, and education providers may want to add additional information, e.g. to reflect further situational factors.

4.4 Summary

In this chapter, we presented the metamodel of PEPML. Thereby, we address Requirement LR5, which states the need for a metamodel to enable programmatic access. PEPML's metamodel is structured into seven packages, which were explained in detail throughout this chapter. The first package provides a foundation for all other packages and enables a basic description of education programmes in terms of entities, relations, stakeholders and objectives. A second package describes tasks and allows attaching them to entities of professional education programmes (cf. LR2). Furthermore, we explained how task constraints can be defined, which can help ensure that changes resulting from a completed task are reflected in a PEPML model. The five remaining packages contain classes and relations that relate to the life cycle phases defined by ADDIE. However, the package structure is introduced for comprehensibility and to ensure that all life cycle phases of professional education programmes are covered (cf. LR3). The language is not limited to specific ADDIE-based Instructional Design processes. The metamodel is extensible (cf. LR7) and allows education providers to add more fine-grained entity types and relations if needed. The concepts introduced for each phase are on a conceptual level and are not bound to a specific process or tooling (cf. LR1). Education components can be hierarchically structured, which enables the modelling of large-scale programmes (cf. LR6).

We referred to the language and expressiveness requirements whenever they were addressed by a concept that had been introduced. While this chapter shows that all expressiveness requirements are addressed, it does not mention all language requirements. Language Requirements LR4 and LR8 are revisited at the end of the next chapter after introducing a whiteboard-compatible visual syntax for PEPML.

Chapter 5

Visual Modelling in Online Whiteboards

The previous chapter covered the metamodel of our modelling language for professional education programmes. To allow collaborative design in whiteboards, we specified the need for a whiteboard-compatible syntax for PEPML (cf. LR8). This chapter analyses the landscape of online whiteboard solutions and their (technical) capabilities and introduces a whiteboard-compatible syntax for PEPML. Specifically, we focus on compatibility with the online whiteboard Miro. This chapter addresses Research Question RQ2, which tries to determine how collaborative online whiteboards can be used as primary modelling tools. With this chapter, we conclude the *Modelling Language Design* step of the language engineering process used to develop PEPML.

Section 5.1 explains our decision to explore the potential of online whiteboards as primary modelling tools. Section 5.2 compares several online whiteboard solutions and justifies why Miro is the best choice for our purposes. Section 5.3 provides details on Miro and assesses its capabilities for visual modelling. Section 5.4 presents a visual syntax for PEPML that can be used within Miro for modelling professional education programmes.

5.1 Rationale for Choosing Online Whiteboards

While collaborative online whiteboards existed prior to the pandemic, their usage significantly increased as a result of the remote work due to the lockdowns. They are used for teaching purposes, collaborative design, and to facilitate online meetings.

During the pandemic, we started using online whiteboards ourselves for teaching and facilitation purposes and gained substantial experience in using these tools. After an initial analysis of available tools, we decided to use Miro due to its feature-richness, ease of use and popularity. G2 lists Miro as the market leader for both visual collaboration[1] and online whiteboards[2]. Our experience showed us that students and professionals can quickly be productive with Miro after just a few minutes of introduction.

In contrast to online whiteboards, dedicated modelling tools restrain users in what they can and cannot model. If users are not entirely familiar with the modelling language and the tool itself, they cannot immediately start to work. Moreover, most modelling tools are not intended for real-time collaboration but instead focus on asynchronous collaboration through versioning systems. Collaboration with clients to design programmes is often time-bound, and clients are not necessarily familiar with domain-specific languages or dedicated modelling tools. Therefore, we decided to explore as part of RQ2 if collaborative online whiteboards can be used as primary modelling tools. Also, we have successfully used an early version of PEPML in an informal way to remotely design a programme with clients, which influenced our decision to explore the potential of online whiteboards.

5.2 Comparison of Online Whiteboard Solutions

Our positive experience with using Miro boards and the fact that it is the market leader in this segment initially influenced the selection of Miro as a basis for our tool support. Nonetheless, it is by far not the only online whiteboard solution. Consequently, we performed an analysis to determine which online whiteboard solution is the best candidate to explore if it can be extended to a primary modelling tool. In the following, we present a comparison of online whiteboard solutions showing that Miro is the best basis for visual modelling tool support based on online whiteboards.

5.2.1 Identifying Relevant Online Whiteboard Solutions

For our comparison, we considered all solutions from the G2 grid for online whiteboards. We added the whiteboards listed in the work of Ahmmad et al. [AMAC21], which compared online whiteboards for their fit in the higher

[1] https://g2.com/categories/visual-collaboration-platforms
[2] https://g2.com/categories/collaborative-whiteboard

education context. While Ahmmad et al. [AMAC21] identified Miro as the best whiteboard based on their requirements, our context differs. Therefore, we only considered the whiteboard solutions they listed but not their comparison results. Additionally, we performed a Google web search for "online whiteboard" and analysed the first five pages of results. Among these results were direct links to whiteboard solutions as well as various blog posts ranking such solutions. We included whiteboard solutions that were directly listed in the results, as well as those solutions mentioned by rankings.

Our initial list contained 53 tools, of which 31 are listed on the G2 grid for collaborative online whiteboards. We performed an initial scan of these tools and excluded 33 tools that are mainly for (freehand) drawing or have a different focus, e.g. video conferencing. The complete list of identified tools and reasons for excluding 33 of them is provided in Appendix B.

5.2.2 Criteria for Comparison

Our goal is to use online whiteboards as primary modelling tools with the capability for syntax validation and extraction of machine-readable models. We defined nine comparison criteria (CC) to identify the best whiteboard solution for our purposes. These nine criteria can be split into four groups: extension capabilities (CC1, CC2), developer support (CC3, CC4, CC5), visual modelling support (CC6, CC7, CC8) and anonymous collaboration support (CC9). The following briefly explains each criterion:

CC1 **API/SDK**: *Does the whiteboard provide an API or SDK to access and manipulate board data?* An API/SDK is necessary to extract the modelled content and derive a formal model. Manipulation of the board is necessary to import models and to correct errors automatically.

CC2 **Extensible UI**: *Does the whiteboard support extensions to its UI?* A UI extension mechanism is needed to provide users the option to access custom features like model validation.

CC3 **Developer Documentation**: *Is the extension mechanism openly documented?* Without open documentation, it is difficult to use an extension mechanism.

CC4 **API/SDK Examples**: *Do open-source examples exist that illustrate the use of the extension mechanism?* Examples of extensions can be a good starting point and reference to see certain features in use.

CC5 **Developer Community**: *Does an active developer community exist for the whiteboard solution?* An active developer community is helpful to ask questions about the extension mechanism and to seek help when facing problems.

CC6 **Graph Support**: *Does the whiteboard support connecting objects to form graphs?* Most visual modelling languages rely on graphs consisting of nodes and relationships to depict models.

CC7 **Shapes**: *Does the whiteboard support the usage of geometric shapes?* The visual syntax of most major modelling languages like UML or BPMN heavily relies on geometric shapes. Hence, support of geometric shapes is crucial for visual modelling.

CC8 **Sticky Notes**: *Does the whiteboard have an explicit representation of sticky notes?* Using sticky notes is a typical way of gathering ideas on whiteboards.

CC9 **Guest access**: *Does the whiteboard allow anonymous guest access?* Forcing clients to have an account for a specific tool is not always possible. Therefore, anonymous guest access is an essential feature for collaboration with external clients.

5.2.3 Comparison

The results of our comparison are shown in Table 5.1, and we explain the reasoning behind each of these criteria and discuss the results of our comparison in the upcoming paragraphs.

If a whiteboard shall be used as a primary modelling tool, it is necessary to have an API to interact with a board and extract the depicted models. This can be a JavaScript library like Miro's WebSDK, an HTTP API or any other form of API. In our comparison, only nine whiteboard solutions provide any form of API. The API of Whiteboard Team and Sketchboard is limited to board creations. Collaboard mentions having an API but does not provide any public information about it. Bluescape Workspace, FigJam, Lucidspark, Miro, Mural and Stormboard provide an HTTP API allowing CRUD operations on board content. Additionally, Miro, Lucidspark and FigJam provide a JavaScript library to interact with a board. Bluescape Workspace also provides a GraphQL-based API in addition to its HTTP API.

API/SDK (CC1)

Whiteboard	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
Canva Whiteboard	no	no	n/a	n/a	n/a	yes	yes	yes	yes
Conceptboard	no	no	n/a	n/a	n/a	yes	yes	yes	yes
draft.io	no	no	n/a	n/a	n/a	yes	yes	yes	yes
Google Jamboard	no	no	n/a	n/a	n/a	no	(yes)	yes	yes
InVision Freehand	no	no	n/a	n/a	n/a	yes	yes	yes	yes
Microsoft Whiteboard	no	no	n/a	n/a	n/a	no	yes	yes	no
Milanote	no	no	n/a	n/a	n/a	yes	no	yes	yes
Productivity Lab Whiteboard	no	no	n/a	n/a	n/a	no	yes	yes	yes
Klaxoon Board	no	no	n/a	n/a	n/a	yes	yes	yes	yes
WebEx Whiteboard	no	no	n/a	n/a	n/a	no	yes	yes	no
Weje	no	no	n/a	n/a	n/a	yes	no	yes	yes
Collaboard	(yes)	no	no	no	no	yes	yes	yes	yes
Sketchboard	(yes)	no	no	yes	no	yes	yes	yes	yes
Whiteboard Team	(yes)	no	yes	no	no	no	(yes)	yes	yes
Stormboard	HTTP	no	yes	no	no	yes	(no)	yes	yes
Mural	HTTP	no	yes	yes	yes	yes	yes	yes	yes
Bluescape Workspace	HTTP+GraphQL	no	yes	yes	yes	yes	yes	yes	yes
FigJam	HTTP+JS	yes	yes	yes	yes	yes	yes	yes	yes
Lucidspark	HTTP+JS	yes	yes	yes	yes	yes	yes	yes	yes
Miro	HTTP+JS	yes	yes	yes	yes	yes	yes	yes	yes

Table 5.1: Comparison of online whiteboard solutions

UI Extension (CC2)

The ability to extend a whiteboard's UI is needed to offer users additional features for model management, e.g. trigger model extraction or creation of model-specific templates. Miro's WebSDK and Lucidspark's Extension API allow to add menu items and dialogs to the respective tools and edit board content. FigJam provides a Widget and Plugin API to interact and extend a board's functionality. The Widget API allows the development of smart board items, like countdown clocks, that can be added to a board and are accessible by any user. FigJam's Plugins can create board content based on user input but are only accessible to individual users.

Developer Support (CC3, CC4, CC5)

FigJam, Lucidspark and Miro provide comprehensive documentation of their APIs and have a community forum for help. FigJam and Miro provide several examples of extensions and API usage on GitHub. For Lucidspark, the examples are limited to those in the developer documentation. In contrast to the other two, Miro maintains a public roadmap about upcoming features of its WebSDK/API. Also, Miro is the only whiteboard that has a marketplace for extensions and a CSS library for proper styling of UI extensions[3].

Visual Modelling (CC6, CC7, CC8)

Shapes and sticky notes are important to organize board content and for visual modelling purposes. Geometric shapes are supported by nearly all whiteboards in our comparison. Only Stormboard does not support shapes directly but allows different shapes of sticky notes. While Google Jamboard and Whiteboard Team support shapes, they do not allow adding text directly to them. Instead, a separate text field must be placed on top of the shape. Most whiteboards in our comparison support the creation of graphs by connecting items. Five out of 20 compared whiteboards only support drawing lines that are detached from other board items.

Anonymous Collaboration (CC9)

Guest access is crucial to start a collaboration with clients quickly. In our comparison, only WebEx Whiteboard and Microsoft Whiteboard did not provide anonymous guest access. All other tools allowed the creation of a link to provide read/write access without requiring any account.

In summary, only FigJam, Lucidspark and Miro satisfy all of our criteria. Lucidspark's extension API is very powerful but did not exist at the time when we had to choose a whiteboard solution. FigJam plugins do not extend the UI for all users, and the widgets are meant to create smart items but not to extend the general board functionality. Overall, Miro is the market leader for this segment and provides a well-designed, well-documented, mature and actively developed extension mechanism. Therefore, we chose Miro as the online whiteboard solution for this thesis.

[3] https://mirotone.xyz

5.2.4 Observations

We have made several observations while comparing whiteboard solutions, which we briefly discuss in the following. The whiteboard solutions, which we excluded since their focus mainly on freehand drawing, were often focused on teaching scenarios or were significantly older than the other approaches. Possible reasons are advances in web technologies and the higher demand for such solutions due to increased remote work/collaboration.

Some whiteboard solutions focus on specific domains. For instance, we discovered multiple approaches focusing on UI/UX or product design, like Aha! or FlowMap. Several solutions also focussed on idea management, e.g. Stormboard or BrightIdea, or extend their feature set towards project management, e.g. Miro or FigJam. In fact, we excluded some tools since their focus was predominantly on project management without a collaborative canvas for interactions.

Working with shapes, sticky notes, connectors and freehand annotations is not necessarily the main focus of all whiteboard solutions. For instance, DEON uses a whiteboard to work on multiple documents embedded in the canvas collaboratively. In contrast, Stormboard only supports the usage of sticky notes, which allows them to generate reports based on the used board template. This illustrates the potential interplay of documents and collaborative whiteboards. Such an interplay is necessary since the content on online whiteboards does not replace but complements other documents, something we acknowledge within our approach to model professional education programmes.

5.3 Online Whiteboard Miro

For the remainder of this thesis, we focus on Miro as an online whiteboard solution. A basic understanding of Miro's features and terminology is needed to understand the Miro-based tooling, which we present in Chapter 6. This section solely focuses on the crucial elements a reader needs to know. Some technical aspects of Miro boards are required, which a typical Miro user would not need to know, to later understand how modelling tool support can be built based on Miro. For further explanations on Miro usage or the implementation of Miro apps, we refer to the user and developer guide[4].

[4] https://developers.miro.com

5.3.1 Boards and Items

Figure 5.1: Simplified overview of relevant concepts on Miro boards depicted as a UML class diagram

Figure 5.1 provides a simplified overview of the Miro concepts in form of a metamodel. This diagram is deduced from using Miro and its API specification. It is neither intended to be complete nor accurately represent the actual implementation. Properties or classes mentioned in the API specification are omitted when not (strictly) needed. Furthermore, classes and properties are partially renamed compared to the API specification and inner classes are added to simplify understanding. Indications are given throughout the following explanations of what parts are omitted or altered.

Miro Metamodel

A Miro board is an infinite 2D canvas on which items can be placed. The term "Item" refers to any element on a Miro board. Both boards and items have an identifier (`id`). A board is accessible via a unique URL, and deep links can be created pointing to specific board items. On the highest level, we can distinguish between two types of items: geometric items and connectors.

Items

Geometric items have a `width`, `height` and position (`x`,`y`) on a board. In most 2D frameworks like the HTML5 Canvas[5], the (x, y) coordinates of geometric items refer to the top-left corner. However, on Miro boards, these coordinates refer to an item's centre. Floating-point numbers are used to denote the position and the dimension of an item. Since a Miro board is

Geometric Items

[5] https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

infinite, it has a centre, which has the position $(0,0)$, but the board expands in all directions. Hence, the (x,y) coordinates can be negative. If items overlap, the `z-index` declares which items are shown on top of one another. Additionally, a link can be added to geometric items, which results in a small arrow icon in the top right corner of an item that leads to the specified URL. Adding links to geometric items is useful to refer to external resources or another element on a board, which provide additional information.

The Miro developer guide does not use the term geometric items explicitly. It is used in this thesis to refer to the group of items that have these specific properties (cf. with `GeometricItem` in Figure 5.1). These item types are further specified by the subclasses of `GeometricItem`, which are explained in the subsequent paragraphs. The geometric items relevant to this thesis are shapes, sticky notes, text, images, (app) cards and frames. Other geometric items, like embedded objects, are out of the scope of this thesis.

Relevant Shapes

Shapes are geometric forms that can be added to a Miro board. Figure 5.2 shows a selection of basic shapes, mainly those used within this thesis. Basic shapes are available in any version of Miro. The enterprise version of Miro also features specialized shapes for popular modelling languages, such as Flow Diagrams or the BPMN. For the remainder of this thesis, we refer to these specialized shapes as stencil shapes, which is the term used within Miro's API for them. In contrast to basic shapes, stencil shapes can have multiple segments, e.g. a UML class stencil can have a segment for the class name, the properties and the operations. However, the programmatic interaction with stencil shapes is limited and custom stencil shapes cannot be added. The limitations regarding stencil shapes are further discussed in Section 5.3.3.

Basic/Stencil Shapes

Sticky notes are similar to a square shape. In contrast to shapes, Miro offers additional functionality for sticky notes, e.g. adding reactions using emojis or clustering them based colours or tags. This additional functionality is handy when working collaboratively with Miro boards.

Sticky Notes

Text items only consist of the text they represent. They can be seen as rectangular shapes without a border. Their dimensions are directly related to the size of the text, meaning that manipulation of width or height directly affects one another. Consequently, the dimensions of text items cannot be aligned to cover a specific area and such items are removed automatically, when they do not contain any text.

Text Items

Shapes, sticky notes and text can all contain text content, which is why we refer to them as `ContentItems`. Miro itself does not use this name, but it is helpful to commonly address these items in this thesis. The content is a

Figure 5.2: Overview of relevant Miro board items and styling options

string containing text that can be wrapped by a limited set of HTML tags. A typical Miro user does not need to know the explicit HTML tags being used, but since we utilize them in our approach, these technical details are relevant. The content consists of HTML paragraphs. By default, all text is in one paragraph (`<p>`). A new paragraph is added in the case of a hard line break (Enter). In the case of a soft line break (Shift+Enter), only a break line tag (`<br>`) is added to the paragraph. Furthermore, text can be bold (`<strong>`), italic (`<i>`), underlined (`<u>`) or stroked (`<s>`). In contrast to shapes or sticky notes, text items support lists and text indentions.

Formatting of Text

Images can be added to Miro boards via the menu, drag and drop, copy and paste or programmatically. Miro allows the use of bitmap or vector images. Images can have a title, but this title is only visible in the menu when the image is selected.

Images

Miro positions itself more as a collaboration platform than as a digital whiteboard. To become such a platform, Miro allows the integration with other tools. App cards are geometric items that can represent information from external tools. For instance, the developer guide provides an example showing how GitHub issues can be added as app cards to a Miro board[6]. These app cards can contain fields which provide specific information from the external tools. In the case of GitHub issues, a field could represent the person assigned to handle the issue. App cards are connected to external tools, and the represented information is synchronized. The synchronization functionality has to be implemented by the developers of the integration. App cards can be enlarged and show a UI that the Miro app developers can customize. Miro also has an item type called “card”, which shares commonalities with the app card type but has fewer capabilities for integrating information from external tools, i.e. it cannot show a custom UI when expanded.

App Cards and External Tools

[6] https://developers.miro.com/docs/web-sdk-how-tos

Frames are special geometric items that can contain other items. Such frames are used to create dedicated areas within a board. In Miro's presentation mode, users can easily navigate from one frame to another. Frames are also the basis for exporting board content to PDF. Each frame becomes a page in the exported PDF. Within a frame, geometric items are positioned relative to the top-left corner of a frame, which has the (x,y) coordinate $(0,0)$. When a frame is moved, all child items are moved as well. As mentioned, the (x,y) coordinates of an item refer to its centre. When the centre of an item is within the borders of a frame on a Miro board, it becomes a child of this frame. The content of frames can also be hidden, which can be useful if depicted information is not relevant at the given point in time.

Frames

Items can also be aggregated into a group. In contrast to a frame, a group itself is not a geometric item and does not have (x,y) coordinates. When an item of a group is selected, the whole group of items is selected automatically. This group of items can then be moved or scaled jointly. Double-clicking on an item still allows users to access individual items within a group.

Groups

Connectors represent lines on a Miro board that are typically used to connect two items. However, connectors do not necessarily need to be connected to an item and can exist independently. When connected to an item, a connector stays connected to the respective items even when they are being moved. The start and end of a connector can have different shapes, which are called stroke caps. An overview of available stroke caps is shown in Figure 5.2. Multiple captions can be added to connectors. For instance, such captions could denote multiplicities at the start and end of a connector that represents an association in a UML diagram.

Connectors

Items and groups can be locked. A locked geometric item cannot be moved or resized. A locked connector cannot be detached from the items it is connected to, but it moves if the connected items move. If a group is locked, all group items are locked, meaning none of these items can be moved unless the group is unlocked.

Locking Items

Board items can be styled in various ways. Not all styling options are shown in Figure 5.1 to reduce complexity. In the most cases, the background colour of items and the line colour, thickness and type (solid, dashed or dotted) of borders or connectors can be changed. The styling options differ between the item types. For instance, it is impossible to specify a background colour for images, the background colours for sticky notes are limited to 16 distinguishable colours, and text items do not have a border. There are further differences in styling which are not relevant to this thesis.

Item Styling

Templates

Miro provides a large set of templates for different purposes. For instance, it provides templates for popular canvas models like the Business Model Canvas [OP10] or the Value Proposition Canvas [OPBS14]. A template consists of prearranged items that can be added to a board. Once added, these items can be manipulated. Custom templates can be created and shared.

Auto-Save and Versioning

Any manipulation of items on a Miro board is immediately saved. Hence, there is no need and even no option to save changes explicitly. There is no option to versionize board content, either. Hence, aside from a temporary history of actions that can be undone, there is no possibility to revert to a prior version. Miro does offer the possibility of duplicating boards, which can be used to back up a specific version.

Import/Export

The UI only offers the possibility to export a board as a PDF or an image. Boards can be backed up in a proprietary format, and a list of items on a board can be downloaded as a Comma-separated Value (CSV) file. However, there is no option in the UI to export all board information in an open, machine-readable format like JSON or XML. Copy and paste between boards and other applications is supported. Boards can be created by importing Lucidchart, Microsoft Visio or Draw.io files, but boards cannot be exported to these formats. Furthermore, the geometric items imported based on any of these formats are not necessarily compatible with those offered by Miro and can behave differently from standard geometric items, e.g. when resized.

5.3.2 App Development and Integrations

Embedding Miro Boards

Miro allows different forms of extensions and integrations. Other web applications can be added to Miro boards as embedded objects, and Miro boards can be embedded via HTML iframes into other web applications. While embedding is handy for merging UIs of different applications to a certain extent, Miro provides a much more powerful extension and integration mechanism with its WebSDK and REST API.

WebSDK

The WebSDK allows the building of Miro apps that can add features to Miro's UI. A menu item in Miro's toolbar is added for each app to access the app's features. Using Javascript/TypeScript as a programming language, the WebSDK allows developers to manipulate board items and react to certain events, e.g. adding, selecting or removing board items. Miro apps are only active if a board is opened in a browser, meaning no interaction with the board is possible over the WebSDK if no user is currently on the respective board. While the use of the WebSDK limits developers to JavaScript/TypeScript, the

WebSDK does not impose constraints on the JavaScript frameworks being used for the UI extensions. The bootstrapping tool for building Miro apps even allows to build skeletons for Miro apps that using different frontend frameworks. Miro apps can be published to Miro's App Marketplace, which in early 2023 contained nearly 150 apps. Among them are apps to interact with other tools, e.g. for visual design or communication.

In contrast to the WebSDK, the REST API can interact with a board even if the board is not currently opened by anyone. It allows CRUD operations on board items. Webhooks can be set up to inform external applications about changes on a board without requiring a Miro app. While items can be manipulated, the REST API cannot add other features to Miro's UI. Miro's REST API, as REST APIs in general, is programming language agnostic, and thus, not limited to JavaScript/TypeScript.

REST API

The WebSDK and REST API have a significant overlap in functionality regarding CRUD operations on board items. However, they are being developed separately, and features are not released simultaneously for both. For instance, CRUD operations on connectors were first released to the REST API and months later added to the WebSDK. Miro actively extends its WebSDK and REST API and maintains a public roadmap[7]. Developers are encouraged to suggest additional features, and roadmap items can be upvoted to signal interest. Furthermore, an actively maintained forum exists where developers can seek help or suggest features. Overall, Miro's efforts regarding extension and integration capabilities are impressive and unmatched by its competitors, as discussed in Section 5.2.

Active Development

5.3.3 Assessment based on the Physics of Notations

The Physics of Notations theory includes nine principles for visual notation design [Moo09]. These principles are often utilized to build new visual languages or improve existing ones [vdLH19]. The choice of the technological basis for the visual modelling tool support can affect to which degree language designers can follow certain principles. In the following, we assess if Miro allows language designers to follow the principles of the Physics of Notations.

Visual Expressiveness

The principle of *Visual Expressiveness* describes eight visual variables to encode information and encourages using all of them. Miro supports using the

[7] https://developers.miro.com/page/roadmap

planar variables, *horizontal and vertical positions*, to encode information for geometric items, since they have a (x,y) position. Connectors are no geometric items and their position cannot be determined. Hence, we cannot use a connector's path to encode information. Moody [Moo09] does not mention the z-dimension in his paper since he focuses on 2D visual notation. Nonetheless, even 2D languages require a z-index to determine which element is in front when elements overlap. The z-index is only programmatically accessible via Miro's WebSDK, but not via its REST API.

Horizontal/Vertical Placement

The other visual variables are called retinal variables. *Shape* is the only retinal variable with an unlimited capacity since humans can distinguish an infinite amount of different shapes. Using shapes to encode information is supported by either using Miro's basic shapes, images or stencils from specific languages. Unfortunately, the stencils are programmatically not accessible, but support for this is on Miro's roadmap. Programmatic access to custom geometric items imported from other tools, e.g. Visio, is also not possible. Thus, language designers are limited to basic shapes and images.

Shape

Images can be used to add custom symbols to boards. Since images are not content items, no text content can be added directly to an image item. However, it is possible to place a text item above an image item to overlay text. Grouping is required to ensure that image and text stay connected.

Images as Custom Shapes

Composite shapes can be created by arranging multiple basic shapes. For instance, multiple rectangles can be used to depict a table. Nevertheless, such a table remains a collection of rectangles and adding a row means adding a set of rectangles. In contrast, Miro's advanced shape for tables allows the insertion/deletion of rows/columns, but it is not programmatically accessible.

Composite Shapes

The visual variables *size*, *colour* and *orientation* are supported by Miro, but there are differences in their support between the types of geometric items. Shapes support the use of all three variables fully. However, sticky notes cannot be rotated, their width/height ratio is either 1:1 or 2:1, and the available background colours are limited to enable clustering. In the API, it is only possible to set the width for text items, and the height is calculated automatically. A caveat when interacting programmatically with Miro boards is that the WebSDK can only extract a read-only value of the height of a text item. The API does not retrieve a height value for text items at all.

Size, Colour and Orientation

There is limited support for the visual variable *texture*. Miro does support the styling of borders and connectors, e.g. by using a dashed or dotted style instead of a solid line. However, the background texture of items cannot be changed and is always unicoloured.

Texture

Miro does not offer a direct way to manipulate the visual variable *brightness*. The transparency of items can be increased, which has a similar effect to changing the brightness, but if items are stacked, increasing the transparency reveals items in the background. Another possibility is to use RGB values to choose a colour that appears brighter.

Brightness

In summary, Miro supports encoding information using all visual variables, at least to some degree. There is only limited support for using textures, and workarounds are needed for using brightness to distinguish semantics. The remaining visual variables are well supported by Miro, with minor differences depending on the item type used to encode the information.

Semiotic Clarity

The principle *Semiotic Clarity* states that there should be a one-to-one mapping between semantic constructs and visual notation elements. The four types of violations of this principle are introduced in Section 2.2.2.

Symbol deficits or redundancies represent no issues. In the case of a symbol deficit, the respective semantic construct simply cannot be displayed on a Miro board. Symbol redundancy would mean that different items can be used to represent the same construct, which is possible as well.

Symbol Deficit & Redundancy

Symbol excess cannot be prevented on a Miro board since we cannot limit what is being added to a board. Users can freely add any items to a Miro board, independent of whether it belongs to the visual syntax of a language or not. Consequently, it can be difficult for users to distinguish between semantically relevant and irrelevant items. Symbol overload, so the mapping of two different semantic constructs to the same notational element, can be problematic. If it cannot be determined on a syntactical level which semantic constructs are meant, it cannot be ensured that the correct item is extracted. We revisit the issues with symbol excess and overload in Chapter 6.

Symbol Excess & Overload

Dual Coding

The principle *Dual Coding* suggests using text to complement graphics. All of Miro's basic shapes support text content, and text items can be placed freely on the board. If text items are used for dual coding, it is advisable to group these text items with the items they complement. Otherwise, the movement of the graphic does not affect the text item and vice versa. Furthermore, multiple captions can be added to a connector, e.g. at the beginning, middle or/and end. Hence, language designers can follow the dual coding principle.

Dual Coding

Manageable Complexity and Cognitive Integration

Complexity Management

The principle *Complexity Management* states that a language shall provide means to reduce complexity, e.g. modularization or hierarchies. Miro only provides very limited features that can be used for complexity management. The details of (app) cards can be reduced, content can be organized into frames, and frame content can be hidden. Custom collapsible items cannot be defined, but a link can be defined for a geometric item to refer to additional information about that item.

Cognitive Integration

Closely related to complexity management is the principle of *Cognitive Integration*. It defines the necessity of a mechanism that supports the integration of information from different diagrams. Since Miro's canvas is infinite, multiple diagrams can be on one board. In that case, connectors can be used to refer to information in other diagrams. Similarly, the link that can be set for a geometric item can be used to refer to other diagrams, even if this diagram is not located on the same board.

Remaining principles

Graphic Economy

Miro partially influences the principle of *Graphic Economy* since we cannot forbid symbol excess. Consequently, even if a language only uses a small set of graphical items, users can always add any type of item supported by Miro.

Perceptual Discriminability

The principle *Perceptual Discriminability* is limited by the number of different shapes supported by Miro. While there are plenty of different shapes, not all are currently accessible via the API. However, the number of item types supported by the API/SDK increases steadily, and in the future, language designers can base their visual syntaxes on a wider variety of shapes.

The remaining principles, Cognitive Fit and Semantic Transparency, are not influenced by the technological base of the tooling. It is up to the language designer to decide to which degree these principles should be followed.

5.4 Miro-compatible Visual Syntax for PEPML

After explaining PEPML's metamodel in Chapter 4 and explaining Miro's capabilities for visual modelling, we present in this section a Miro-compatible visual syntax for PEPML. First, we give an overview of the elements of the visual syntax. Then, we introduce two viewpoints for PEPML models. We use the running example to illustrate the use of both viewpoints. Chapter 6 explains how user-specific visual syntaxes, e.g. canvas models, can be used.

5.4.1 Overview of Elements and Semantics

In the following, we address Language Requirement LR8 by introducing a Miro-compatible visual syntax for PEPML. To this end, we first provide an overview of the elements of the visual syntax of PEPML. In addition to the visual representations, we explain how information can be accessed which is not represented visually and how decorators allow providing additional information about entities.

Visual Representations

Figure 5.3 shows the standard visual representations of PEPML elements. We use shapes to represent topics, objectives, comments, tasks, education components and activities, e.g. rectangles represent education components. The border style indicates if an element is mandatory or optional. All other PEPML entities are represented via icons. These icons are the default suggestions. Language users can use their own icons, e.g. to increase semantic transparency [Moo09]. More information on custom icons is given in Chapter 6.

Shapes and Icons

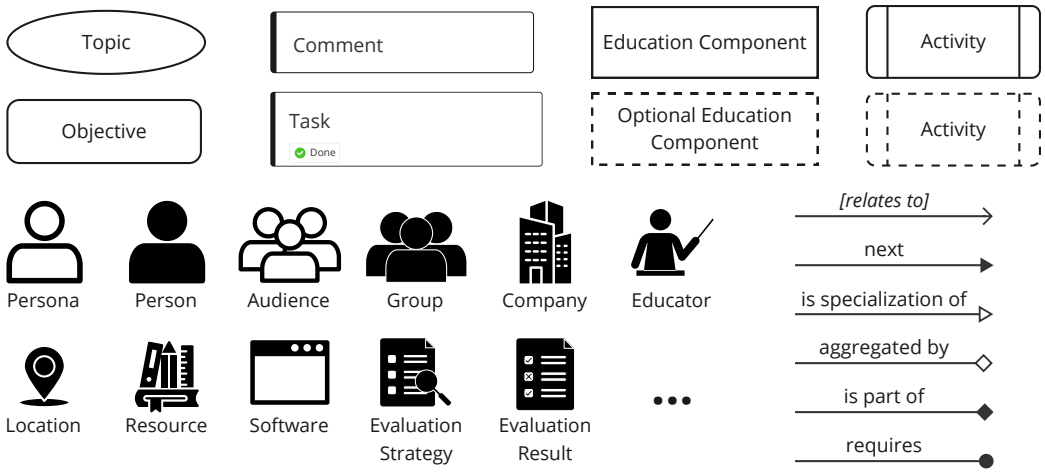

Figure 5.3: Visual elements used in PEPML

Stroke caps are used to distinguish between different kinds of relations. For the generic "relates to", a label is required to denote the concrete semantics. The other relation types are more specific and labels for them are optional. We intentionally used the same visual representation as the UML for aggregations, compositions and specializations to simplify the understanding for people already familiar with the UML.

Visualizing Entity Relations

Positioning of education components encodes information about containment and temporal relations. As shown in Figure 5.4, a sequential ordering

Temporal Order via Horizontal and Vertical Placement

is expressed by placing elements horizontally next to each other. Parallel components are vertical neighbours. Placing a component inside another component denotes containment. A component containing another component is called a composite.

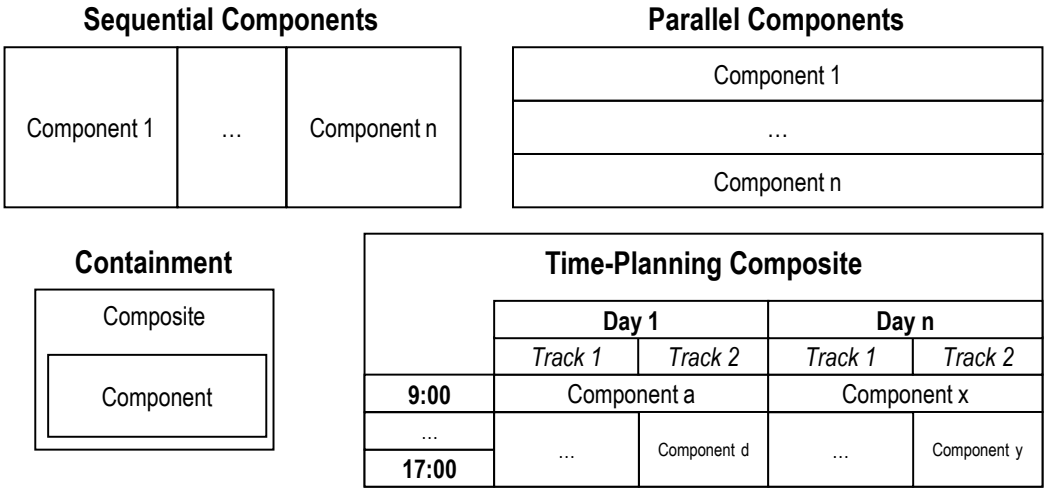

Figure 5.4: Depicting containment and temporal relations

Duration Expressed via Item's Height

The time-planning composite is for more narrow temporal planning. It represents one or multiple days. A day may have multiple tracks. A time scale on the side indicates start times and durations of components.

Usage of Visual Variables

We explained in Section 5.3.3 that Miro supports the use of nearly all visual variables to encode information. Table 5.2 summarizes the visual variables used by PEPML and the information they encode. The shape of an element is the main differentiator between the entities that can be represented by PEPML. The size of an item is only utilized in a limited fashion, i.e. only to denote durations in time-planning composites. Outside of time-planning composites, we decided against using an item's size to denote any information because an item's size is typically influenced by the text it contains. So, we prefer readable content over an indication of information via an item's size.

Using other Online Whiteboard Solutions

The representation of comments and tasks is Miro-specific. Furthermore, the geometric shape to denote an activity or all types of stroke caps may not be available in every whiteboard solution. If PEPML shall be used in a whiteboard solution other than Miro, we suggest denoting tasks and comments as sticky notes and use labels to compensate for missing types of stroke caps.

Table 5.2: Semantics of visual variables

Visual Variable	Semantics
Shape	Distinction between entity types
Brightness, Colour	Grouping entities into categories
Vertical/Horizontal Placement	Temporal relation and containment of components*
Size	Height represents the duration in time-planning composites*
Texture	Border texture encodes if a component is mandatory or optional
Orientation	No semantic meaning

* = only relevant in temporal viewpoint

Defining Properties of Entities

In PEPML's visual syntax, the values of some class properties are represented visually. For instance, we use a different border style to denote if a component is optional or not. Also, start times and durations of components contained in time-planning composites can be inferred based on the provided time scale.

Each entity has a name and a description. Typically, the first line of the text within a shape is used as the entity's name and the remaining text as the entity's description. The text content of an education component may also refer to a date, timeframe, duration or involved educators. We explain in Section 6.2.3 how this information can be identified and extracted.

Entity Name and Description

It is common that not every aspect of a modelling language is represented visually. For instance, BPMN does not have a visual representation for extensions or all properties, e.g. the number of activation tokens is not represented visually. The remaining properties of PEPML's metamodel and the user-defined properties are not visually represented. They have to be specified separately. In our tooling, we provide a properties tab for this (see Section 6.2.1). However, we support customizing the visual syntax and visually indicating property values using decorators.

Remaining Properties

Decorators

Icons in the top-left or top-right corner of geometric shapes provide further details about an entity. We call these icons "decorators", and they can indicate a specific subtype of an entity or particular property values.

Standard Decorators

Table 5.3 shows PEPML's standard set of decorators. This set includes decorators to distinguish that an objective is a goal or an outcome. Decorators exist to show the delivery mode, e.g. synchronous face-to-face or asynchronous online delivery. Moreover, we use a circled number in the top-right corner to indicate how many open tasks are associated with an entity.

Table 5.3: PEPML's default entity decorators

Icon	Name	Application Condition
	Goal	Instance of `Goal`
	Outcome	Instance of `Outcome`
	Face-to-Face Session	Instance of `EducationComponent` and `isOnline` = `false` and `isSync` = `true`
	Online Session	Instance of `EducationComponent` and `isOnline` = `true` and `isSync` = `true`
	Self-Study	Instance of `EducationComponent` and `isOnline` = `false` and `isSync` = `false`
	Online Self-Study	Instance of `EducationComponent` and `isOnline` = `true` and `isSync` = `false`
	Assessment	Instance of `Assessment`
	Open Tasks	Instance of `Entity` having $N > 0$ open tasks

Custom Decorators

Our tooling supports the addition of custom decorators by specifying the conditions an entity must fulfil for the decorator to apply. Especially for custom decorators, a legend is required to explain the semantics. We illustrate the use of decorators in the next section.

5.4.2 Viewpoints, Complexity Management and Examples

PEPML has two viewpoints to express (1) dependencies between entities and (2) the temporal structure of education components. The following explains each viewpoint and its scope concerning PEPML's metamodel and shows the running example from the respective viewpoint.

Dependency Viewpoint

The dependency viewpoint allows the expression of the dependencies between entities. It can be used to express PEPML entities and relations from any package as a graph. Even temporal relations and containment are explicitly denoted as connectors in this viewpoint. Users can be selective about which entities and relations shall be shown. This allows collaboration on specific entities without overwhelming users with the full PEPML model.

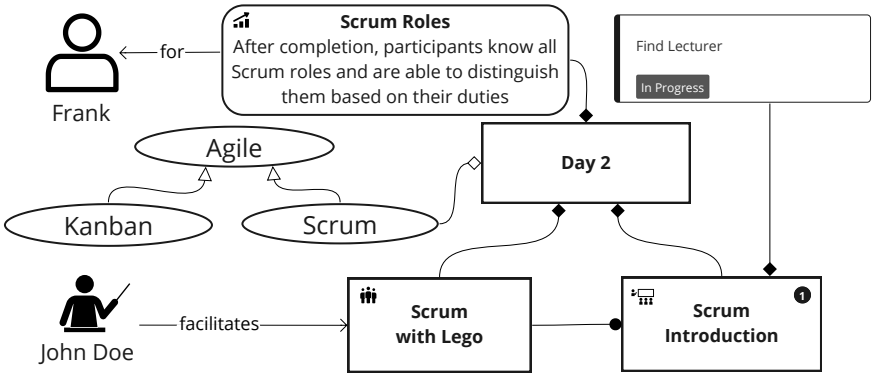

Figure 5.5: Example of the dependency viewpoint

Figure 5.5 shows an excerpt of the running example from the dependency viewpoint. It shows that Day 2 contains two components: an introduction to Scrum, and a group work where participants experience the Scrum process using Lego bricks. For the former, an open task and decorator indicates that the search for a lecturer is ongoing. The latter is facilitated by the educator John Doe. The topic "Agile" is broken down into two more specific subtopics: Kanban and Scrum. Scrum is covered on Day 2 of the programme, and a learning outcome states what participants shall know after Day 2. A typical participant of the programme is represented by the persona "Frank".

Temporal Viewpoint

The temporal structure viewpoint focuses on the temporal ordering and containment of education components. The viewpoint's scope is limited to the *Design* package, and it only allows modelling education components. It uses the horizontal and vertical placement notation presented in Figure 5.4 to express temporal ordering and containment. The (temporal) structure of an education programme is a core element of a programme's design and focus point for collaboration with clients. It gives an overview of the education components and can be used as an index to access specific components.

In Figure 4.10, we presented a possible component structure for the running example in the abstract syntax. Figure 5.6 shows this structure in the concrete syntax. The example has three sequential top-level composites: "Kick-Off", "Self-Learning" and "Closing". The example programme starts with a two-day kick-off event specified using a time-planning composite. Complex board items like the two-day kick-off can be generated (see Section 6.2.1). Alternatively, users can use the standard rectangular shape to add an education component. The time scales on the left of the time-planning composites provide information

on start times and durations of contained components. The kick-off is followed by a self-learning period consisting of two parallel streams: peer reflection and preparing for one out of two Scrum roles. The dashed border of the contained components indicates that they are optional. The programme ends with a two-track closing day with a joint welcome session, followed by separate assessments for the different roles and finishes with handing out certificates. The colouring indicates that entities belong to the same category.

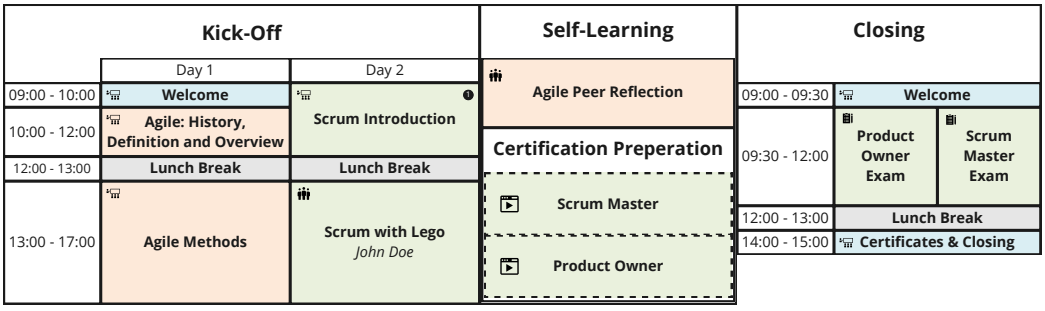

Figure 5.6: Example of the temporal structure viewpoint

Complexity Management

As a complexity management strategy, we allow an entity to be modelled simultaneously on different boards or frames but not multiple times on the same board or frame. We aggregate the information from all entity representations and inform users if there is any conflicting information. Figures 5.5 and 5.6 partially contain the same education components but provide different information. In Figure 5.5, we define that the "Scrum with Lego" has the "Scrum Introduction" as a prerequisite. Figure 5.6 provides the actual order of the components and shows that there is a break in between.

5.5 Summary

This chapter analysed 20 whiteboard solutions for the fit as the basis for visual modelling tool support. We identified Miro as the online whiteboard with the best extension capabilities and developer support. We assessed Miro based on the Physics of Notations and showed which principles for visual modelling languages are impacted when using Miro for visual modelling. Finally, we presented a Miro-compatible visual syntax for PEPML consisting of two viewpoints, one for expressing dependencies and another for the temporal ordering of education components.

The visual syntax presented in this chapter addresses Language Requirement LR8 and is the default visual representation for PEPML models. While PEPML's metamodel introduces several classes and associations, the visual syntax relies on a few geometric shapes and icon-based representations. A legend could explain relevant icons, shapes and associations within a visual model to ease understanding, especially when custom icons are used. Furthermore, PEPML can be used to brainstorm potential programme designs collaboratively with clients and serves as the basis for more detailed designs able to assist in managing programmes. Thus, PEPML fulfils the criteria to be considered lightweight, as defined by Language Requirement LR4.

Whiteboards provide a great degree of freedom and flexibility. Limiting whiteboard users to a specific visual syntax can hinder the adoption of PEPML. For instance, a typical way of modelling in online whiteboards is using canvas models. Those define areas for specific types of information. Any sticky note in such an area represents an instance of the respective type. The next chapter presents the visual modelling tool support for PEPML. As part of this, we explain how user-specific viewpoints can be added and how canvas models can be mapped to PEPML. Hence, we enable education providers to define visual syntaxes for PEPML that fit their specific needs.

Chapter 6

Miro-based Visual Modelling

This chapter presents our Miro-based tool support for visual modelling with PEPML. It directly addresses research question RQ2, concerned with using online whiteboards as primary modelling tools. In particular, we show that PEPML models can be extracted from collaborative online whiteboards. Regarding the language engineering process, this chapter shows the technical feasibility of using PEPML for collaborative visual modelling on online whiteboards. Hence, it contributes to the *Modelling Language Validation* step. We also discuss how our Miro-based visual modelling tool support can be used for other languages.

Section 6.1 states the requirements for our tool support, gives a solution overview and discusses related work. Section 6.2 explains the implementation of the different building blocks. Section 6.3 evaluates our tool support based on PEPML and revisits the requirements we defined for the tooling.

6.1 Overview

In the following, we detail the requirements for our Miro-based tool support for visual modelling. Afterwards, we provide an overview of the building blocks that form the foundation of our tooling and present related work.

6.1.1 Requirements

We derive the following requirements for visual modelling tool support based on online whiteboards. These Tool Requirements (TR) are derived based on the characteristics of these tools, the requirements for modelling PEPML and research question RQ2.

TR1 **Derive formal model**: *The tooling shall be able to derive a formal, machine-interpretable model from a Miro board.*

The information stored within online whiteboards can be interpreted by humans familiar with the visual syntax of the used modelling language. However, the content is not necessarily machine-interpretable since the board only contains shapes or other items. It is necessary to derive a model that conforms to the metamodel of the modelling language. For instance, in the case of PEPML, we want to derive that a rectangle represents an education component.

TR2 **Language-independent**: *The tooling shall be able to support languages besides PEPML.*

While this thesis focuses on PEPML, the tool support shall not be limited solely to this language. It shall be possible to use the approach for other languages as well. This is also necessary since PEPML is meant to be extensible (see LR7), requiring that the tool support can also deal with an augmented metamodel.

TR3 **Support of multiple viewpoints**: *The tooling shall support languages with multiple viewpoints.*

Some languages may just have a single viewpoint, but for PEPML, we already specified two viewpoints in the previous chapter and noted that further will be necessary in the future. Consequently, support of multiple viewpoints is required.

TR4 **Complexity management**: *The tooling shall support merging information of multiple visual models into a single formal model.*

If visual models get too complex, it becomes harder for humans to extract meaningful information [Moo09]. Furthermore, PEPML shall be able to work for professional education programmes of different scales (see LR6), which also requires a complexity management technique at the tool level. We already discussed in Section 5.3.3 that Miro only provides limited support for complexity management. To still be able to manage complex models with Miro, it must be possible to split them into smaller models that are modelled individually and merged when deriving the formal, machine-interpretable model. This also enables collaboration across multiple boards.

TR5 **Syntax validation**: *The tooling shall support the syntactical validation of models.*

Some online whiteboards, like Miro, provide shapes of popular modelling languages but do not provide the possibility for syntactical model validation. If online whiteboard solutions shall be used as primary modelling tools, it must also be possible to check that the visual models they are used to create are syntactically correct.

TR6 **Flexible Visual Syntax**: *The tooling shall allow users to adapt the visual syntax when needed.*

Online whiteboards provide users with the flexibility to create models as they see fit. Restricting them to a particular visual syntax is not possible. Hence, users should be enabled to use custom visual syntaxes.

TR7 **Queryability**: *The tooling shall offer the possibility to access and query the derived model.*

Deriving models is one aspect, but those models also must be accessible. Concerning the tool support, we can break this down into a machine-readable format, access via an API and a querying language that allows retrieving information.

6.1.2 Solution Overview

Figure 6.1 depicts an overview of our Miro-based visual modelling tool support, which addresses the requirements above. The following explains the different building blocks of our solution and the design decisions behind them. Section 6.2 provides in-depth explanations of each building block.

We extended Miro by developing a Miro App (see Ⓐ). Once installed, the app is available on all boards of a Miro team[1]. Our app offers users language-independent as well as language-specific features. For instance, in the case of PEPML, users can add representations of educators to a board or create a time-planning composite. In Figure 6.1, we show two boards to emphasize that the approach can handle multiple viewpoints (cf. TR3). Please note that the approach can support more than two boards. MIRO APP

Our app allows the extraction of board content as a Miro model (see Ⓑ), which conforms to an adapted version of Miro's metamodel (see Section 6.2.3). For our tooling, we use the Neo4j graph database for model persistence EXTRACTION

[1] A Miro team is an organization that shares boards.

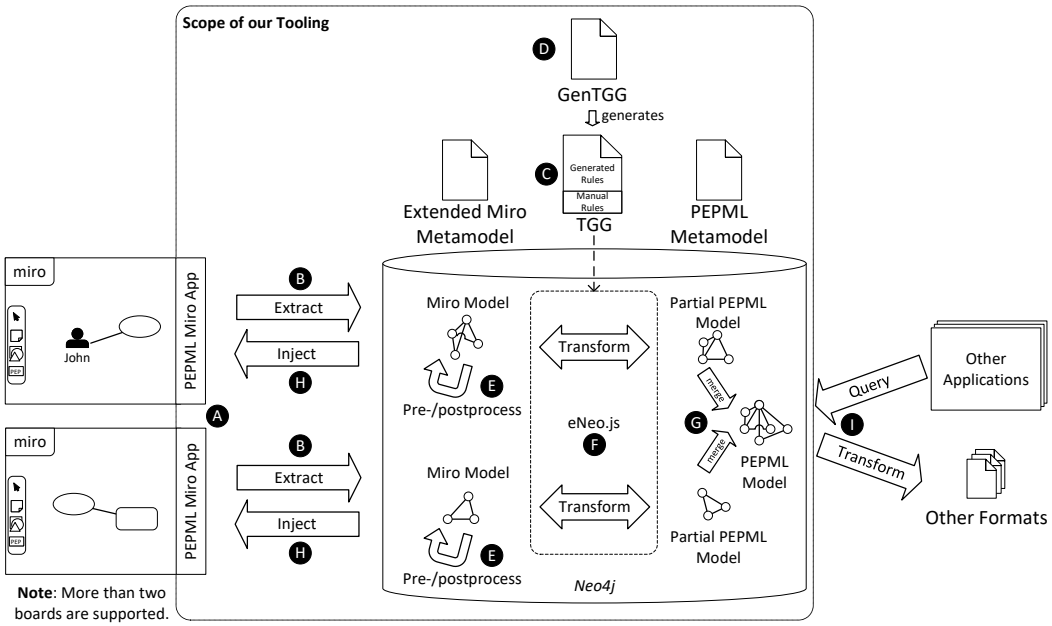

Figure 6.1: Overview of our tool support for visual modelling based on the collaborative online whiteboard Miro

and transformation. We decided against file-based storage, e.g. XMI files, because databases provide greater processing capabilities and endpoints to access data (cf. TR7). We also decided against any EMF-based tooling since EMF models must strictly comply with their metamodels, which hinders extensibility (cf. LR7). Within the Neo4j database, the models are stored in abstract syntax, whereas Miro contains (parts of) the models in concrete syntax. More information on why we chose Neo4j and how models are mapped to the Neo4j data model is provided in Section 6.2.2.

TGGs for Transformation

Once we have a Miro model, we can apply exogenous model transformations to obtain an instance of the desired target modelling language. We decided to use TGGs as a transformation language (cf. ⓒ) since they provide the possibility for bidirectional transformation. This means we can use TGGs to transform a Miro model into a PEPML model and vice versa. We specified TGGs for both of PEPML's viewpoints. Additionally, we introduce a TGG for a custom canvas model for planning the activities of an education component in Section 6.3. Users can choose upon extraction the type of model displayed on the Miro board, and the respective TGG is used for transformation.

Generating TGGs with GenTGG

Writing a TGG can be time-consuming, error-prone and often repetitive. We developed a language called GenTGG, which allows the definition of mappings between metamodels in a succinct format, based on which we can generate a TGG (cf. ⓓ). GenTGG cannot express every TGG but simplifies

the specification of a specific subset of TGG rules that is particularly relevant to our use case. Chapter 7 explains GenTGG in detail and shows that it is expressive enough to cover nearly the full TGG required for PEPML. As we explain in Section 6.3, only the Negative Application Conditions (NACs) for one rule of each viewpoint required manual specification.

Previous versions of PEPML's tooling hardcoded the transformations between Miro and PEPML elements [WE23]. Consequently, changing the transformations required changing the source code of the app. Using TGGs instead of hardcoded transformation has several advantages but also some disadvantages. The mappings between the source/target language and the respective metamodels are maintained outside the code. They can even be manipulated at runtime, e.g. to extend the language and its visual syntax without having to redeploy the application. Of course, alterations must be done with care since they can lead to models not confirming to the metamodel of the language. Moreover, users are limited to the expressiveness of TGGs and not every scenario can be addressed.

Hardcoded Transformation vs. TGGs

To extend the application scenarios of TGGs, we use preprocessors that add additional information to a model before executing a model transformation (see ❹). For instance, determining spatial relations, like containment, based on (x,y) coordinates is complicated up to impossible, depending on the used TGG approach. In Section 6.2.4, we introduce multiple language-independent preprocessors, including one for computing spatial relations. In cases where preprocessing and TGGs are not sufficient for extracting models of target languages, users can always resort back to custom code transformers. However, we show throughout the rest of this thesis that TGGs combined with our preprocessors are expressive enough to cover all scenarios relevant to our approach. In particular, it is sufficient to describe the relation between PEPML's metamodel and its visual syntax.

Preprocessing

We developed a new lightweight, JavaScript-based TGG engine called *eNeo.js* to execute the model transformations (see ❺). We decided against using the existing eMoflon::Neo TGG engine [WA21] for our tooling since it only allows TGG operationalization at development time within an Eclipse IDE. Furthermore, it does not offer an API for remote interaction with the engine, which limits its use in web-based projects. eNeo.js is developed using web technologies and supports TGG operationalization at runtime for FWD and BWD. Thereby, we can enable users to alter TGGs within the browser without having to redeploy the application. In Section 6.2.5, we discuss different options to extract PEPML models from Miro boards and show that

eNeo.js

FWD and BWD transformation are sufficient for our use case when combined with a model merging approach. We also discuss which use cases can be covered by eMoflon::Neo and its advanced operationalization, like the fault-tolerant model synchronization approach developed by Weidmann [Wei22].

Our tooling supports multiple viewpoints (cf. TR3) by allowing users to describe partial models on Miro boards and later merge them into a single coherent model of the target modelling language (see ⓖ). For the merging to be successful, overlapping information on the different boards must be equal or complement one another. In the case of contradicting information, e.g. multiple different statuses for a task in a PEPML model, users are informed about the problem. Similarly, syntax problems (cf. TR5) can be detected during extraction, preprocessing, transformation and merging. When problems are detected, they can be highlighted to assist users in resolving these issues. Information about the origin of elements is stored in the merged model to trace information back to its source. Being able to merge multiple partial models also serves as a complexity management technique (cf. TR4) since users are not forced to model complete models on a single board. In the case of PEPML, users could specify on one board that an educator knows a topic and express on another board which education component covers this topic.

Syntax Validation

Our tooling is not limited to PEPML and its two viewpoints. The extraction, preprocessing, transformation and syntax validation part of our app is generic and can be reused for other modelling languages as well. Users can define their custom visual syntaxes by providing TGGs for their transformation (cf. TR6). Additionally, a custom TGG plus a custom target metamodel allows the usage of our tooling for languages besides PEPML (cf. TR2). In Section 6.2.1, we provide detailed information about the app's implementation and how it can be reused for other languages.

Usable for other Languages besides PEPML

A TGG can also be used to transform an instance of the target language (see ⓗ), e.g. PEPML, back into a Miro model, which we can then inject via our Miro app into a Miro board. Before injection, post-transformation processing may be needed, e.g. to add (x,y) coordinates of new elements.

Injection

Once a target model exists, it can be queried for information or transformed into other formats (see ⓘ). For this purpose, Neo4j provides a powerful querying language called Cypher and an API to access the database (cf. TR7).

Queryability

6.1.3 Related Work

Collaborative online whiteboards like Miro are a recent development that was enabled with the release of HTML5 in 2014. The broader adoption of such tools came during the pandemic. Several studies show the usefulness of such tools for co-design activities [BAHT21, LZXL21]. Within these studies, the whiteboards are used as informal modelling tools without trying to extract information conforming to a specific metamodel. PEPML is not only usable for co-design, but with our tooling, we can extract formal, machine-interpretable models afterwards, which can be processed further.

Co-Design with Online Whiteboards

Stillhart and Moos [SM20] developed a tool to extract instances of the Context Mapping Language (CML) from Miro boards. Their approach is limited to CML and only supports three Miro templates from which these models can be extracted. In his Bachelor's thesis, Bernal Tejedor [BT23] developed a syntax check for models described on Miro boards. His work was a first step to generalizing the tool support we initially developed for PEPML [WE23]. He defined his own JSON-based format to describe metamodels and mappings to Miro items. While his approach allows syntax validation and can inform users about syntax issues, it does not extract a model and instead directly works on the Miro representation.

Extracting Models from Miro

The approach Img2UML [KC13] can convert images of UML class diagrams to formal models in the XML Metadata Interchange (XMI), a serialization format for models provided by the OMG. When modelling UML class diagrams on online whiteboards, such as Miro, the board's content could be exported as an image and transformed into an XMI-based model using Img2UML. Approaches like SUMLOW [CGH08] and, more recently, OctoUML [VJC17] allow UML class diagrams to be sketched using freehand or pen input on e-whiteboards[2], which are then converted into machine-interpretable UML models. Schäfer et al. [SvdALS23] list various approaches for other languages, such as Flowcharts, Petri Nets or general recognition of geometric shapes. Additionally, they present Sketch2Process, an advanced approach for converting freehand drawings of BPMN process models into an XML representation of the depicted process. Such approaches can be very beneficial to converting freehand drawings on online whiteboards into actual models of specific languages but cannot assist with other languages. FlexiSketch [WSG19] goes a step further and allows users to define metamodels for their sketches. Thereby,

Images and Freehand Drawings to Formal Models

[2] e-whiteboards are electronic whiteboards that often support in-room collaboration via multi-touch input and are different from collaborative online whiteboards such as Miro, which support real-time remote collaboration via the Internet.

sketches of domain-specific modelling languages can be converted into formal models. FlexiSketch focuses on graph-based diagrams and freehand drawings. While graph-based diagrams are relevant in online whiteboards, canvas models are also very popular. Furthermore, freehand recognition is not required since we can extract the geometric shapes and connectors programmatically.

By exporting the content of an online whiteboard using its API, we can extract models conforming to the whiteboard's metamodel. Afterwards, we can use model transformations to convert this into an instance of our target language. Unidirectional transformation languages such as ATL[3] can be used for this purpose. However, we decided to use Triple Graph Grammars (TGGs) [Sch94] since they allow transformations in both directions based on a single set of rules. TGGs can be operationalized for forward/backward transformation (FWD/BWD) [KLKS10], consistency checks [LAS17] and unidirectional/bidirectional model synchronization [GW06, GW09, Wei22]. As we explain later in this chapter, FWD and BWD are sufficient for our approach. We do not extend the capabilities of TGG-based model transformations in this thesis, but with GenTGG, we reduce the effort of writing specific TGGs.

Model Transformation

Several tools exist to perform TGG-based transformations [HLG+13, HNB+14, WA21]. These tools are all based on Eclipse, and except for eMoflon::Neo [WA21], they all use EMF for representing models. In contrast, eMoflon::Neo stores and transforms models within a Neo4j graph database[4]. We initially based our tooling on eMoflon::Neo, but ultimately, we developed our own JavaScript-based TGG engine to be independent of Eclipse and enable TGG operationalization at runtime.

TGG Engines

Seifermann and Henß [SH17] compare QVT-O and HenshinTGG for the bidirectional transformation of different UML models in textual and graphical representation. Their findings show that bidirectional transformation can be used for such use cases but also show the limitations of current tools.

BX for Concrete Syntaxes

As we explain throughout this chapter, we combine FWD and model merging to integrate multiple views into a single PEPML model. Sharbaf et al. [SZS23] provide an extensive overview of model merging techniques. We present in this chapter a pragmatic model merging approach, which can detect conflicts and make users aware of the issues but requires manual conflict resolution. We decided against techniques with automated conflict resolution because they can lead to unintended changes.

Model Merging

[3] https://eclipse.dev/atl
[4] https://neo4j.com

6.2 Implementation

The previous section stated requirements and gave an overview of our tool support. This section presents our tool support in detail. The prototypical implementation is available on GitHub[5].

6.2.1 Miro App

We developed a Miro app to access our tool support for PEPML within Miro. This app has a modular design and contains parts specific to PEPML and generic parts that can be reused for tool support for other languages. We first explain the user perspective of the app and then the app's structure.

User Perspective

Through an icon in Miro's toolbar, users can access the UI of our Miro app, which is structured into four tabs (see Figures 6.2 and 6.3). The functionality provided by these tabs is presented in the following.

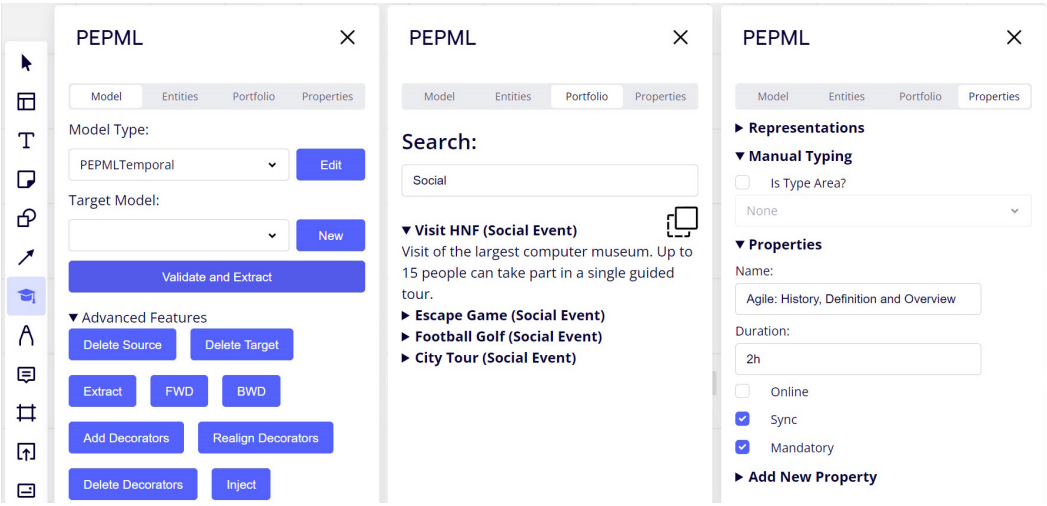

Figure 6.2: Screenshot showing the *Model*, *Portfolio* and *Properties* tabs of our Miro app

The *Model* tab (see left of Figure 6.2) enables users to validate the model and extract it as a PEPML model to the database. Any errors identified during validation are presented as a list to the user, and by clicking on an item in this list, the respective element is highlighted. When highlighting elements on the board, we store the original styling of the element and undo

Model Tab

[5] https://github.com/dwolters/pepml

the style changes once the element has been inspected. The *Model* tab also allows adding decorators to Miro items that indicate specific properties of a PEPML entity (see Section 5.4.1). The *Model* tab also provides access to the model transformation control centre, where users can manipulate metamodels and TGGs and set transformation options, e.g. the rule application order.

Portfolio Tab

The *Portfolio* tab (see middle of Figure 6.2) allows importing PEPML elements from archived PEPML models. For this, it provides a search form to browse the portfolio containing PEPML models indexed for reuse. Portfolio items can then be added via drag & drop to the board and, thereby, also to the PEPML model.

Properties Tab

The *Properties* (see right of Figure 6.2) tab allows editing, adding and removing properties of PEPML entities. When the *Properties* tab is open, users can select a board item and the properties of the corresponding PEPML entity are shown. Since PEPML is extensible and allows arbitrary properties to be added to any entity, users are free to add and remove properties. Only the class properties defined by the PEPML metamodel cannot be removed.

Entities Tab

Since PEPML's visual syntax is based on regular Miro shapes, PEPML entities can be added using Miro's regular toolbar. Alternatively, users can add PEPML entities using the *Entities* tab. As shown in Figure 6.3, the *Entities* tab allows the customizable generation of time-planning composites and includes the default images for PEPML entities.

The *Entities* and *Portfolio* tabs are specific to PEPML. The *Properties* and *Model* tabs are generic and can be reused when building tool support for other visual modelling languages.

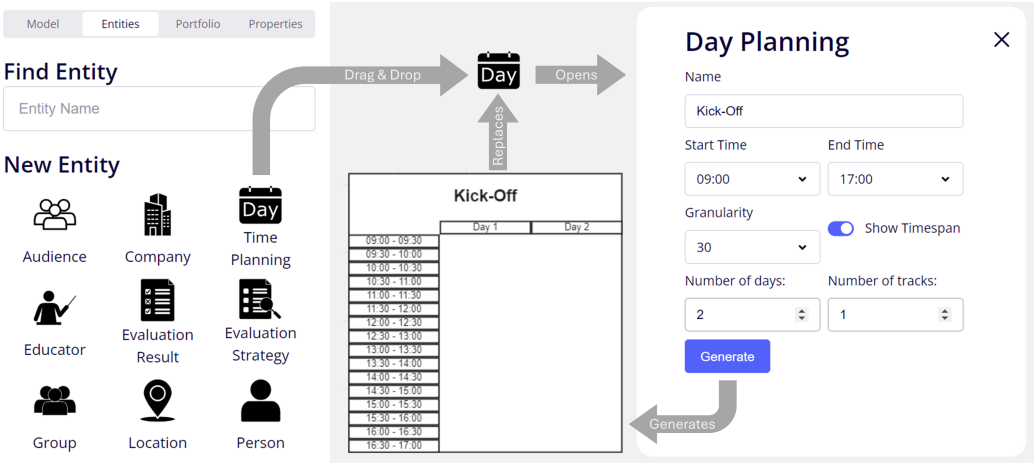

Figure 6.3: Annotated screenshot showing the *Entities* tab of our Miro app and the dialog to generate time-planning composites

App Structure and Implementation

Figure 6.4 provides an overview of our Miro app's structure. The `App` component handles the main UI logic of the app. This component is opened when a user clicks on the app's icon in Miro's toolbar. Every tab in our Miro app is encapsulated as a UI component. The components for each tab register at the `App` component with their name and the events they want to receive. The `App` component subscribes to receive these events from the Miro board and passes them to the active tab. For instance, the *Properties* tab listens to the `selection` event to know for which item the properties shall be shown.

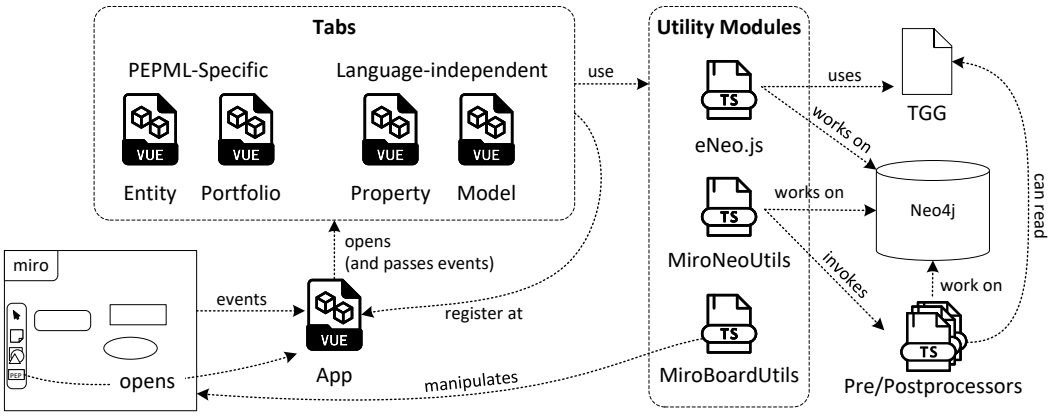

Figure 6.4: Simplified structural overview of our Miro app

The PEPML app also includes three utility modules. The `MiroNeoUtils` module handles the extraction of Miro board content into a Miro model stored in Neo4j. Moreover, the module handles the preprocessing, which is done prior to the model transformation. The injection of PEPML models into a Miro board is handled by this module as well. More information on the model extraction and transformation is given in Section 6.2.3. The `eNeo.js` module allows the operationalization and execution of a TGG for FWD and BWD. Furthermore, it provides functions to copy, compare, delete and merge models stored within a Neo4j database. The `MiroBoardUtils` module provides advanced functions to interact with a Miro board, like placing decorators or highlighting items.

Utility Modules

Our Miro app is developed in TypeScript using Vue.js[6] as a frontend technology. Miro does not prescribe a specific frontend framework for Miro apps since the UI of apps is embedded via an HTML inline frame. We decided to use Vue.js due to familiarity with the framework, but other frameworks like React or Angular could also be used.

Frontend Technology

[6] https://vuejs.org

6.2.2 Storing Models in Neo4j

File-based vs. Database Storage

For storing models, we had to choose between file-based or database storage. If our main goal would be to export a model file that can be further processed using modelling tools, file-based storage in the XMI format would be the ideal choice. However, Requirement TR7 goes beyond storage and states the need for a querying language and an API to access the model. While querying languages exist for some file-based formats, a database provides storage, querying capabilities and an API out of the box. Also, we deal with multiple models simultaneously, as we see in later sections, which requires an application internal representation of these models, which could also be addressed via a database.

Rationale for Neo4j

The ranking of DB Engines[7] shows that relational databases are still the most dominant database technology. Yet, we decided to use a graph database since the models we are concerned with in this thesis are typed graphs. Hence, the data model of graph databases is an intuitive match for storing such models. Also, graph querying languages provide advantages over SQL since they can traverse an arbitrary number of edges, while SQL is limited to a finite number of joins in a query. We decided to use Neo4j because it is the highest-ranked pure graph database[8] and provides a powerful graph querying language called Cypher.

Schemaless

Most graph databases, including Neo4j, are schemaless databases. While they prescribe that data is stored as a graph consisting of nodes and relationships, there is no schema restricting which nodes/relationships can or must exist. Since we want PEPML to be extensible, a schemaless database is a good choice because it does not require types and fields to be known in advance.

Graph Metamodel

Even though Neo4j is schemaless, data has to follow the Neo4j graph metamodel, which is shown in Figure 6.5. Data in Neo4j is organized into nodes and relationships. Nodes can have an arbitrary number of labels that can be used to cluster nodes. Relationships are binary, directed edges between a source and a target node. Each relationship has exactly one type. However, multiple relationships can have the same type. Neo4j supports storing arbitrary properties on nodes and relationships. Several atomic property types exist, like `Boolean`, `String` or `Number`. A property can also be an array of a single atomic type, but it cannot contain an object or an array containing different atomic types.

[7] https://db-engines.com/de/ranking
[8] https://db-engines.com/en/ranking/graph+dbms

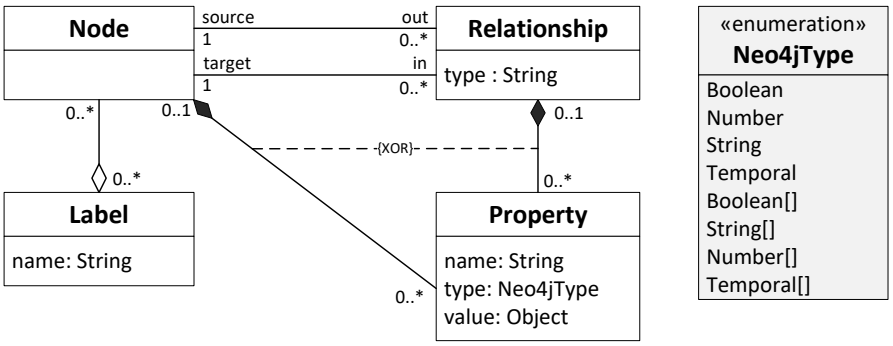

Figure 6.5: Neo4j metamodel

For storing a model in Neo4j, we store each object as a node and each link as a relationship. Object types are denoted as labels of the respective node. For instance, if an object is an instance of a class C_A, which is a subclass of C_B, it will have two labels, C_A and C_B, that indicate that it can be treated as an instance of any of these classes. Links are instantiations of associations, and the association type is stored as a relationship type.

Labels to Denote Types

We follow approach of eMoflon::Neo [WA21] to avoid name clashes between multiple metamodels and add the name of the metamodel as a prefix to the respective label. For instance, the label for the class `Entity` of PEPML's metamodel has the label "PEPML_Entity". Like eMoflon::Neo, we also use the `enamespace` property to differentiate between different models in the same Neo4j database. Consequently, we can store multiple models within the same database but with different values for the `enamespace` property.

Class Prefixes & Model Namespace

6.2.3 Extracting Miro Models

The content of a Miro board has to be extracted to a Miro model stored in the Neo4j database to be transformed into an instance of a target modelling language like PEPML. The extraction process explained consists of five steps and is depicted in Figure 6.6.

As a prerequisite to the extraction, we identify all dangling connectors. A dangling connector is not attached to a start or end item (see Figure 6.7.a). Miro allows having dangling connectors, but PEPML does not. A connector may be close to the border of an item but not connected to it. Since connectors do not have start/end coordinates, we cannot identify such cases programmatically. Instead, we highlight dangling connectors and let users decide to solve or ignore the problem.

Dangling Connectors

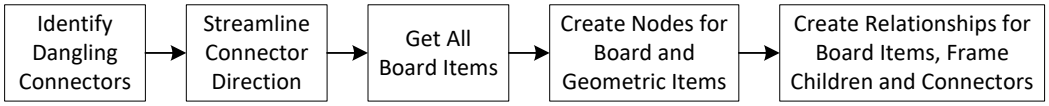

Figure 6.6: Process to extract Miro board content and store it as a Miro model in Neo4j

Internal vs. Visual Representation

As shown in Figure 6.7.b, the visual and internal representation of connectors does not always match. The internal denotation (grey labels in the figure) of start and end items is unimportant to users. For them, the direction of a connector is denoted by the start/end stroke caps. Before the actual extraction, we align the internal and visual representation of connectors since the internal representation is important for proper model transformation. Otherwise, both cases (aligned and not aligned representations) would need to be covered during transformation. If a connector only contains a single arrowhead, it is expected to be the end cap. A (filled) diamond used to represent containment relations is always assumed to be the start cap.

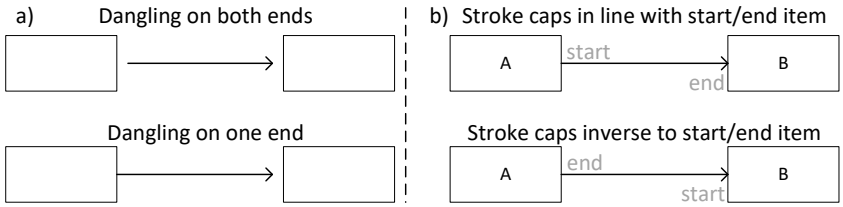

Figure 6.7: Examples for (a) dangling connectors and (b) differences between the visual and internal representation of connectors

Storing the Miro Model in Neo4j

In the third step, we obtain all board items using Miro's WebSDK. We group these items into connectors and geometric items. Connectors are represented as relationships in Neo4j, and to create those, their source and target node must exist. Consequently, geometric items are added first. Alongside the geometric items, we add a node representing the board and the model itself. The latter model node indicates of which metamodel the model is an instance. In addition to adding all connectors as relationships, we create relationships between frames and their children and between the board node and all geometric items.

Inaccessible Properties

When creating a node for a geometric item, we transfer most of the item's properties to the node in Neo4j. Unfortunately, not all properties of an item are programmatically accessible. For instance, the property `locked` of `Item` is neither exposed by the API nor the SDK, and we cannot determine if items are aggregated into groups. The development roadmap indicates that grouping items will be supported in the WebSDK/API in the future.

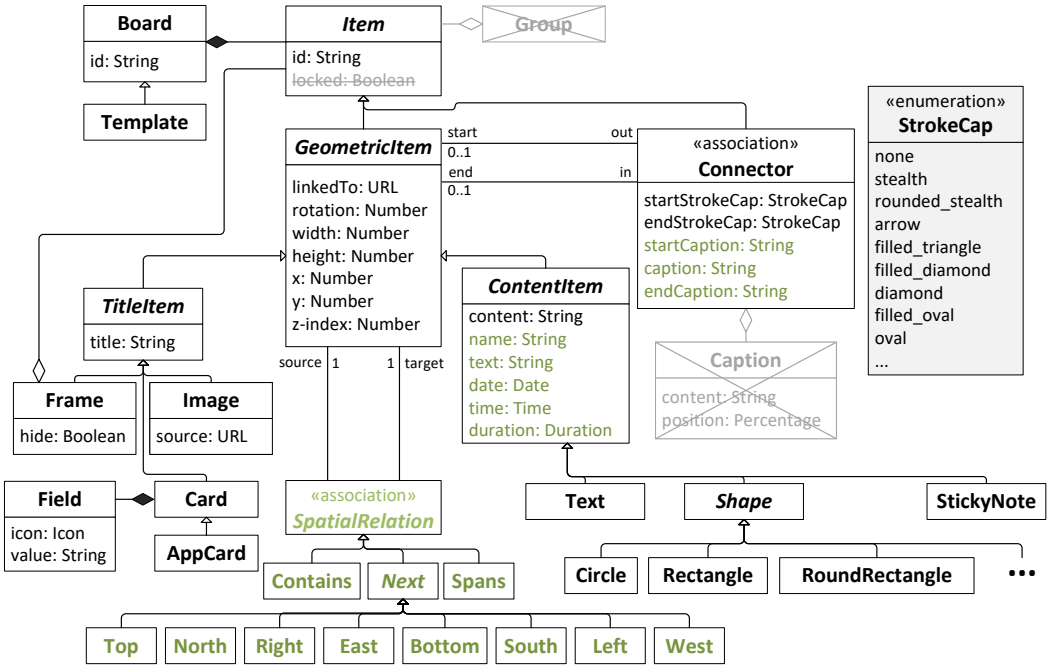

Figure 6.8: Adapted Miro metamodel used by our approach

Figure 6.8 shows an adapted form of the Miro metamodel, which we initially presented in Section 5.3. The properties and classes not supported by our approach are greyed out and crossed out. Consequently, **every time we refer to a Miro model, we refer to an instance of the metamodel in Figure 6.8.** In the next section, we explain how we use preprocessors to prepare the Miro model for model transformation. One of these preprocessors translates the connector captions into single-value properties of the connector, which is why they are greyed out in the adapted metamodel.

Adapted Miro Metamodel

6.2.4 Preprocessing Models

After extracting a Miro model, we use preprocessors to enrich the information in the Miro model. As we show in this section, it would not be possible to use TGG-based model transformations to transform a Miro model into a PEPML model without preprocessors. Preprocessors are added as modules to our Miro app. They can enrich and manipulate the extracted Miro model. A preprocessor is written in JavaScript/TypeScript and can use Cypher queries to manipulate a model. The preprocessors we show in this section are language-independent and can be reused for other modelling languages besides PEPML. After preprocessing, we have an enriched Miro model still conformant with the Miro metamodel in Figure 6.8.

Preprocessors

Caption Preprocessor

Partion Connector Captions

Miro supports adding captions to connectors at different positions relative to the start and end. For instance, a caption could be placed at position 50%, meaning it is in the middle of the connector. Distinguishing between a slight change in the positioning of captions is difficult. Therefore, we provide a preprocessor that partitions the caption space into three intervals and stores each caption in a different property: 0% to 33% is the `startCaption`, 34% to 67% is the `caption` and 68% to 100% is the `endCaption`. Figure 6.9 shows an example of how the caption preprocessor transforms a caption object into a connector property and how this information is transformed into a PEPML model.

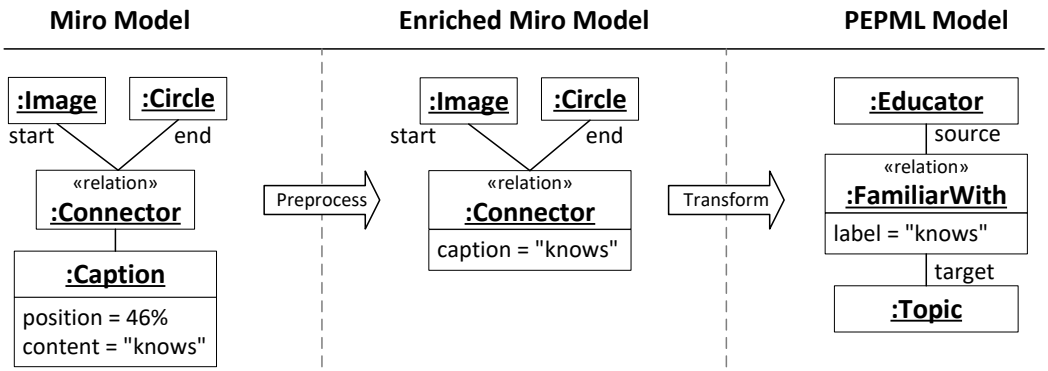

Figure 6.9: Caption Preprocessor Example

Content Preprocessor

Content Preprocessing

Similar to captions, we map the property `content` to different properties. The text content of items, such as text items or shapes, can be organized in multiple paragraphs, with each paragraph having multiple lines. Interpreting the HTML code used for this formatting is beyond the capabilities of TGGs. To address this limitation, we extract multiple properties from the text content and make it available for further processing with TGGs. In Figure 6.10, we show how the `content` property is broken down into the `name` and `description` properties. Additionally, the content preprocessor can identify timeframes, durations and dates. If a line does not contain any of these data types and does not contain a full stop, it is considered the item name. All remaining lines are stored in the `text` property without HTML tags. The HTML-formatted content remains available via the `content` property. These new properties are shown in the adapted Miro metamodel in Figure 6.8.

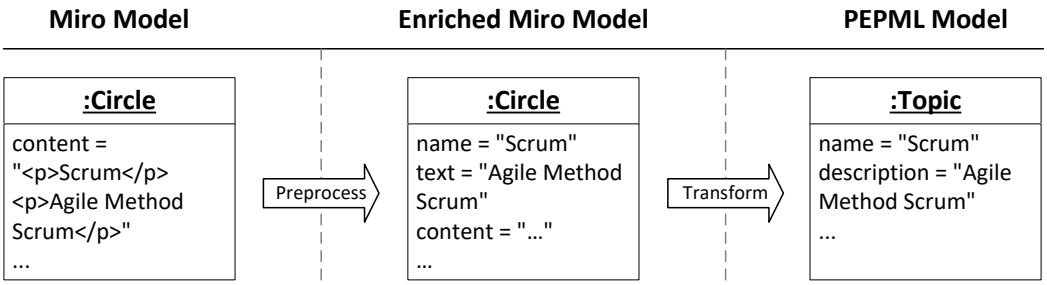

Figure 6.10: Content Preprocessor Example

Spatial Relations Preprocessor

Relations between items are explicitly expressed via connectors in the Miro metamodel. In addition, the positioning of items can implicitly encode semantic information as well [Moo09]. Sophisticated property conditions are needed to transform such information with TGGs, and only some TGG approaches support them. Even if they are supported, calculating such spatial relations in property conditions is error-prone. The spatial relation preprocessor adds spatial relations explicitly to the enriched Miro model, which allows transformation without the need for complex property conditions. For this, it uses the `SpatialRelation` association added in Figure 6.8.

Positioning Encodes Information

Figure 6.11 shows a concrete example of what information is added by the spatial preprocessor. The Miro model on the left contains two disconnected rectangles. However, based on their coordinates and dimensions, we can deduce that $r1$ contains $r2$. The preprocessor makes this spatial relation explicit in the enriched Miro model.

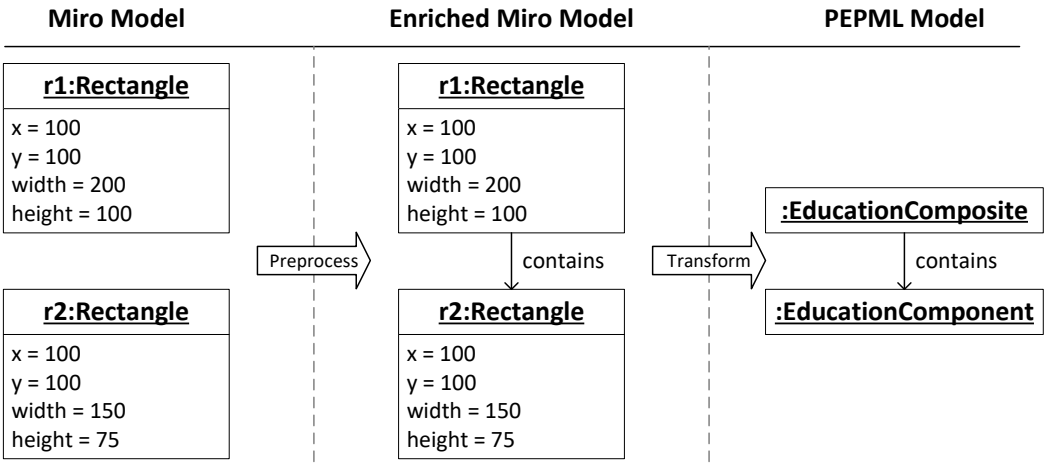

Figure 6.11: Spatial Preprocessor Example

Figure 6.12.a provides a vocabulary for the areas surrounding a geometric item. Our spatial preprocessor allows adding spatial relations for all areas except those diagonal to the item. Adding all spatial relations is typically not needed and would result in unnecessary computations. As indicated in Figure 6.4, preprocessors have access to the TGG used for model transformation. Relevant spatial relations can be extracted from this TGG, and only those are added to the Miro model. For instance, the TGG for PEPML's dependency viewpoint uses the spatial relation `south` to relate text and image items.

Identify Required Spatial Relations based on TGGs

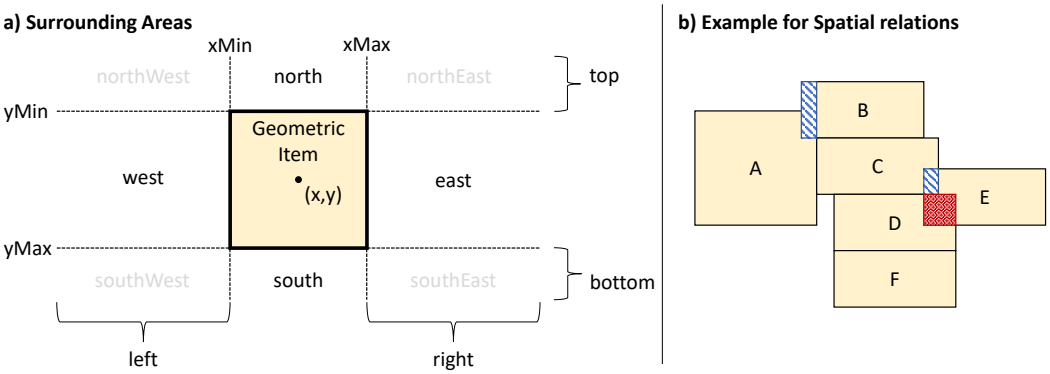

Figure 6.12: (a) Names for the areas surrounding a geometric item, (b) Examples of correct and incorrect spatial placement

Figure 6.12.b shows examples of spatial placement. The items B, C and D are all to the right of A. C is also to the east of A since its $yMin$ and $yMax$ values are within the bounds of those of A. Item E is to the right of C and, thereby, transitively to the right of A. Items can overlap up to a user-defined threshold. Let us assume the overlaps of A and B as well as C and E are still below the threshold, but the overlap of D and E exceeds it. If an overlap exceeding the threshold does not yet represent a containment, the preprocessor throws an error, interrupting the model extraction process.

Examples of Spatial Relations

In addition to positioning items next to each other, they can contain one another. Figure 6.13 illustrates different forms of containment. We consider a spatial relation a containment if the centre of one item is within the bounds of the other item. A containment is unclear if the bounds of one item are not within a user-defined tolerance inside the bounds of the other item. In the case of an unclear containment, an error is thrown, and users are notified where this error occurs. Users can then resolve the error and restart the extraction. Since the option to determine the z-index did not exist when the tooling was implemented, we do not distinguish if item A contains item B or if A covers B. However, this could be changed in future versions.

Containment Relations

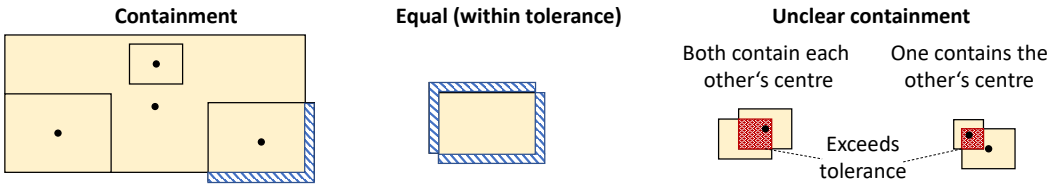

Figure 6.13: Examples of the containment relation based on Miro item placement

Ignore Items Preprocessor

A parametrizable preprocessor exists that removes items from the Miro model. For instance, this can be used to ignore empty shapes or rectangles containing the time frame information in time-planning composites. Without removing such items, they may be translated into unintentional PEPML entities.

IGNORING ITEMS

In addition to presented preprocessors, our tooling allows users to define language-specific preprocessors. For instance, we add a PEPML-specific preprocessor for time-planning composites, which we explain in Section 6.3.2.

LANGUAGE-SPECIFIC PREPROCESSORS

6.2.5 Transformation and Merging

After preprocessing, the enriched Miro model is transformed into an instance of the target modelling language. In the following, we first explain why pure FWD or model synchronization do not suffice for our purposes. We then introduce a combination of FWD and model merging that satisfies our requirements. We use PEPML as an example target modelling language but also discuss which transformation approaches can work for languages with different characteristics.

Forward Transformation

Figure 6.14 shows the case in which FWD is used to create a PEPML model. FWD is executed on the enriched Miro model and builds up a PEPML model with correspondences to the Miro model. The correspondences are important for our tooling since we use them to identify which PEPML entities are represented by a Miro item. FWD is sufficient if there is just a single source from which a model is extracted and if all information of the target model is present in the Miro model. This is not the case for PEPML since it works with multiple viewpoints (cf. TR3) and boards (cf. TR4), and not all information stored in a PEPML model can be represented visually on a Miro board. For instance, using the *Properties* tab, we can add information to PEPML entities

not represented by Miro items. Such custom properties are problematic when the Miro model is updated since we no longer can discard the target model and build a new one without losing the values of those properties.

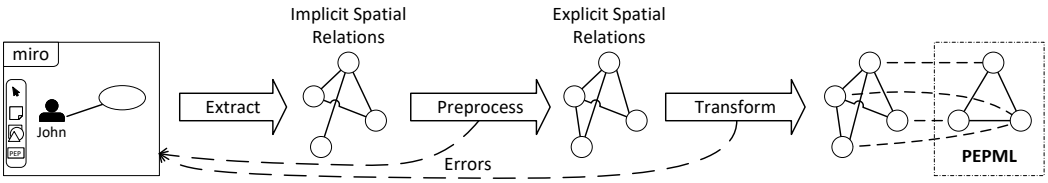

Figure 6.14: Forward transformation from Miro to PEPML

Copying Custom Properties

A simple approach to avoid information loss when the target model is always built from scratch using FWD is shown in Figure 6.15. Initially, we build up a target model like in Figure 6.14. When the board content changes and the target model is updated, a completely new target model is built using FWD, and we manually copy all custom properties that the new model has not set. This approach works well if only a single board exists.

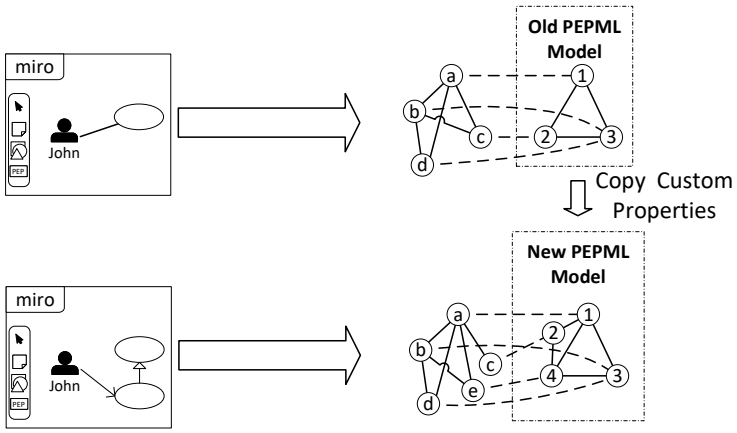

Figure 6.15: Keeping custom properties when only using forward transformation

Model Synchronisation

Model synchronization synchronizes changes between the source and target model. Hence, it does not fully build up the target model, and custom properties would be kept. For model synchronization to work, changes in the source/target model must be denoted explicitly [Wei22]. Figure 6.16 shows how this can be done by comparing a newly extracted Miro model to the previously extracted one. Then, model synchronization can be used to update the old target model.

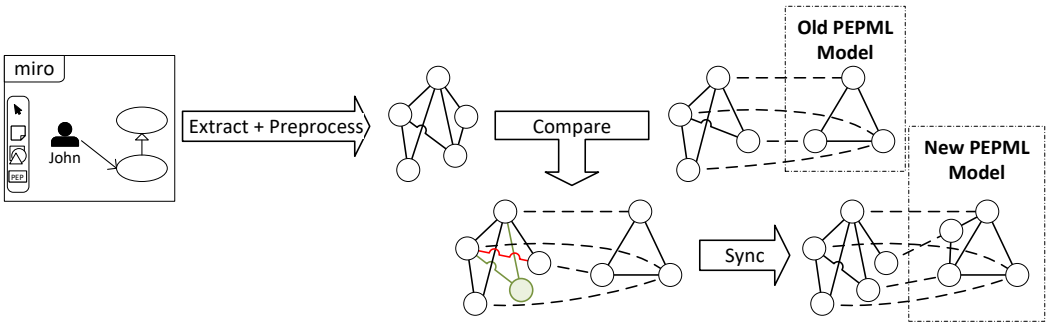

Figure 6.16: Forward model synchronisation from Miro to PEPML

The model synchronization approach works well if information from a single board is extracted. If information from multiple boards is synchronized, problems can occur during the synchronization. Stünkel et al. [SKLR21] refer to this as the view-update-problem. For instance, a user may just want to remove a representation of a PEPML entity on one board but keep it on the other. Model synchronization always tries to reach a consistent state and may delete the entity along with all corresponding items in models of other Miro boards. While the items on the other boards would not be deleted immediately because Miro models have to be injected for changes to apply, custom properties of affected entities and the correspondences would be lost.

Limited to one Board per Model

Another caveat of model synchronization is that property changes are not synchronized [Wei22]. Consequently, if properties represented by Miro items change, like a former mandatory education component is now optional, these changes are not synchronized. To deal with this problem, items with changed property values could be marked as new nodes while marking the nodes that formerly represented these items as deleted. However, this might lead to the deletion of the corresponding PEPML entity and the creation of a new one, which again results in information loss. Alternatively, properties could be represented as nodes, but this requires far more complex metamodels and TGGs, which makes this workaround difficult to use in practice.

Synchronizing Property Changes

Forward Transformation + Model Merging

FWD or model synchronization alone is only sufficient for languages with a single viewpoint modelled on a single board. In PEPML's case, we can have multiple overlapping viewpoints and boards, e.g. two different boards may represent the same entity and specify different information. We focus on FWD in the following and combine it with a model merging approach to address support for multiple viewpoints and boards.

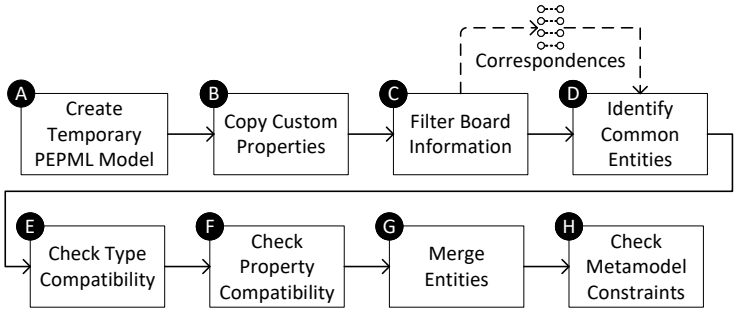

Figure 6.17: Process of merging new information provided by a Miro board into an existing PEPML model

Figure 6.17 shows the process behind this combined approach, and Figure 6.18 depicts an example of the process in action. Step Ⓐ creates a temporary PEPML model based on the current board content using FWD. In Step Ⓑ, we copy the custom properties of the entities connected to the respective Miro items from the old PEPML model, following the same idea as in Figure 6.15. Afterwards, in Step Ⓒ, we remove all existing information from this board from the old PEPML model. To be able to do this, we track the sources of information within a PEPML model, e.g. which entities, relations and properties are obtained from which board. In Figure 6.18, the information source is encoded via colours. Blue represents information provided by the board on the left, yellow represents information from another board, and black denotes that the information is contained on both boards. If a particular information, e.g. an entity's description, is provided by multiple boards, we keep this information in the filtered PEPML model. We remove a property, relation or entity if it is only provided by the board from which the updated information shall be added.

In Step Ⓓ, we identify common entities of the new temporary PEPML model and the filtered version. Common entities can be identified based on former correspondences, which are extracted from the old PEPML model before removing the data provided by the respective Miro board. Alternatively, we assume two entities are the same if they have the same name. For potentially common entities, we check the type compatibility in Step Ⓔ. The type of two entities, E_1 and E_2, is compatible if it is the same or if one is a subtype of the other. For instance, if E_1 is an instance of `EducationComponent` and E_2 is an instance of `Assessment`, the types would be compatible. If E_2 is instead an instance of `Topic`, the types would not be compatible, and the entities would no longer be marked as common entities.

Type Compatibility

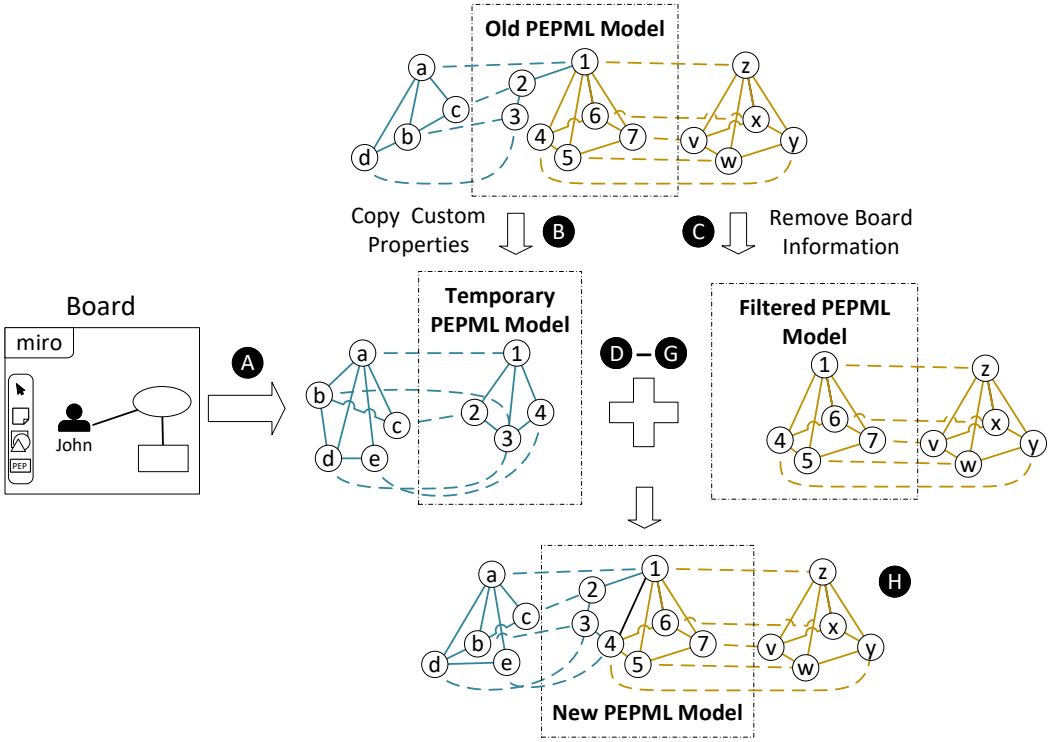

Figure 6.18: Merging updated information into an existing PEPML model

Before merging two common entities, we check the compatibility of property values in Step ❶. If entities that shall be merged specify a value for the same property, it also has to be the same value. If they specify different values, users must be involved to resolve the conflict. If a property is just specified by one of the entities, this value is kept. In other words, the information of the merged entities can complement but not contradict one another.

Property Compatibility

In Step ❻, we merge the common entities by replacing the two entity nodes in the database with a single one. As part of this, existing relationships with other entity nodes must be redirected. Similarly, the property information of both entities is added to this new node.

Merge Entities

Lastly, we check in Step ❽ if the metamodel constraints are fulfilled. These metamodel constraints are expressed by the multiplicities in the metamodel and the additional constraints we define in Section 4.3.2. A violation could be that the merged entities each specify a different status. The merged entity would then be associated with two separate statuses, and the metamodel only allows one. Hence, users must be involved to fix this issue.

Evaluating Metamodel Constraints

6.2.6 Postprocessing and Injection

In the previous subsection, we showed how PEPML models can be obtained from Miro boards and merged with PEPML models from other boards. This subsection covers the reverse direction and explains how PEPML models can be injected into Miro boards to be represented visually. For injection, we have to differentiate between two cases: (1) Injecting content that has no relation to anything on the board and (2) adding content related to existing items on a board. Case (1) occurs if we have an empty board or when we import entities from the portfolio. As we explain in Chapter 9, the portfolio contains archived PEPML models that can be reused when building professional education programmes. Case (2) occurs if we add entities of the PEPML model that are not shown on the board yet. Such entities may be represented on other boards or are extracted from other applications. We explain how entities are extracted from other applications in Chapter 8.

Injecting & Postprocessing

Figure 6.19 shows the Case (1) of importing a PEPML model into an empty board or when there is no overlap with items. The PEPML model is transformed into a Miro model using BWD. Before the items listed in the Miro model can be added to a Miro board, we must postprocess them. Postprocessors are partially the reverse operations of preprocessors. For instance, the content preprocessor splits the `content` into `name` and `text`, and a postprocessor must again define a value for `content`. Similarly, a postprocessor exists to properly place connector captions based on the properties `startCaption`, `caption` and `endCaption`.

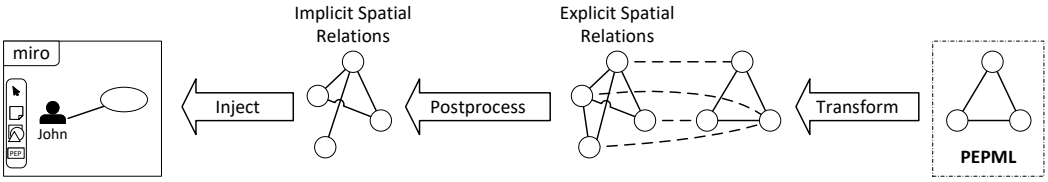

Figure 6.19: Adding entities from a PEPML model to an empty Miro board

Adding Concrete Coordinates

A major part of postprocessing is positioning the items and providing values for `width` and `height`. Miro items newly created based on a PEPML model do not have (x,y) coordinates. Before postprocessing, Miro models explicitly specify spatial relations via relationships like `east` or `south`. A postprocessor must transform these explicit spatial relations into (x,y) coordinates, implicitly defining the same relation. For instance, when a spatial relation defines that a text item shall be south of an image item, the (x,y) coordinates of the respective item must reflect that.

Figure 6.20 shows how to proceed when selected entities shall be added to an existing board. A sub-model of the PEPML model is selected that contains the entity to be added and entities already present on the Miro board. We perform BWD for this sub-model and compare the result to a freshly extracted Miro model. Based on this comparison, we can identify the items that must be added to the board. During postprocessing, these new geometric items get (x, y) coordinates and a width and a height. Due to spatial relations to existing items, an adjustment to the positioning of existing items may be necessary.

Injecting Entities with Relations to Existing ones

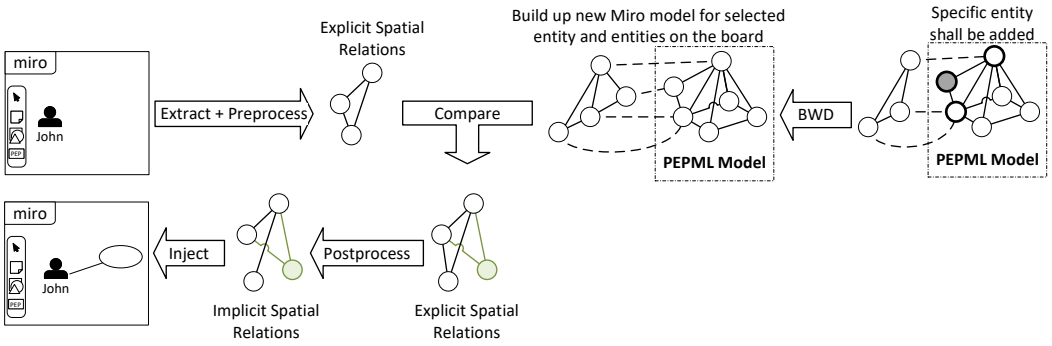

Figure 6.20: Adding entities from a PEPML model to an existing Miro board

We can add single PEPML entities since there is always a 1-to-n mapping between a PEPML entity and the Miro items representing it. In contrast, adding a single visual representation to a PEPML model is not always possible. For instance, an image must always be accompanied by a text item to represent an entity. Solely adding an image or a text item is not possible in PEPML's case.

Injecting Single Entities

6.2.7 Additional Features

In the following, we explain additional features of our tooling. In particular, how decorators are realized technically, and how models can be queried or exported to other formats.

Decorators

We introduced decorators for PEPML entities in Section 5.4.1. Within our tooling, decorators can easily be added by providing a name, a URL to an icon, a location to place the decorator (top-left or top-right) and a criteria defining when the decorator applies. A legend describing the available decorators can be created automatically based on the name and icon of the registered decorators. An example decorator definition is given in Listing 6.1.

Decorator Definitions

Listing 6.1: Example decorator definition

```
1 {
2     "name": "Instructor-Led Session",
3     "placement": "top_left",
4     "url": "https://dwolt.de/pepml/assets/session.svg",
5     "targetType": "PEPML__EducationComponent",
6     "constraint": "NOT t:PEPML__EducationComposite AND isSync = true
          ↪ AND isOnline = false"
7 }
```

Listing 6.2 shows a simplified version of the Cypher query generated for the decorator described in Listing 6.1. Line 1 of the query identifies a pair of corresponding nodes (s,t) from the source and target model. If `targetType` refers to a class for which hierarchical semantics are defined, we must collect inherited values. As explained in Section 4.3.1, education components shall inherit values for certain properties from the education composites to which they belong. Line 2 of the query traverses the hierarchy and collects all ancestor composites. Line 3 of the query makes the source/target node and the inherited values accessible for the remaining part of the query. The variables s and t allow access to the source and target node. For each inherited class property of the defined `targetType`, we define a variable named after this property that contains the potentially inherited value. Hence, all inherited class properties should be accessed directly via their name and not via t since the target node may not define the value directly. In the example, *t.isSync* may be *null*, while *isSync* might contain a different inherited value. Line 4 of the query is the constraint specified in Line 6 of the decorator definition. While the `targetType` of the example decorator definition is set to `EducationComponent`, the constraint ensures that it is not also an `EducationComposite`. The prefix "PEPML__" in front of the class names is necessary to denote that they belong to PEPML's metamodel (see Section 6.2.2). Line 5 returns the `itemId` of items to which the decorator shall be added.

Generating Queries for Decorator Definitions

Listing 6.2: Query generated for the decorator defined in Listing 6.1

```
1 MATCH (s {enamespace: $sourceModel})-[:corr]->(t:
      ↪ PEPML__EducationComponent {enamespace: $targetModel})
2 OPTIONAL MATCH (t)<-[:contains*]-(c:PEPML__EducationComposite {
      ↪ enamespace: $targetModel})
3 WITH s,t, coalesce(t.isSync, head(collect(c.isSync))) as isSync,
      ↪ coalesce(t.isOnline,head(collect(c.isOnline))) as isOnline
4 WHERE NOT t:PEPML__EducationComposite AND isSync = false AND isOnline
      ↪ = false
5 RETURN s.id AS itemId
```

Hierarchical Semantics

Listing 6.2 illustrates the power of graph databases since we can query an arbitrarily long path and collect values from the nodes along this path. It also exemplifies how the hierarchical semantics defined in Section 4.3.1 are realized on a technical level. While decorator definitions are language-specific, the support for decorators in general is language-independent and can be used for other modelling languages as well.

Querying PEPML Models

Once extracted, users can query target models for further information. Since multiple models can be contained in a single Neo4j database, the `enamespace` must be used to focus on a specific model. Furthermore, users must remember that all class names have the name of the metamodel as a prefix, e.g. "PEPML_Entity" (see Section 6.2.2). Manipulating PEPML models, like adding, altering or deleting entities, is possible but requires that changes are injected into the respective Miro board.

Transformation to other Formats

As indicated in the overview in Figure 6.1, a PEPML model could be exported into other formats. This export could again be done via TGGs or custom code transformers. For instance, a PEPML model could be transformed into a Word document describing the programme. Our tooling focuses on extracting the information in a Miro board, generating a PEPML model and injecting changes back into Miro. Transformation into other formats was largely out of scope. However, for testing purposes, we developed a model export to eMSL's model notation. The source code for this transformer is available on GitHub[9].

6.3 Feasibility Study: PEPML

We used PEPML as a case study to test our tooling. In the following, we discuss each of PEPML's viewpoints. Also, we introduce a canvas model for more detailed planning of an education component as an example of a user-defined visual syntax. Finally, we discuss the strengths of our approach and what limitations exist.

[9] https://github.com/dwolters/emsl-model-dumper

6.3.1 PEPML's Dependency Viewpoint

The visual syntax used in PEPML's dependency viewpoint is graph-based and well-suited for our tooling. As preprocessors, the viewpoint requires the content preprocessor to extract entity names based on an item's text content. Moreover, the spatial relation preprocessor has to compute all spatial relations listed in the respective TGG rules. In the case of the dependency viewpoint, this is only the spatial relation between images and the text item below them.

We developed a TGG to transform the preprocessed Miro model into a PEPML instance. The rules of the TGG can be divided into four groups:

(1) An axiom mapping an education programme to a board,
(2) mapping a geometric shape to a PEPML entity,
(3) mapping an image and text item to a PEPML entity, and
(4) mapping connectors or spatial relations to PEPML associations.

Group (1) only contains a single axiom, creating the objects needed as context for objects created by the rules in Groups (2) and (3). Groups (2)-(4) all contain multiple rules with a repetitive style. Chapter 7 presents an approach to generate these rules based on mappings between Miro's and PEPML's metamodel. Representative rules of each group are shown in Figure 7.4 as examples of rules that can be generated based on such mappings. Hence, the generatable rules and further details on these rules are given in that chapter.

In the following, we focus on the rule transforming a rectangle to an education component, which belongs to Group (2). The rule itself is generated based on metamodel mappings using the approach presented in Figure 7.4. However, we discuss this rule since it requires manually specified NACs to not interfere with other rules. A simplified version of this rule is shown in Figure 6.21. In contrast to the full rule, only the name property is mapped, and all other properties are omitted. The NACs are needed to ensure that composites are not mistakenly transformed into regular components since education components and composites share the same visual representation (symbol overload). The source-side NAC ensures that the rectangle representing an education component does not contain another rectangle. The target-side NAC ensures that the `EducationComponent` is not also of type `EducationComposite`. Without these NACs, a correct transformation of components and composites would not be possible.

Different algorithms exist for performing FWD and BWD [KLKS10, Wei22]. The trivial approach arbitrarily takes the first rule that matches and applies it

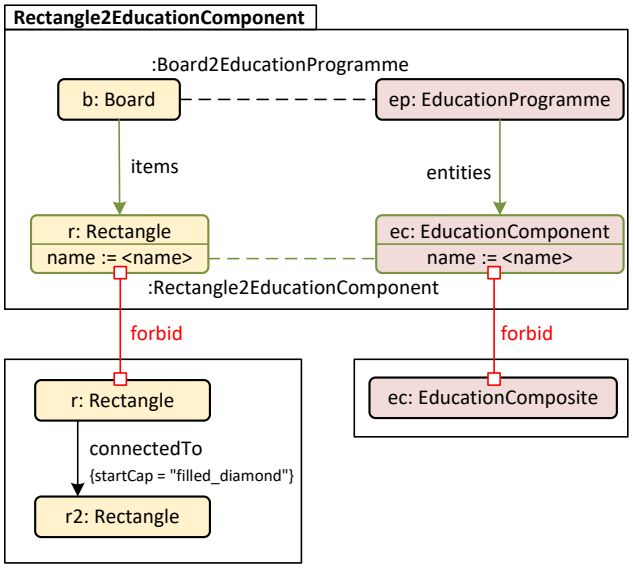

Figure 6.21: Simplified rule to map a rectangle to an education component with two negative application conditions

as often as possible. However, depending on the order of the rule application, more or less of the model may be transformed. Weidmann [Wei22] developed an approach based on integer linear programming (ILP) that tries to transform as much as possible. Yet, the ILP-based approach is significantly slower compared to the trivial approach. For our purposes, it is sufficient to define a rule application order for the transformation to work correctly. The highest-ranked matching rule is always applied next. Users have to ensure that rules, which are specific subcases of other rules, are ranked higher than the more generic rules. For instance, our rule to translate the generic `Relation` association between entities would apply to all connectors in a Miro model. More specific rules, like the translation of the more specific `involvedIn` relation, must be ranked higher in the rule order.

Based on rules like those presented in Figures 6.21 and 7.4, we can transform a Miro model into a PEPML model and vice versa. As mentioned, when we translate a PEPML model into a Miro model, layout information has to be added before the model can be injected into the board. Layouting graphs is not the focus of this thesis. Yet, to show general feasibility, we implemented a rudimentary force-directed layout algorithm based on the d3-force library[10]. Our layout algorithm is intended for the dependency viewpoint and is limited to handling the spatial relation `south`, which is required to position text items beneath the respective images.

[10] https://github.com/d3/d3-force

6.3.2 PEPML's Temporal Viewpoint

The temporal structure viewpoint solely relies on rectangles and focuses on education components and composites. Hence, the TGG contains far fewer rules than the one for the dependency viewpoint. We again need a similar rule to the one shown in Figure 6.21. The only difference is that the `connectedTo` link in the source-side NAC has to be a `contains` link for the temporal viewpoint. All other rules can be described using our mapping language described in Chapter 7. Nonetheless, the rules to handle temporal ordering inside and outside of time-planning composites require deeper explanation.

Before we can explain how to handle the temporal ordering, we need to explain how time-planning composites are identified. Since they have the same visual depiction as education composites, it is difficult to distinguish them with TGGs. We decided to develop a special preprocessor to identify time-planning composites in the visual representation based on the timescale they contain. NACs would not have been an alternative because they are not expressive enough to identify arbitrary timescales. Based on our preprocessor, each rectangle containing a timescale gets the custom property `meta_type` with the value "TimePlanning". In the TGG, we exploit this custom property to transform only those items having this property value to time-planning composites. Moreover, the rule for identifying time-planning composites must be ranked higher than the rules identifying regular education composites. The preprocessor for time-planning composites also extracts information about the start time and duration of the components contained in the composite and adds this information to the rectangles representing the components.

We use the spatial relations `east` and `south` to extract temporal relations. Outside of time-planning composites, an `east` relationship matches a `next` association, and a `south` relationship matches a `parallel to` association. However, for components inside time-planning composites, `east` matches `parallel to` and `south` matches `next`. The rules in Figure 6.22 show how this difference can be addressed. The rule on the left creates education components that are part of time-planning composites. Instead of using `:Rectangle2EducationComponent` as a correspondence type, this rule uses `:Rect2ECinTPC`. The rules on the right in Figure 6.22 exploit these correspondence types to transform the `south` links differently. The top-right rule handles education components inside and the bottom-right outside of time-planning composites.

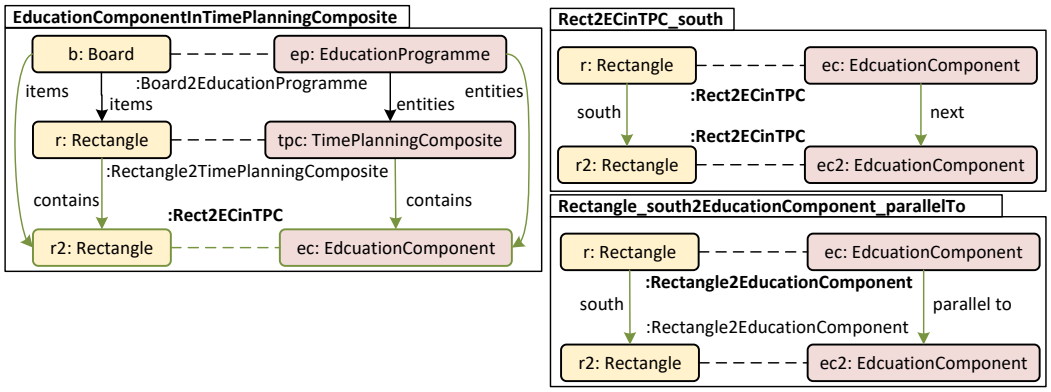

Figure 6.22: Example rules to handle temporal orders inside and outside of time-planning composites

Figure 6.23 depicts the abstract syntax graph obtained by transforming the model shown in Figure 5.6. The graph shows that with the time-planning pre-processor, all time-planning composites are correctly identified. Furthermore, the rules presented in Figure 6.22 properly handle the different interpretations of spatial relations inside and outside of time-planning composites. Also, the rule application order must rank the rule to identify time-planning composites higher than the one to identify regular education composites.

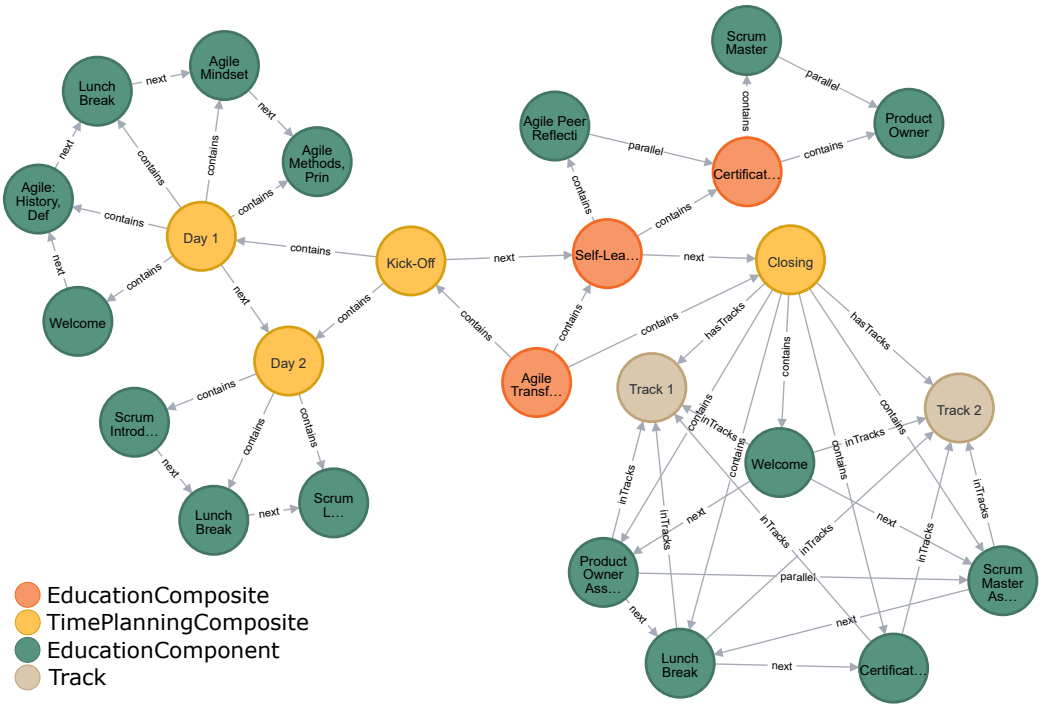

Figure 6.23: Visualization of the graph stored in the Neo4j database for the temporal structure view of the running example

6.3.3 Custom Visual Syntax: Activity Canvas

Canvas models, which provide a structure to cluster sticky notes, are a popular way of modelling on online whiteboards. The Business Model Canvas (BMC) [OP10] is one of the most well-known canvas models, and Miro provides a template to add a BMC to a board quickly. While the BMC is very popular, we decided against using it as an example of a canvas model. The BMC contains nine different areas to place sticky notes, which would all be handled in the same fashion by a TGG. As an example, the BMC would be more repetitive than challenging. Instead, we introduce a custom canvas model called *Activity Canvas*. It shows how such areas to structure sticky notes can be defined and how they can be combined with more advanced visual syntax concepts like time-based planning.

The Activity Canvas is shown in Figure 6.24. The topmost rectangle specifies the education component's name for which the activities shall be defined. On the left, the activities are defined as rectangles, and a time scale indicates the length of the activities. The right side contains an area to define learning outcomes and one to define required resources.

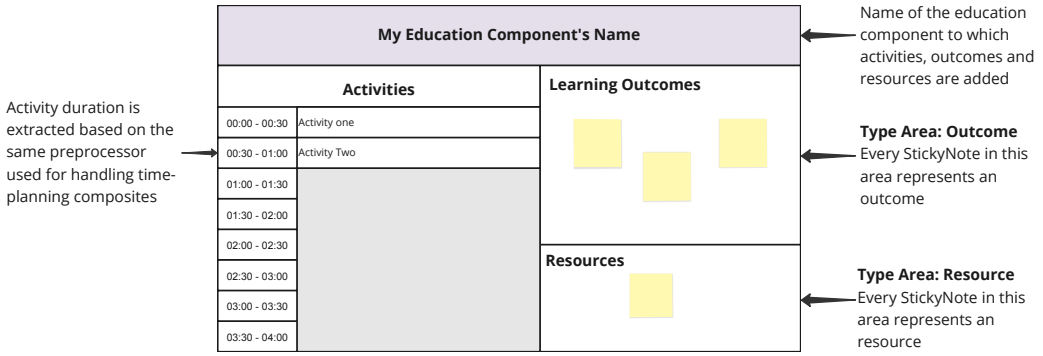

Figure 6.24: Custom visual syntax: Activity Canvas

For extracting a model based on the Activity Canvas, we require the content and the time-planning preprocessor. The latter automatically identifies the rectangle containing the timescale as a time-planning composite and assigns the custom property `meta_type = "TimePlanning"`. Furthermore, it assigns the proper durations to the activity rectangles. The rectangles labelled "Learning Outcomes" and "Resources " describe type areas, which means that every sticky note contained by this area represents an instance of a specific PEPML type. Identifying the type areas for the classes `Outcome` and `Resource` could be done based on their text content. For instance, every sticky note contained

by the rectangle labelled "Learning Outcomes" could be transformed into an instance of `Outcome`. While this works, it is prone to errors, e.g. when the rectangle has a different name like "*Desired* Learning Outcomes", the TGG rule would no longer apply. Within the *Properties* tab of our Miro app, users can specify that a geometric item represents a type area (see the right tab in Figure 6.2). An item defined as a type area gets a custom property called `meta_area` with the name of the selected type as a value. This custom property can then be exploited in the TGG rules to transform items in that area into the correct type.

Figure 6.25 shows important TGG rules to transform an Activity Canvas into a PEPML model. As described by the top-left rule, the rectangle containing the name of the education component shall have the rectangle with a `meta_type = "TimePlanning"` to its south. Using this rectangle as context ensures that not every rectangle is transformed into an education component. The `meta_type` information is added by the time-planning preprocessor. The top-right rule maps every rectangle in the time-planning rectangle to an activity. Symbol overload, i.e. activities are also represented as rectangles, is handled based on the context of the items.

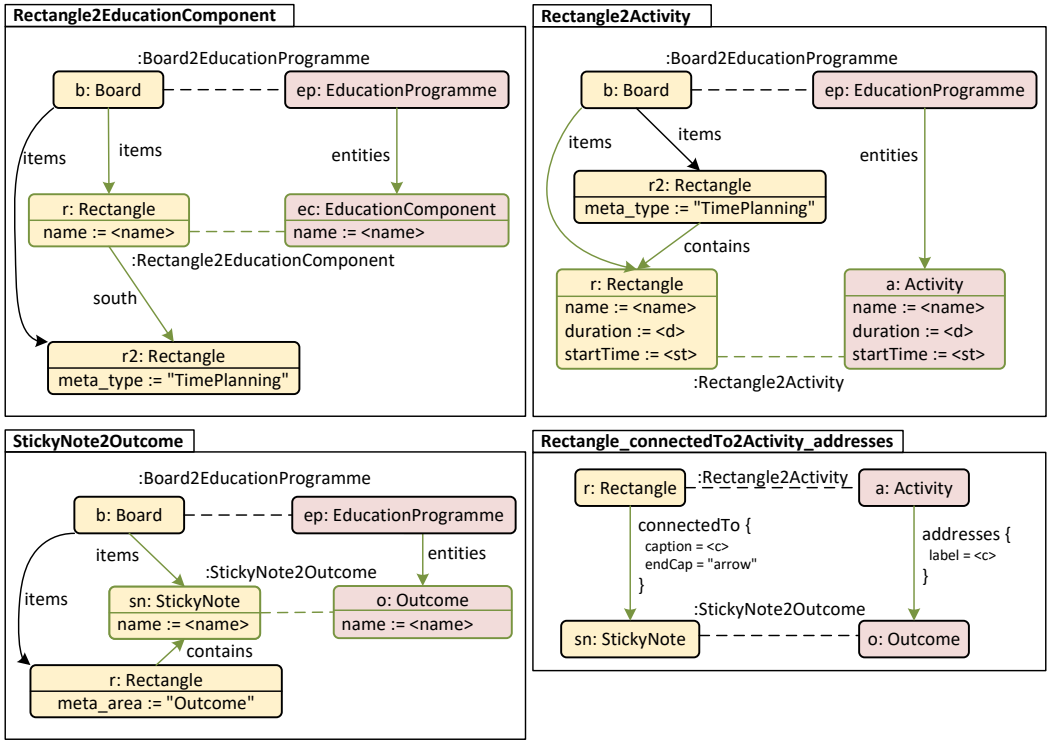

Figure 6.25: Partial TGG for transforming an Activity Canvas

The bottom-left rule shows how the rectangle with `meta_area` property is used as context to transform sticky notes into instances of `Outcome`. The bottom-right rule describes that if a rectangle representing an activity is connected to a sticky note representing an outcome, it means that this activity addresses this outcome. The full TGG contains analogous rules to handle resources. Analogous rules to the bottom-right would also exist to add activities, resources and outcomes to the education component.

In addition to the rules shown in Figure 6.25, we need rules that connect all activities, outcomes and resources to the respective components. Figure 6.26 depicts such a rule for connecting outcomes to the education component. This is a so-called *source-idle* rule since it does not affect any objects on the source side. The NAC ensures that the link is only added once. Without the NAC, the rule would apply arbitrarily often. To avoid specifying such NACs, we added the option to eNeo.js to specify how often rules are applied. The NAC in Figure 6.26 can be removed if we specify that the rule can only be applied once. If it is just applied once, it should be applied after all other transformations have been processed. Meaning the rule should be ranked lowest in the rule application order.

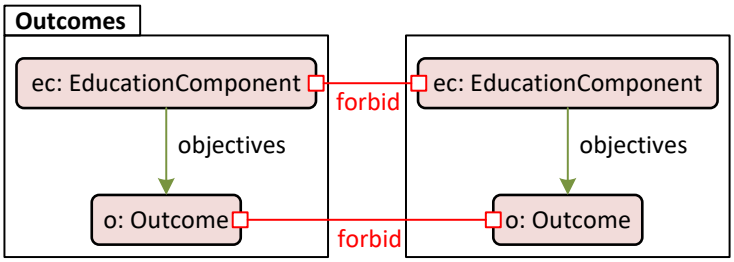

Figure 6.26: Source-idle TGG rule for adding outcomes to education components

Please note that an Activity Canvas represents an excerpt of a model of an education programme. Additional information can be provided on other boards or in other tools and can be merged based on the approach presented in Section 6.2.5 into a single PEPML model.

6.3.4 Discussion & Limitations

For the moment, we only allow one visual syntax per board. Combining different visual syntaxes on the same board without a clear separation is difficult.If the visual syntaxes overlap in their use of visual representation, a proper extraction cannot be ensured. Frames can be used to separate different

models on one board. One frame's content can then be transformed based on different rules than another frame's content. While frame-based separation is technically possible, it is not implemented in PEPML's tool support yet.

When using online whiteboards for visual modelling, the visual syntax is limited to what can be represented on a board. If formal models shall be extracted, the syntax is further limited to the representation that can be extracted programmatically. Miro supports a wide variety of shapes, including shapes specific to languages like the UML or BPMN. Yet, programmatic access is only available for basic geometric shapes. The development roadmap indicates that access to language-specific shapes will be available in the future.

The absence of the grouping functionality from the WebSDK and API is more problematic. Complex board items could be created by grouping basic geometric shapes. However, without the possibility of retrieving information about grouped items programmatically, we cannot easily identify a group of items forming a complex board item. Instead, we have to exploit spatial relations to do so. In the case of PEPML, we use the `south` relation to identify the text item belonging to an image. If users decide to place the text item above the image, the extraction would not work. Partially, we can address this with a more flexible TGG, but combining the image and the text item into a grouped item is preferable. We would not only know that these items belong together, but users could also jointly move and resize them.

Decorators are an easy solution to provide additional information about an item without requiring a completely separate visual representation. Unfortunately, using decorators in online whiteboards has a few limitations. Decorators can be placed on the item to which they belong, but they are not attached to it. Consequently, if users move an item, the decorator does not move along by default. Grouping would partially solve this problem, but it has the drawback that resizing an item might lead to a disproportional decorator. To avoid this, a fixation of the aspect ratio of an item would be needed or the ability to place items relative to other items. Such a relative placement already exists for frames. Child items of frames are not placed relative to the canvas' centre but rather in the top-left corner of the frame. The API documentation suggests that relative placement may be available for other item types in the future.

Moody [Moo09] encourages that there should be a one-to-one match between metamodel concepts and their visual representation. We have already discussed symbol excess, deficit, redundancy and overload in Section 5.3.3. When using TGGs for model transformation, a symbol excess would simply

not lead to any element in the target language. Symbol redundancy requires that TGG rules for each of the representations exist. It makes the TGG more complex but can be handled. A symbol deficit is unproblematic unless the elements not represented visually are required as context elements in the abstract syntax. If so, this has to be considered when defining TGG. Symbol overload, however, can be challenging. PEPML defines visual representations for its main classes. Subclasses of these classes use the same visual representation, e.g. an education component and an education composite are both represented as a rectangle. As mentioned, we use NACs to properly distinguish between a component and a composite. Using a different visual representation would be easier, but for the moment, we are limited to basic geometric shapes. Symbol overload across different viewpoints can be problematic for users, but it is technically easier to solve since we have separate TGG rules for each viewpoint. In the case of the Activity Canvas, we argue that using rectangles for activities instead of education components is manageable for users since we use a completely different style of visual syntax, i.e. a canvas model. Our concept of pretyping, either through preprocessors or based on user input, greatly helps in dealing with symbol overload.

Our approach of using TGG in combination with a set of reusable preprocessors is sufficient for a wide range of scenarios. The dependency viewpoint shows that graph-based visual notations can be used with our tooling. The temporal structure viewpoint shows we can also support a block-based visual syntax by exploiting spatial relations. The Activity Canvas illustrates that it is possible to use canvas models as well.

The current set of pre-/postprocessors is already very powerful, but further ones will be necessary. For instance, more generic pre-/postprocessors to split and merge text content. Furthermore, we only added a simple layout algorithm for postprocessing models from a dependency viewpoint. Especially for graph-based visual syntaxes, a wide range of layout algorithms exist that could be incorporated in the future [GFV13]. PEPML's temporal structure viewpoint, however, requires a specially designed layout algorithm.

Styling information is largely lost during the transformation between a Miro model and a PEPML model. Default styling could be incorporated into a TGG, or pre-/postprocessors could be developed that provide more fine-grained access to styling information. For instance, properties could be added to define the styling for the first paragraph of a `ContentItem`.

We chose eMSL as a language to describe TGGs, and with eNeo.js, we provide our own TGG engine. Due to the usage of eMSL, we largely stay

compatible with eMoflon::Neo [WA21]. Yet, eMoflon::Neo currently neither operationalizes property constraints between associations nor NACs. Both are used for PEPML and supported by eNeo.js. Furthermore, our flexible editing of TGGs at runtime is impossible with eMoflon::Neo since it operationalizes TGGs at development time. Compared to eNeo.js, eMoflon::Neo provides additional operationalizations, e.g. fault-tolerant model synchronization or a more advanced FWD/BWD based on ILP [Wei22]. The latter could remove the need to specify an explicit rule order but is also computationally far more complex. Hence, it would be slower than using a greedy-based FWD/BWD with an explicit rule order. Another candidate TGG engine would have been eMoflon::IBEX [WARV19] since it is more versatile in describing property constraints, e.g. it supports arithmetic expressions on properties. However, eMoflon::IBEX also operationalizes TGGs at development time and is based on EMF, which makes the metamodels more strict and static compared to model transformations based on a schemaless graph database.

Since we can manipulate TGGs at runtime, it is possible to alter the model extraction and handle cases where users diverge from the intended visual syntax. Our pretyping approach can further increase flexibility, i.e. users could manually type items when they diverge from the visual syntax of the modelling language. If users wish to extend PEPML or another target modelling language, they can do so at runtime by manipulating the target metamodel. For instance, they could add further subclasses of `Person` or `EducationComponent`. They can stick to the default visual representation of the superclasses or add rules to the TGG, mapping the new classes to their own visual representations.

Our tooling can be used for other visual languages as well, e.g. UML Use Case Diagrams. To do so, a metamodel and corresponding TGG have to be specified. In the next chapter, we illustrate how the effort for the latter is reduced. Languages with complex visual elements, like classes in UML Class Diagrams, can be tricky. A class is typically depicted by a rectangle with three compartments for the class' name, its properties and its operations. Miro offers a dedicated stencil for this class representation, but as mentioned, it is not exposed via the API/SDK yet. Hence, users have to resort to using a rectangle for each compartment and using the `south` relation to connect compartments. While this works, it could be more user-friendly.

6.4 Summary

In this chapter, we explained how we support visual modelling on the online whiteboard Miro. We provide a Miro app, allowing the use of bidirectional model transformation based on TGGs to transform board content into a formal model and vice versa (cf. TR1). Preprocessors enrich the Miro model extracted from a board before it is transformed, which allows covering a broader range of transformations with TGGs. Similarly, when transforming a formal model back into a Miro model, we use postprocessors to add information like (x,y) coordinates before this model is injected into a board. The PEPML-specific parts of our tooling are technically isolated, and the tooling can be reused for other languages (cf. TR2). Within our tooling, the metamodel of the target language, as well as the TGG to extract it, can be manipulated at runtime, allowing custom alterations to handle cases where the language is extended, or users diverge from the standard visual syntax (cf. TR6). We discussed different operationalizations of TGGs, e.g. FWD and model synchronization, and explained their benefits and limitations. In our tooling, we rely on a combination of FWD and model merging to support multiple viewpoints (cf. TR3) and deal with complex models (cf. TR4). Consequently, we do not only allow collaborative design on a single but across multiple boards. Syntactical problems can be identified at various stages (cf. TR5), i.e. during the extraction of the board content, the preprocessing of the Miro model, the model transformation or the final model merge. If everything is successful, the formal model is available in a Neo4j graph database and can be queried by other applications or transformed into other formats (cf. TR7). Overall, our tooling shows that online whiteboards can be used as primary visual modelling tools with the possibility for syntax validation and proper extraction of formal models (cf. RQ2).

We used PEPML as a case study to test our tooling. We showed that the relation of PEPML to visual representations on Miro boards can be expressed via TGGs. Moreover, we illustrated that TGGs, in combination with our preprocessors, are powerful enough to cover different styles of visual syntaxes, e.g. graph-based, block-based or canvas-based. In the next chapter, we present a mapping language that allows specifying mappings between two metamodels in a succinct format. Based on these mappings and the metamodels, we can then generate comprehensive TGGs. As we explain in the next chapter, this mapping language is particularly useful for our purposes of mapping a modelling language to its visual representation.

Chapter 7

Generating Triple Graph Grammars

In the last chapter, we explained that we use TGGs to define the relation between PEPML and its Miro-based visual syntax. We then use these TGGs to extract PEPML models from Miro boards. Unfortunately, the TGGs needed for this use case are complex, repetitive and difficult to maintain when specified manually. This chapter introduces GenTGG, a language to define succinct mappings of classes, properties and associations of two metamodels. Based on these mappings, we can then generate TGG rules. With GenTGG, we can significantly reduce the effort required to specify and maintain the TGGs needed within this thesis. We explain GenTGG and illustrate its use with an excerpt of our mapping for PEPML's dependency viewpoint syntax. We show that GenTGG is not limited to our use case and show its expressiveness by using it to describe TGGs used as test cases for eMoflon::Neo [WA21].

Section 7.1 introduces requirements, gives an overview of GenTGG and discusses related work. Section 7.2 introduces the GenTGG mapping language and illustrates its use via an example. Section 7.3 presents the process to generate a TGG based on mappings specified using our language. Section 7.4 shows our evaluation to determine GenTGG's expressiveness, usefulness and limitations.

7.1 Overview

This section provides an overview of our approach to generating TGGs based on metamodel mappings. It covers the requirements for our approach, a solution overview and related work.

7.1.1 Requirements

The requirements for our approach for generating TGGs are based on our use case of leveraging TGGs for extracting PEPML models from Miro boards. All TGG rules required for PEPML model extraction can be divided into the four groups we mentioned in Section 6.3.1:

Groups of TGG Rules Required for PEPML

(1) An axiom mapping an education programme to a board,
(2) mapping a geometric shape to a PEPML entity,
(3) mapping an image and text item to a PEPML entity, and
(4) mapping connectors or spatial relations to PEPML associations.

Mappings Informally Visualized

After writing the first rules for this TGG manually, we noticed that we could generate these rules if we knew which metamodel elements relate to one another and by applying certain heuristics. Figure 7.1 informally shows examples of the metamodel mappings required for rules of all four groups and serves as a running example for this chapter. A mapping encodes a semantic relation between two classes, properties or associations.

Changes to the Metamodels

Please note that the metamodels in Figure 7.1 are specified using eMSL and slightly differ from those shown in the previous chapters: eMSL requires all associations to be directed and to have explicit names to enable TGG-based model transformations. To simplify the example, we denote Miro's association class `Connector` as the `connectedTo` association and list the `south` association explicitly between `Image` and `Text`. The latter is originally a reflexive association of their common superclass `GeometricItem` (see Figure 6.8).

To reduce the effort of creating the TGG for PEPML model extraction, we require an approach to specify such mappings and generate the required TGG rules. We define the following Generation Requirements (GR):

GR1 **Mapping of Source and Target Class**: *The approach shall allow us to specify mappings between a class of the source and a class of the target metamodel.*

Class mappings are the basis of all rules required for our purposes, e.g. mapping `Circle` to `Topic`.

GR2 **Mapping of Properties**: *The approach shall allow us to specify mappings between a property of the source class and a property of the target class.*

When classes are mapped, (selected) properties in these classes often have to be mapped as well. We see this in Figure 7.1 for the properties of `Circle` and `Topic`.

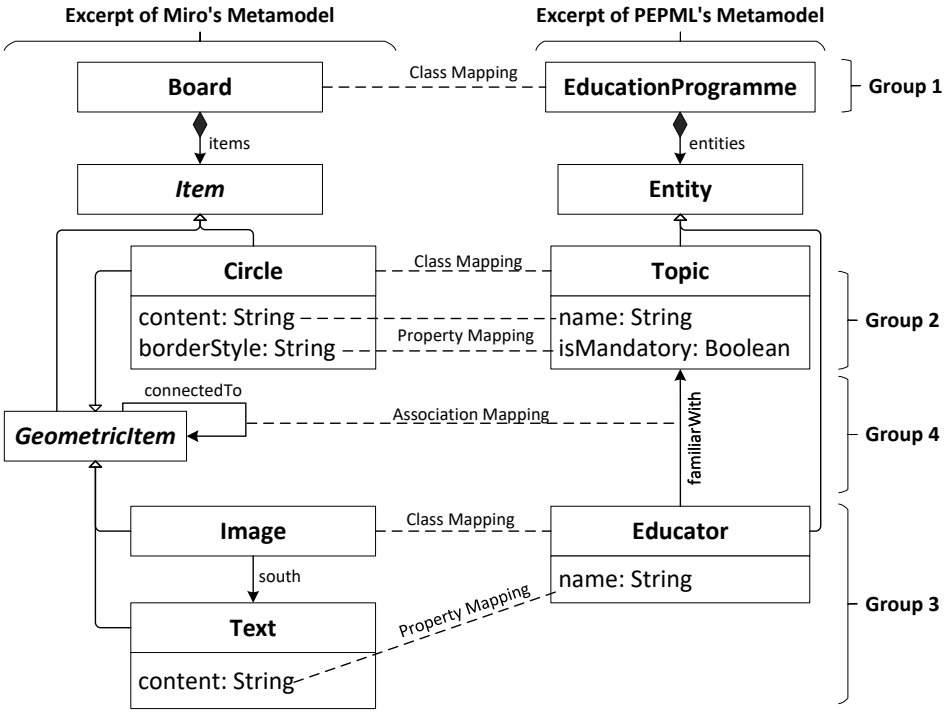

Figure 7.1: Excerpt of Miro's and PEPML's metamodels specified using eMSL and connected via metamodel mappings

GR3 **Mapping of Properties of Associated Classes**: *The approach shall allow us to define property mappings that include a property owned by a class associated with a source or target class.*

A class may not own all properties relevant to the class to which it is mapped. Sometimes, properties of associated classes have to be mapped to properties on the other side. We see an example of this in Figure 7.1, where the property `content` of `Text` is mapped to the property `name` of `Educator`, even though the class mapping is between `Image` and `Educator`. Since images cannot contain editable text content in Miro, we must use text items instead to specify an entity's name.

GR4 **Handle Property Type Differences**: *The approach shall allow us to specify how to deal with mapped properties with a different type.*

When two properties with the same data type are mapped, we can directly copy property values between instances of these classes having these properties. In Figure 7.1, we have an example where the property `borderStyle` of type `String` is mapped to property `isMandatory` of type `boolean`. Values of these properties are incompatible and it has to be specifiable how the incompatibilities shall be resolved.

GR5 **Mapping of Associations**: *The approach shall allow us to specify mappings between an association of the source and one of the target class.*

When two classes are mapped, their associations often relate to one another as well. For instance, Miro allows us to connect geometric items. We use such connectors to express relations between PEPML entities.

GR6 **Deduce Relevant Association Targets**: *The approach shall only consider those targets of mapped associations for which a class mapping exists.*

Figure 7.1 shows an association mapping between `connectedTo` and `familiarWith`. While the latter is between just two classes, the former is a reflexive association on an abstract class. As a result, `connectedTo` can be used between various subclasses of `GeometricItem`. For the association `familiarWith`, only `connectedTo` from `Image` to `Circle` is relevant. This can be deduced based on the class mappings. If multiple suitable class mappings exist, multiple TGG rules are necessary.

GR7 **Specification of Variations**: *The approach shall allow us to specify if an object of a class has to be created, if it has to exist or if variations for both cases are needed.*

Slight variations of TGG rules can be necessary, e.g. one variant where a particular object is created and another variant where it is assumed that the object exists. Such variations are needed when a parent object needs to be created in conjunction with the first child and all remaining child elements are added to this parent object.

GR8 **Deduce Context based on Metamodels**: *The approach shall consider metamodel information when generating TGG rules to add relevant context elements automatically.*

Aside from axioms, TGG rules require a context for the objects that shall be created. This context defines which objects must exist on the source and target side to create the new objects. Based on the compositions and multiplicities, we can deduce which objects must be present.

GR9 **Generation of Constraints for Static Semantics**: *The approach shall automatically generate constraints that validate static semantics specified by multiplicities in the source and target metamodels.*

Neo4j is a schemaless database and cannot enforce static semantics specified by multiplicities. eMSL allows defining constraints that must hold after transformation, which can enforce static semantics.

7.1.2 Solution Overview

As depicted in Figure 7.2, GenTGG consists of a mapping language and a generator. The GenTGG language addresses multiple requirements by allowing to specify mappings between classes (GR1), properties (GR2 and GR3), property values (GR4) and associations (GR5). The GenTGG generator takes the mappings and the metamodels as input to generate a TGG. During the generation process, the mappings are extended to TGG rules by taking metamodel information into account (GR6 and GR8) and applying heuristics to specify which objects must be created and which should exist. Within mappings, users can add modifiers to create multiple rule variants (GR7). Furthermore, multiplicities defined in metamodels are not only used to extend mappings to rules but can also be converted into constraints to enforce static semantics (GR9). The generated TGG is then used for model transformations. If the generated TGG cannot cover all relevant scenarios, manually written TGG rules can be added.

Mapping Language + Generator

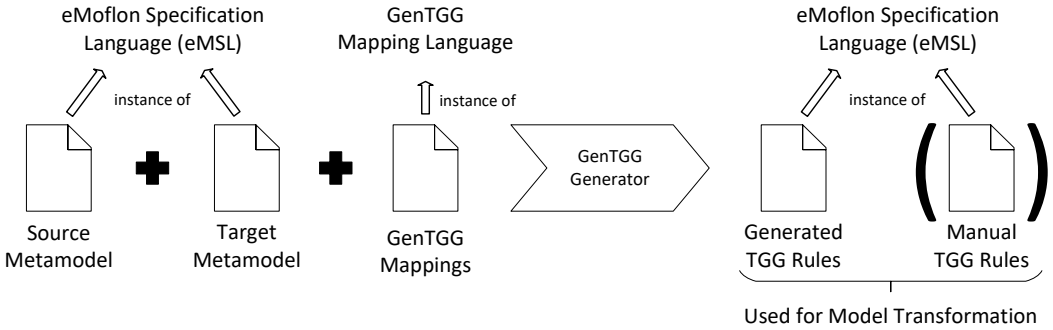

Figure 7.2: Overview of GenTGG

The metamodels and the generated TGGs are specified within eMSL. We chose eMSL since it is a lightweight textual language to define metamodels and TGGs. Furthermore, with eMoflon::Neo [WA21] and our eNeo.js, two engines exist to execute TGGs specified based on eMSL.

Using eMSL

7.1.3 Related Work

Over 50 years ago, Wijngaarden [VW74] used a two-level approach to generate the grammar for the programming language ALGOL 68. Writing the grammar for ALGOL 68 would have been very verbose and difficult to manage. The two-level grammar approach allowed for a simpler representation based on which the actual grammar could be generated. Similarly, GenTGG uses a compact notation to generate larger, often very repetitive TGGs.

Two-Level Grammar

Triple Patterns

Lara et al. [LGB07] present an approach to generate TGG rules based on normal graph grammar rules combined with so-called Triple Patterns. Operationalized TGG rules can be obtained by applying triple patterns on normal graph grammar rules. Like our approach, they also take metamodel information into account, e.g. class inheritance. GenTGG does not depend on existing graph grammar rules but enables users to specify mappings between classes, properties and associations, which are then extended to TGG rules.

Traceability Mappings

Diskin et al. [DGC17] use traceability mappings to specify model-to-model transformations. They focus on ATL as a transformation language, which is not suited for our purposes since it is unidirectional. Furthermore, we take metamodel information into account when generating TGG rules.

Correspondence Metamodel

A correspondence metamodel is part of the TGG formalism [Sch94], which defines correspondence types between classes. GenTGG also defines mappings between classes, which is equivalent to defining correspondence types. However, GenTGG goes beyond classes and allows mappings between properties and associations. Property constraints in TGGs can be used to express property mappings but must be defined for each rule. Relations between associations can also only be defined on the level of TGG rules and they are even less explicit since multiple associations can be created on each side. In such cases, it is unclear if there is a pairwise relation between associations of the source and target metamodel.

Model Transformation By Example

Instead of directly writing model transformations, several approaches allow the generation of transformation rules based on examples [KLR+12]. For instance, Arifulina [Ari16] uses a genetic algorithm that can generate model transformations based on pairs of mapped models. Such approaches enable users without model transformation expertise to create model transformations. However, a set of carefully selected/constructed examples is needed for such approaches to work correctly. In contrast, GenTGG relies on succinct mappings between classes, properties and associations, which can be easier to manage than a large set of examples. Both for GenTGG and example-based approaches, model transformation expertise becomes necessary when the generated transformations do not fully cover all intended transformation scenarios. For example-based approaches, this can mean that the set of examples needs to be extended or manual, rules must be added. Similarly, if a TGG generated based on GenTGG mappings does not cover all scenarios, additional mappings may be needed or the TGG has to be supplemented with manual rules. We explain in this chapter which types of TGG rules can be generated based on GenTGG mappings and which not.

7.2 Specifying Metamodel Mappings

GenTGG uses a set of mappings between the source and target metamodels to generate a TGG. A GenTGG mapping begins with a class mapping followed by property and association mappings. Listing 7.1 shows three example mappings starting in Lines 1, 6, and 11. The source metamodel for this example is the adapted Miro metamodel shown in Figure 6.8. The target metamodel is PEPML's metamodel presented in Chapter 4. Figure 7.1 depicts the relevant excerpts of both metamodels and an informal visualization of the mappings.

GenTGG Mappings

A class mapping is a one-to-one relation between a source and a target class and addresses GR1. For instance, Line 1 maps Miro's `Circle` class to PEPML's `Topic` class. We refer to the classes specified in the class mapping as origin classes of the respective GenTGG mapping.

Class Mapping

For each class in a class mapping, a pattern can be defined in curly brackets after the class name. A pattern is optional and describes the required conditions for the mapping to apply on the source or target side. In contrast to patterns in regular TGGs, the patterns in our language are restricted to the properties and associations of the respective class, and we only support the definition of patterns with a distance of, at most, one association to the respective source/target class. As we see in Section 7.3, this can already be very powerful when exploiting information from the metamodels in parallel. Furthermore, the limitation enables a compact textual notation. Even though associations are directed, we support using both incoming and outgoing associations. Examples of associations in patterns and usage of incoming associations are provided in Section 7.4. The pattern in Line 6 specifies that an image's title indicates if an image represents an educator.

Pattern

Listing 7.1: Excerpt of the GenTGG mappings between Miro's to PEPML's metamodel

```
1   Circle <=Circle2Topic=> Topic
2   .name <=> .name
3   .borderStyle <=> .isMandatory
4   ["normal"=true","dashed"=false]
5
6   Image{.title="educator"} <=> Educator
7   -south->Text.name <=> .name
8   -connectedTo {.endCap="arrow"} <=> -familiarWith
9   (.caption <=> .label)
10
11  Board <=> EducationProgramme
12  -items <=> -entities
```

Operator Operationalization

In a pattern, we only allow the "=" operator because this is the only operator that eMoflon::Neo [WA21] can operationalize for all operations, like FWD, BWD or model synchronization. Other operators are more difficult to operationalize for all possible operations. Assume we would define in a pattern on the source side that a property value shall be unequal ("≠") to a particular value. This can be operationalized for FWD since the source side would exist, and we have a concrete value for the comparison. However, these property constraints cannot be operationalized for BWD since it is unclear what value should be used for this property when generating the value on the source side. There are different options to deal with this problem: The property constraints could be ignored for BWD, or a random unequal value could be generated, or, as in the case of eMoflon::Neo, the operator stays unsupported. To keep compatibility with eMoflon::Neo, we decided against allowing any other operator within patterns.

Value Alternatives

Even though it is limited to the "=" operator, GenTGG allows defining value alternatives. These value alternatives can be used to specify that a property must have one out of a selection of values. For instance, we could replace the pattern for the class `Image` in Line 6 with *.title* = ["*Educator*","*Teacher*"] if we want to allow different titles to classify an image as an educator.

Association and Property Mappings are relative to Class Mappings

After a class mapping, property and association mappings can follow, which are always relative to the class mapping. Hence, they can only refer to properties and associations of the source/target class. In contrast to patterns, those mappings define relations between the source and the target side.

Property Mappings

The simplest case of a property mapping is a direct mapping of a property of the source class to a property of the target class (cf. GR2). Lines 2 and 3 of Listing 7.1 show two property mappings, defining that a circle's name[1] reflects the topic's name and that the border style indicates if it is a mandatory topic.

Value Mapping

Properties can directly be mapped if the data type is equal, e.g. mapping a string of the source metamodel to a string of the target metamodel (see Line 2). If property types differ, GenTGG allows the specification of a value mapping (cf. GR4). The properties mapped in Line 3 are of a different type: `borderStyle` is a string, and `isMandatory` is a boolean. Line 4 shows a value mapping specifying that the value "dashed" is mapped to the value "false" and that "normal" is mapped to "true". Hence, mandatory topics have a normal (solid) border, and optional ones have a dashed border. Such a mapping can also be added when properties of the same data type are mapped. In that case, the mapping only applies to those values.

[1] A circle's name is extracted by the content preprocessor explained in Section 6.2.4.

A property can also be mapped to a property of a class referenced via an association, which addresses GR3. Again, we support both incoming and outgoing associations for referencing other classes. The mapping starting in Line 6 maps Miro's `Image` to the PEPML's `Educator` class. In Line 7, the `name` property of class `Educator` is not directly mapped to a property of the class `Image`, but instead to the name property of a Text item located south of the image. Such a mapping results in a 2-to-1 mapping since Miro's `Image` and `Text` classes are mapped to PEPML's `Educator` class. Also, Line 7 illustrates that patterns cannot only be defined for classes but for associations as well.

Property to Referenced Property Mapping

Lines 8 and 12 contain examples of association mappings, which are required by GR5. The target class of these associations can be expressed explicitly, or as in the case of the example, the target class is not listed and later extracted from the respective source or target metamodel (cf. GR6). All association mappings referring to at least one composition are called composition mappings. Line 12 is a composition mapping since it refers to two compositions. The next section covers the TGG generation and explains that association and composition mappings are handled differently from other mappings and result in separate rules.

Association Mapping

7.3 Generating a TGG

This subsection explains how a TGG is generated based on a set of mappings. An overview of the process to generate a TGG is shown in Figure 7.3. Each step of the process is labelled with a circled letter, which we use to refer to individual steps. Circled letters with a white background refer to the main and with a black background to the sub-process. We chose letters to label the steps to set the process in relation to the mapping shown in Listing 7.1, which we use to illustrate the process in action. Hence, all line numbers mentioned in this section refer to that listing.

Generation Process

We split this section into four subsections: Section 7.3.1 covers the Steps Ⓐ to Ⓒ of the main process. Section 7.3.2 covers the sub-process detailing Ⓓ. Section 7.3.3 covers the remaining steps, Ⓔ to Ⓗ. Section 7.3.4 covers advanced features to tweak the generation process.

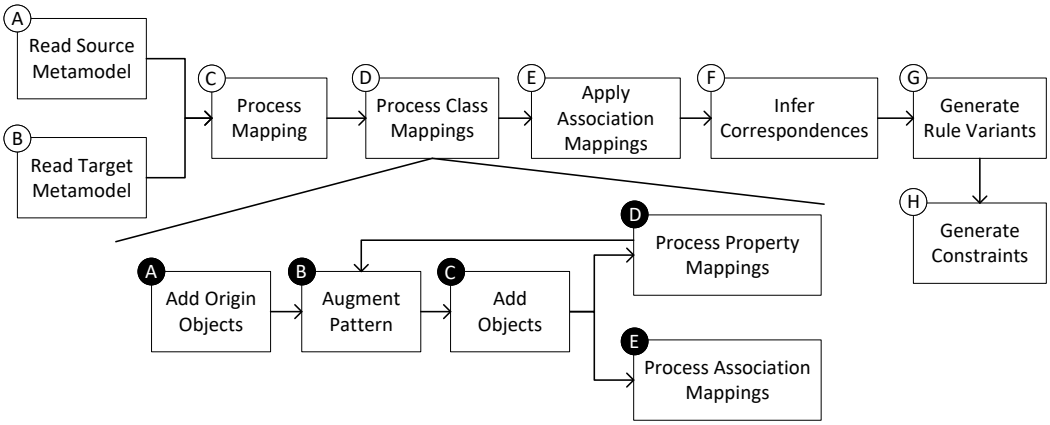

Figure 7.3: Process of generating a TGG based on GenTGG class mappings

7.3.1 Validating Metamodels and Mappings and Deriving Additional Information

Reading Metamodels and Mappings

The generation of a TGG based on GenTGG mappings starts by processing the source and target metamodel (cf. Ⓐ and Ⓑ). Both metamodels are checked syntactically and transformed into an internal data structure that allows querying the respective metamodel for further information. This is needed to satisfy GR8. If both metamodels are syntactically correct, the GenTGG mapping is processed and transformed into an internal data structure (cf. Ⓒ).

Adding Implicit Information

As part of this transformation, it is validated that all classes, properties, and associations mentioned in the mapping exist in the metamodels. Implicit information, like classes at the other end of associations (cf. GR6), is added explicitly to the internal data structure using the information the respective metamodel provides.

7.3.2 Processing Mappings

Processing Mappings

The subprocess detailing Step Ⓓ is executed for each mapping. The execution of the subprocess results in at least one TGG rule for each mapping. The rules generated based on the mappings shown in Listing 7.1 are depicted in Figure 7.4. While we have three mappings, the resulting TGG has five rules due to association and value mappings. Value alternatives and modifiers, which are explained later, also lead to the creation of multiple rules per mapping. We use the process step labels to indicate which part of a rule is created by which step. Listing 7.1 contains three mapping examples, each with different characteristics. Consequently, we explain the application of the sub-process for each mapping separately.

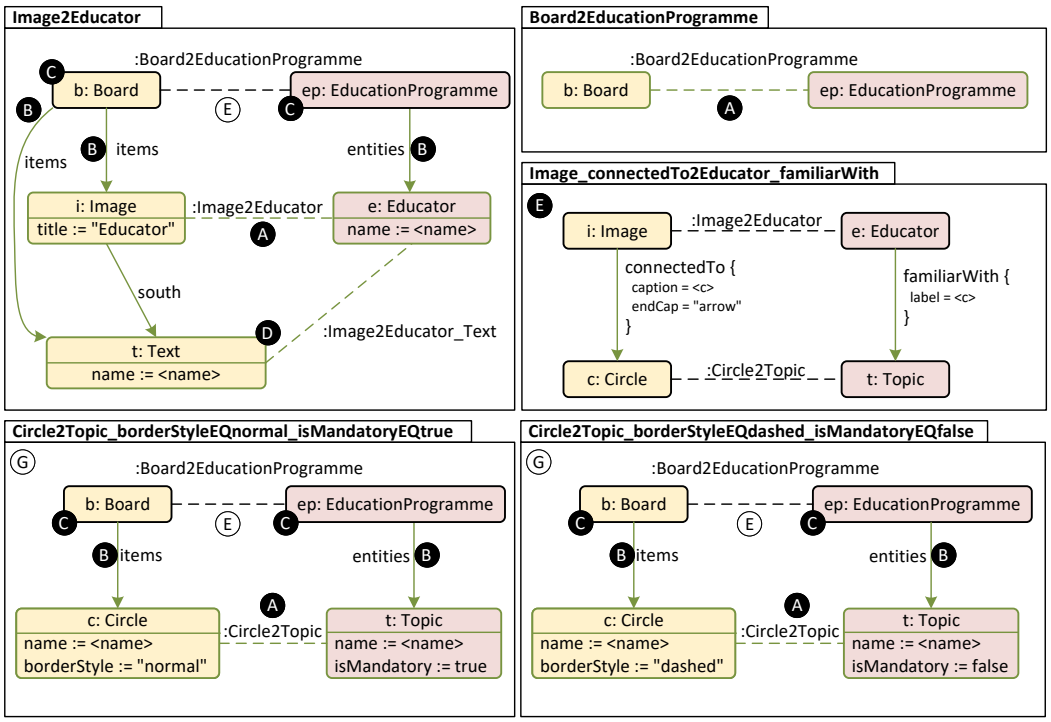

Figure 7.4: Generated TGG rules for the example shown in Listing 7.1

Board to Education Programme Mapping

Even though it is the last mapping in Listing 7.1, we start with the mapping of the classes `Board` and `EducationProgramme` (cf. Line 11). This mapping results in the axiom shown in the top-right corner of Figure 7.4. This axiom is created by the first step of the subprocess (cf. Ⓐ). As part of this step, the origin objects are created, which are instantiations of the classes listed in the mapping. In addition, a correspondence type for these two classes is added to the TGG definition, and the origin objects are linked using that correspondence. The creation of the rule `Board2EducationProgramme` is already complete after Step Ⓐ since no pattern is defined for either class, and no additional patterns can be inferred. The mapping also includes an association mapping in Line 12, which we cover later when discussing Step Ⓔ of the main process.

Creating Origin Objects

Circle to Topic Mapping

As with the previous example, the processing of the mapping in Line 1 starts with the creation of origin objects and setting them into correspondence. This results in the `Circle` and `Topic` objects and their correspondence shown in the two bottom rules in Figure 7.4. Please note that the subprocess leads to a combined version of these rules, which is split into two variants in Step Ⓖ.

Naming Conventions

Names of rules, correspondences, and objects are created based on conventions. For instance, object names result from taking each capitalized letter in the class name and adding a number if that name is already used elsewhere. The rule and correspondence name is automatically created based on the names of the origin classes. Explicit names can be provided by augmenting the mapping operator "<=>" to "<= *RuleName* : *CorrespondenceName* =>". If any of these names is omitted, the generated version is used. In the case of the mapping in Line 1, the name `Circle2Topic` was specified explicitly for illustration purposes.

Pattern Augmentation

In Step ❽, we augment the patterns defined for the source and target class. For this, we assume compositions are only used in metamodels if the composed element cannot exist independently of the composing element. Expressed as multiplicities, we imply a lower and an upper bound of 1 at the source end of the composition. Self-compositions are the only exception to this rule. To avoid endless compositions, a root element is allowed. Additionally, patterns are augmented if the provided pattern does not satisfy all lower bounds of associations. In the case of the rule `Circle2Topic`, no explicit pattern is provided for the source or target class. As part of Step ❽, we add that `Circle` is part of `Board` and that `Topic` is part of `EducationProgramme`.

Adding Objects

In Step ❾, we add objects to the TGG rule for all classes mentioned by associations in a pattern. We aim to add as few objects as possible to a rule. If two distinct associations refer to the same class, we only add one object, and the links created for the associations point to this common object. If an association is listed multiple times in a pattern, e.g. to satisfy a lower bound greater than 1, we add multiple objects. If two or more distinct associations occur multiple times in a pattern, we notify users about possible ambiguity regarding the linked objects.

Heuristic for Green Objects

By default, we assume objects mentioned in patterns must exist, so the `Board` object does not have a green border indicating object creation. In contrast, we assume origin objects and all objects resulting from property mappings containing a referenced property must be created as part of the resulting TGG rule (green objects). Section 7.3.4 explains how this default behaviour can be overwritten using modifiers.

Reflexive Composition

Since compositions are automatically added to the source/target pattern, reflexive compositions can be problematic. For reflexive compositions, the composing object is of the same class as the composed object, either directly or by inheritance. We detect this automatically and, in Step ⓖ, generate rule variants with and without the parent to avoid infinite containments.

The property and association mappings are processed after creating objects based on the (inferred) patterns. The `Circle2Topic` mapping contains two property mappings. The first (cf. Line 2) is converted into the property `name` of the `Circle` object and the `name` of the `Topic` object. Both properties are set to the same variable, meaning they must have the same value.

Processing Property Mappings

The second property mapping (cf. Lines 3 to 4) contains a value mapping. In the case of value mappings, the properties are added to the respective objects of the rule, but no variable is assigned. Instead, we assign the properties to a rule parameter and store the value mappings as parameter values, which are set when rule variants are created as part of Step Ⓖ of the main process. We do not create multiple rules immediately since further parameters might be added for additional value mappings, reflexive compositions or modifiers.

Processing Value Mappings

Image to Educator Mapping

The initial processing of the mapping starting in Line 6 is analogue to the first three processing steps of the previous example. As visualized in the top left rule in Figure 7.4, an `Image` and `Educator` object and a correspondence between them are created within Step ❶. Since `Image` is a subclass of `GeometricItem` and `Educator` is a subclass of `Entity`, the same pattern as in the previous example can be inferred in Step ❷, which results in adding a `Board` and an `EducationProgramme` object in Step ❸. Additionally, Line 6 specifies a pattern that an image shall have the title "educator", which is translated into a property of the `Image` object.

Line 7 maps the property `name` of the `Text` class to the property `name` of the class `Educator`. The `Text` class is associated with the `Image` class using the `south` association. To reflect this association in the rule, a `Text` object is added in Step ❸. This object is then linked to the `Circle` object. As mentioned, we assume by default that objects based on property mappings must be created by the resulting rule. The object creation is visualized by the green border of the `Text` object in the top left rule of Figure 7.4. The `name` properties of both classes are mapped by setting them to the same variable. Furthermore, an additional correspondence is added between the `Text` and `Educator` objects.

Processing Mappings of Associated Properties

Whenever new objects are added to a rule while processing property mappings, the subprocess returns to Step ❷ and checks if the additional objects imply a pattern. Since the class `Text` is also a subclass of class `GeometricItem`, the `Text` object created within Step ❹ also needs to be part

Augmenting New Objects

of a board. This is detected during the second iteration of Step ❽, and an `items` link is added between the `Board` and `Text` object. We reuse existing objects if possible and warn users if multiple alternatives exist. Similarly, if multiple property mappings contain a referenced property belonging to the same class, we assume the properties must be part of the same object.

Step ❾ processes association mappings. Each association mapping leads to at least one rule. The rule `Image_connectedTo2Educator_familiarWith` results from the association mapping in Line 8. This association mapping does not specify the target classes of the association. The information about possible target classes is derived from the metamodels. In the case of the association `familiarWith` of class `Educator`, `Topic` is the only target class. However, multiple target classes exist for the association `connectedTo`, which the class `Image` inherits from the class `GeometricItem`. The association `connectedTo` points to the class `GeometricItem`, which has several subclasses. We only consider those target classes corresponding to a target class on the other side of the mapping (cf. GR6). If there are multiple options, multiple rules are created. For the example, we require a correspondence type linking a subclass of `GeometricItem` to the `Topic` class. The only suitable correspondence type is `Circle2Topic`, introduced by the mapping in Line 1. Based on this correspondence, a rule for the association mapping is created containing a `Circle` and a `Topic` object. The parentheses around the property mapping in Line 9 indicate that this is a mapping of properties of the associations listed in Line 8. In the UML versions of Miro's and PEPML's metamodel, we define association classes having these properties. eMSL directly allows defining properties for binary association, making dedicated binary association classes obsolete. Hence, `connectedTo` and `familiarWith` are shown as regular associations in Figure 7.1.

Process Association Mappings

Identifying Target Classes

Mapping Properties of Associations

Note that within the rule for the association mapping, only the links are green, meaning only they are created and none of the objects. The objects are created based on other rules, which is also why the `Image` object does not require the property `title` to be set to the value "educator". Any `Image` object in correspondence with an `Educator` object using the correspondence type `Image2Educator` already fulfils this pattern. If multiple mappings between the classes `Image` and `Educator` would exist, and they shall not all be handled equally, it is necessary to define a custom correspondence name explicitly. In contrast to the patterns for the origin classes, patterns specified for associations or classes in an association mapping are added to the corresponding rule.

Patterns in Assication Mappings

7.3.3 Generating the TGG

After all mappings are processed, the main process continues with Step Ⓔ. Please note that at this point, `Circle2Topic` is still handled as a single rule having multiple alternative value pairs for the properties `borderStyle` and `isMandatory`. How this rule is split into the two visualized variations is explained when discussing Step Ⓖ.

Step Ⓔ checks if any rule resulting from an association mapping can be applied to other rules. This is necessary to avoid links on one side without the corresponding link on the other. Such symmetry is important to use the resulting TGG for transformations in both directions. As an example of this, consider the mapping of the composition `items` to association `entities` specified in Line 12. This composition mapping results in the three rules shown in Figure 7.5. Multiple rules are created since several correspondence types exist that match the target classes of the mapped associations. The composition mapping rules shown in Figure 7.5 can be applied to all rules that contain an `items` or `entities` link. Hence, they apply to the rules `Circle2Topic` and `Image2Educator`. In both cases, the necessary composition on the other side already exists due to the pattern augmentation. Applying the composition mapping only adds the missing correspondence between the `Board` and `EducationProgramme` objects. By default, composition mapping rules are deleted after Step Ⓔ since this information is at this point contained in the rules that create the respective objects. If users wish to keep these rules, they can override this default behaviour (see Section 7.3.4).

Applying Association Mappings

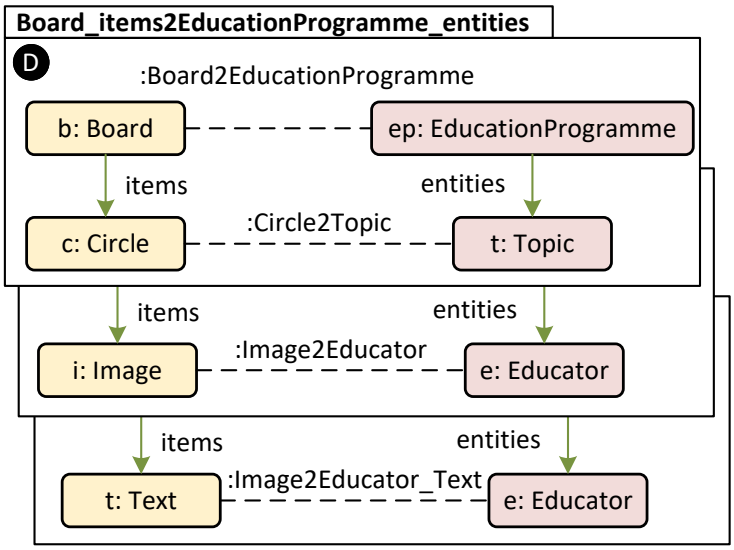

Figure 7.5: Temporary rules for composition mappings

Step Ⓕ traverses all created rules and checks if objects in rules exist that are not part of a correspondence. If any such objects are found, it is checked if a suitable correspondence type exists, and if so, a correspondence is created. Warnings are issued if no suitable correspondence type is found or no partner object exists which is not already part of a correspondence. However, this can be intended because not every object has to correspond to an object on the other side. If the correspondence between the `Board` and `EducationProgramme` objects had not been added within Step Ⓔ, it would be added in Step Ⓕ.

Inferring Correspondences

Step Ⓖ identifies all parameterized rules and creates a rule variant for every possible parameter value, which partially addresses GR7. For instance, the rule `Circle2Topic` has a parameter due to the value mapping in Line 4. This value mapping contains two value pairs, and a separate rule variant is created for each pair. The resulting two variants are shown at the bottom of Figure 7.4. Value alternatives in patterns, the usage of the "?" modifier (see next section) and reflexive compositions also lead to rule parameters and the creation of multiple rule variants. If multiple rule parameters exist, rules for all possible parameter value combinations are created.

Creation of Rule Variants

Finally, Step Ⓗ generates constraints that can be deduced from the metamodels and adds them to the TGG (cf. GR9). In particular, constraints are added to enforce the lower and upper bounds of associations. As we explain in the next section, further constraints for set semantics of association or root objects can be generated if certain options are specified in a mapping.

Constraints for Static Semantics

7.3.4 Advanced Features

GenTGG offers additional features beyond those illustrated using the example in Listing 7.1. The following describes how options can be set to manipulate the generation process and how modifiers allow overwriting default behaviour.

Influencing the Generation Process

Options influencing the generation process can be specified at the beginning of a mapping file. The following options can be set:

- `:root=Class1, ...;` can specify classes of which only a single object shall be allowed in a model. The option results in an additional constraint for each class listed as option values. For instance, this option can specify that a Miro model shall only have a single `Board` object.

- `:set=Class1.association, ...;` can specify that set semantics for the listed association shall apply. If set, constraints are generated, ensuring these associations cannot point multiple times to the same object.

- `:constraints=set,root,lowerBound,upperBound;` defines which constraints are generated within Step Ⓗ. By default, no constraints to enforce the lower bounds are generated since those are guaranteed due to the lower bound pattern augmentation.

- `:disableLowerBoundAugmentation` disables pattern augmentation based on lower bounds of associations. Instead, they can be enforced using constraints (see previous option). Similarly, pattern augmentation based on compositions is disabled by `:disableCompositionBoundAugmentation`.

- `:keepCompositionRules` disables that composition mapping rules are deleted after Step Ⓔ. This option can be useful if these rules shall be adapted or to see which composition mapping rules are generated.

- `:excludeRules`, `:excludeConstraints` and `:excludeCorrespondences` can exclude certain information from the generated TGG. A list of rules, constraints, or correspondence names follows each option. Excluding certain information is beneficial when these rules are manually altered after the initial generation (cf. Section 7.4.4).

- `:matchCommonProperties` allows adding property mappings automatically for properties with the same name. For instance, if both classes in a class mapping have a property `name`, it would be automatically added as a property mapping if `name` is not used within any other property mapping. Alternatively, this option accepts a comma-separated list of properties to which the option shall apply.

Modifying Default Object Creation

We assume that the origin objects of a mapping and all objects resulting from property mappings must be created as part of the corresponding rules (green objects). Furthermore, we assume that all objects within patterns must exist, independently of whether the objects are explicitly defined or inferred. These default assumptions can be overwritten by adding modifiers after a class name. Modifiers
A "+" modifier ensures that the object is created within the rule derived from that mapping. This modifier is implicitly added to each class in a class or

property mapping. In contrast, the "!" modifier states that the object of that respective class must exist for the rule to match. This modifier is implicitly added to each class in a pattern. The "?" modifier indicates that both cases (creation and existence) have to be considered. For each "?" modifier, a rule parameter is added during the mapping processing. In Step ⓖ, variants of the rule are created, where the object is created, and where it exists (cf. GR7).

Link Creation

Links in association mapping rules are always created since link creation is the purpose of those rules. For all other rules, the existence or creation of the linked objects determines the link creation. Assume we have an object A and B connected by a link c. Only if the rule specifies that both A and B must exist, the link must exist as well. Otherwise, the link has to be created because it cannot exist without an existing object on either end.

Dealing with Uninstantiated Classes

We detect automatically if a class mentioned in a pattern is not instantiated by any TGG rule and issue warnings during the generation. A possible resolution of the problem is adding a "+" or "?" modifier to the class in the pattern or creating an additional rule that creates the object in question. Modifier usage examples are given as part of the evaluation in the next section.

7.4 Evaluation

This section provides insights on the implementation of the GenTGG generator and evaluates the expressiveness of GenTGG. Furthermore, we provide additional details on using GenTGG for PEPML and discuss limitations.

7.4.1 Implementation

Available on GitHub

We implemented the GenTGG generator in JavaScript/TypeScript. The source code is available on GitHub[2]. A web-based generator and editor exists to quickly generate TGGs[3]. The generated TGG is based on eMSL and can be used in combination with eMoflon::Neo and eNeo.js.

Not based on Eclipse/EMF

eMoflon::Neo is built based on Java, Eclipse, EMF, xText and xPand. It also provides a code editor for eMSL with auto-completion. This technology stack is very powerful, especially for MDE, but it has a steep learning curve and problems with incompatibilities. While we want to enable users of GenTGG to execute generated TGGs with eMoflon::Neo as an engine, we did not want to couple GenTGG to eMoflon::Neo's technology stack.

[2] https://github.com/dwolters/gentgg
[3] https://dwolt.de/gentgg

7.4.2 Expressiveness

We evaluated GenTGG's expressiveness by determining to which degree it can specify eMoflon::Neo's TGG test cases. Table 7.1 lists all test cases with information about the number of rules of each TGG. The TGGs `CompanyToIT` and `FamiliesToPersons` are among the set of standard examples of the BX community[4]. Moreover, the latter is a benchmark for bidirectional transformations [ADJ+17]. In the following, we explain for each TGG individually what can be described using GenTGG mappings and highlight important characteristics of each TGG. We only show selected and simplified metamodels, rules and GenTGG mappings. The complete versions of the original eMoflon::Neo TGG test cases[5] and the GenTGG mappings[6] to describe them are available on GitHub (see footnotes). The GenTGG mappings for these evaluated TGGs can also be selected in the online editor (see Section 7.4.1).

Expressing eMoflon::Neo test cases

Table 7.1: TGGs used for evaluating expressiveness

TGG	Axioms	Normal Rules	Single-Sided Rules	Total Number of Rules
`ClassInhHier2DB`	1	3	0	4
`CompanyToIT`	1	3	0	4
`FacebookToInstagram`	2	3	4	9
`FamiliesToPersons`	1	8	0	9
`JavaToDoc`	1	2	0	3

ClassInhHier2DB

Figure 7.6 shows a simplified version of the source and target metamodels of this TGG. In contrast to the original metamodels, the depicted version does not contain class properties. The source metamodel has two interesting characteristics: (1) it defines backward associations, and (2) it uses the multiplicity 1..1. A backward association means that for an incoming association, there is also an outgoing association to the originating object with the opposite semantics. Examples are `classes`/`package`, `attributes`/`class` and `subClass`/`superClass`.

The TGG `ClassInhHier2DB` contains four rules, among them one axiom. Aside from the axiom, all rules of the original TGG ignore the backward associations. Thereby, they also violate the 1..1 multiplicity.

[4] https://bx-community.wikidot.com/examples:home
[5] https://github.com/eMoflon/emoflon-neo/tree/master/examples
[6] https://github.com/dwolters/gentgg/tree/master/public/examples

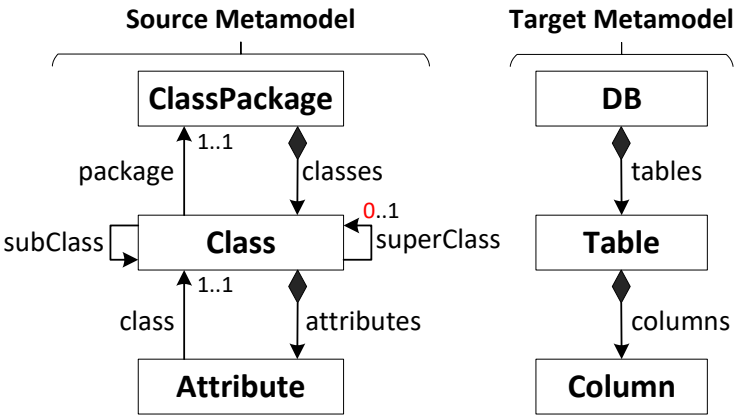

Figure 7.6: Simplified metamodels for the TGG `ClassInhHier2DB`

Listing 7.2 shows an excerpt of the GenTGG required to express the TGG `ClassInhHier2DB`. Property mappings are omitted because they do not cause any differences in generated rules. The axiom generated for Line 1 is equivalent to the axiom of the original TGG. In contrast to the original rules, the rules generated for Lines 2 and 3 properly set the backward associations because they can be deduced based on the multiplicities. We decided not to show these rules here since we see the same deduction is made for the fourth rule, which we discuss in detail.

Listing 7.2: Excerpt of the GenTGG mappings for `ClassInhHier2DB`

```
1    ClassPackage <=> DB
2    Class <=> Table
3    Attribute <=> Column
4    Class{-superClass
5         <-subClass} <=SubClassToTable=> Table!
```

The rule generated based on Line 3 also contains a redundant part compared to the original. A `ClassPackage` object for the attribute's class and a `Database` object for the column's table are added during pattern augmentation. However, this makes no semantic difference to the original TGG since those objects are required to create any `Class` or `Table` object.

Figure 7.7 compares the original `SubClassToTable` rule (left) with the version generated based on Lines 4 and 5 (right). The original can only partially be described using GenTGG since it contains a 2-to-1 correspondence. One of these correspondences would be created based on a GenTGG mapping, while the other would need to be set manually. During TGG creation based on mappings, warnings are raised for all objects without correspondences. Hence, users can act upon this warning and add the missing correspondence manually.

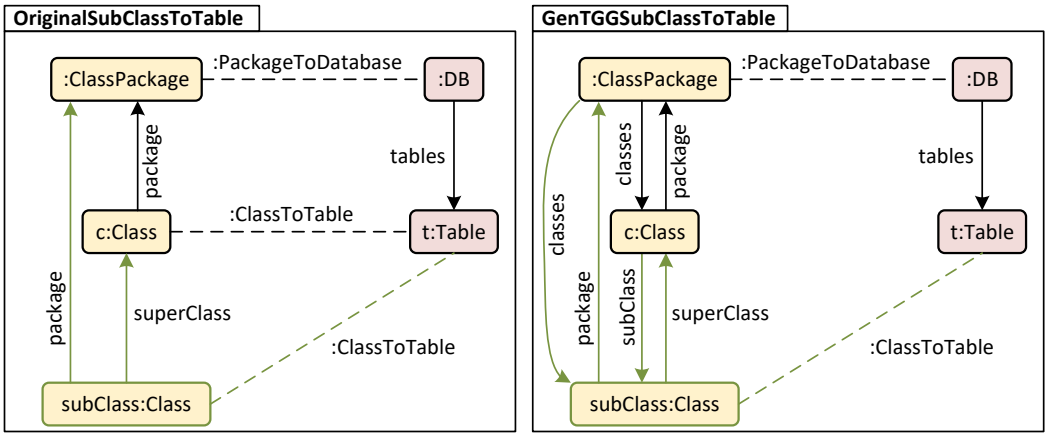

Figure 7.7: Differences of the `SubClassToTable` rule

The original `SubClassToTable` rule on the left of Figure 7.7 does not define the `classes` and `subClass` links. For our generated rule, we can deduce the former since it is a composition and because no `Class` object should be without a `ClassPackage`. We added the association to the pattern in Line 5 to also include the `subClass` link. Line 5 is also an example showing that incoming associations can be used within patterns.

We use the "!" modifier on the `Table` class at the end of Line 5 to denote that the rule shall assume that an object of this class exists. This operator is necessary since the original `SubClassToTable` rule only alters the source side.

For our generation to work, we altered the lower bound of the `superClass` association to 0 (see Figure 7.6). Otherwise, every class would require a superclass. We believe this alteration is fair for the comparison since the original TGG does not create every class with a superclass, either. So, the TGG can nearly completely be described using GenTGG. One rule is equivalent, two are more accurate concerning the source metamodel than the original rules, and one requires adding one further correspondence after generation.

CompanyToIT

A TGG semantically equivalent to the `CompanyToIT` can be generated based on GenTGG mappings. The only difference is the absence of the `Admin` and `Router` objects in the generated version of rules `EmployeeToPCRule` and `EmployeeToPCLaptop`. However, these objects are redundant and have no semantic effect. We decided not to include examples for this TGG in this section since GenTGG mappings can fully describe it. Please consult the sources listed at the beginning of this section to see the original TGG and the corresponding mappings.

Noteworthy is that within the `AdminToRouteRule`, the `Admin` object requires a link to an existing `CEO` object. GenTGG can infer this link if the multiplicity for the `ceo` association is changed from 0..1 to 1..1. Otherwise, this link has to be added to the pattern of the `Admin` class in the mapping. We decided to change the multiplicity because the original TGG never creates an `Admin` object without an associated `CEO` object.

Families2Persons

Figure 7.8 shows the source and target metamodels used by the TGG `Families2Persons`. In contrast to the original metamodels, we changed the aggregation between `FamilyRegister` and `Family` to a composition. This change is in line with what is expressed by the original TGG since `Family` objects are never created without being part of a `FamilyRegister`.

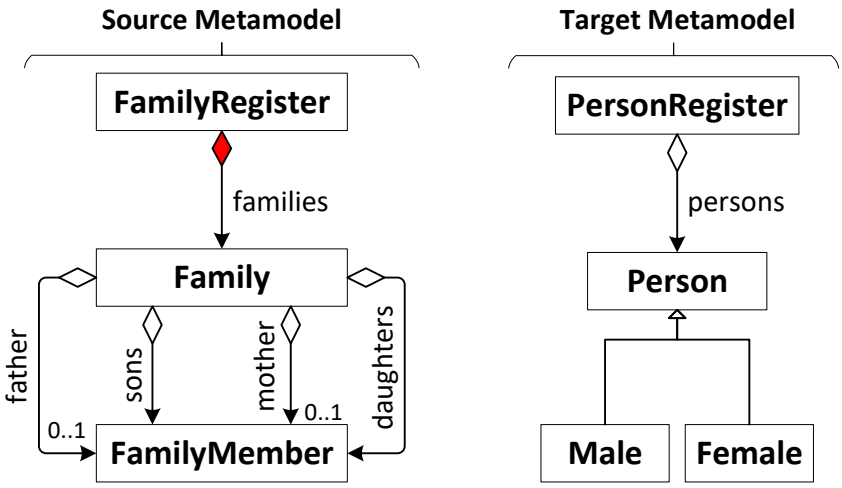

Figure 7.8: Simplified metamodels for the TGG `Families2Persons`

The original TGG describes how family members are matched to males and females. The `Family` class has multiple associations pointing to the `FamilyMember` class to denote sons, daughters, the mother and the father. The information about the gender of a family member is deduced based on these associations.

Listing 7.3 shows an excerpt of the GenTGG mappings to generate the TGG `Families2Persons`. Again, property mappings are omitted to reduce unnecessary complexity. Line 1 maps the two register classes and results in an axiom equivalent to the one of the original TGG. In the following, we discuss the rules generated for Line 2. The rules generated for Lines 3 to 5 are analogue to the rules for Line 2 and handle other types of family members.

Listing 7.3: Excerpt of the GenTGG mappings for `FamilyToPersons`

```
1    FamilyRegister <=> PersonRegister
2    FamilyMember{<-daughters-?} <=> Female
3    FamilyMember{<-sons-?} <=SonToMale=> Male
4    FamilyMember{<-mother-?} <=MotherToFemale=> Female
5    FamilyMember{<-father-?} <=> Male
```

Figure 7.9 shows the two rules generated for Line 2. Both rules specify that a daughter of a family is represented as a female on the target side. The difference between these two rules is that the left rule creates a `Family` object, while the right rule assumes that this object exists. For FWD, one of these rules would suffice because their operationalizations for FWD are equivalent. For BWD, we require both rules. If only one of these rules would exist for BWD, either every family member is created with a new `Family` object or family members can only be added to an existing `Family` object.

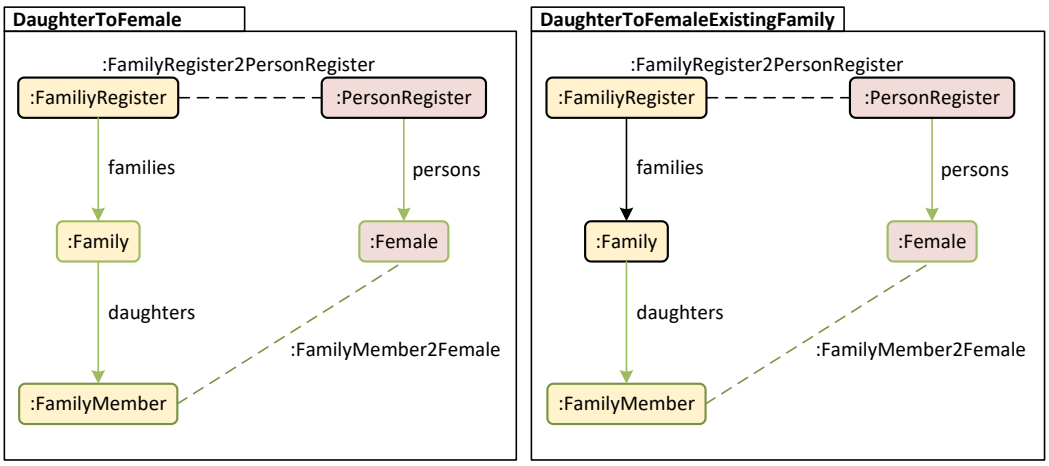

Figure 7.9: Variants of the `DaughterToFemale` rule

Without the "?" operator in Line 2, only the right rule of Figure 7.9 would be generated. However, without this operator, our approach would raise a warning since no rule exists that creates a `Family` object. To address this warning, the object must either be created using a single-sided rule or users must indicate in the GenTGG mapping that the object shall also be created. The latter is done by adding the "?" operator in Line 2, which results in both rule variants being created. Using a "+" operator does not suffice since then only the left rule is created. Due to the "?" operators, we can generate nine rules based on the five mappings in Listing 7.3.

One difference exists between the original rules and those created for Lines 2 to 5. The original TGG rules use a property constraint to split the

person's full name into first and last names and only assign the first name to the family member. This is not shown in our excerpt since we omitted properties. Defining such property constraints is not supported by GenTGG yet. If eNeo.js is used for transformation, we can add a preprocessor to split up the name prior to transformation, making the property constraint obsolete.

FacebookToInstagram

The TGG `FacebookToInstagram` contains four single-sided rules that only define objects on one side. Single-sided rules are out of the scope of GenTGG since a mapping always starts by relating a source to a target class. Hence, such rules must be added manually. Three rules of the TGG can be expressed using mappings. However, the rules `UserToUserIslandRule` and `UserNetworkBridgeRule` are expressed as a single rule within our TGG. Most interesting are the two rules that cannot be generated based on GenTGG mappings, which are shown in Figure 7.10 and discussed in the following.

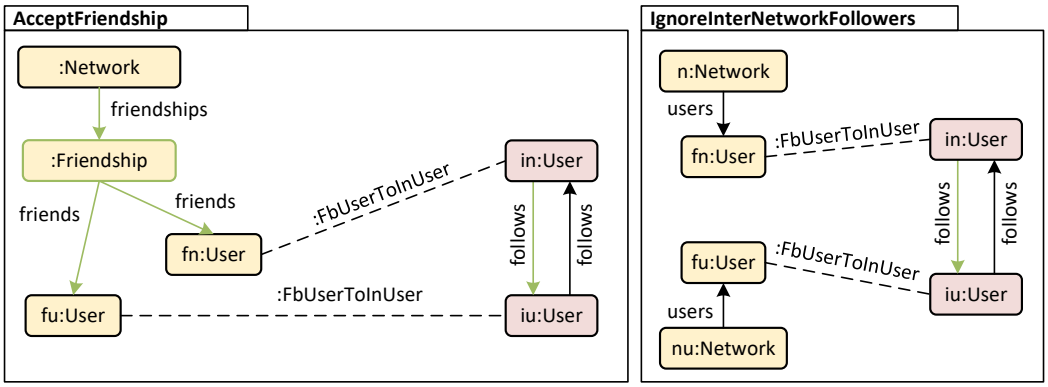

Figure 7.10: Example of two rules not expressible with GenTGG

The `AcceptFriendship` rule on the left of Figure 7.10 creates a `Friendship` object on the source side and a `follows` link on the target side. Relating an association class like `Friendship` to an association is not possible with GenTGG. Only mappings between the same kind of elements are supported, i.e. between two classes, two properties or two associations.

The `IgnoreInterNetworkFollowers` rule on the right of Figure 7.10 has two independent patterns on the source side. We cannot create independent patterns within the generated rule since all objects are always directly or transitively connected to the origin objects on the respective side of the rule.

JavaToDoc

The TGG `JavaToDoc` contains three rules. The rule `ClassToDocRule` can be expressed completely using GenTGG. The other two rules can have minor differences when specified by mappings. The metamodels of this TGG are interesting because they contain reflexive compositions. As mentioned, we detect such reflexive compositions and allow a root object to exist.

The original axiom `RootToRootRule` only creates a `Package` object on one side and a `Folder` object on the other. Our generated version of the rule also creates a `Doc` object representing the package. In the original TGG, `Doc` objects are only created for sub-packages. We provide a tweaked version of the mapping that does not create the `Doc` object for the root package[7]. However, the tweak requires a double specification of the `Package` to `Folder` mapping and using the option to exclude specific generated rules.

The rule `SubToSubRule` handles the creation of sub-packages and maps objects representing such packages to a `Folder` and `Doc` object on the other side. In the original rule, no correspondence is set between the `Package` and the `Doc` object. However, the generated rule has such a correspondence, and we argue that it makes sense to have one because the `Doc` object represents the package as well.

Result Overview

After discussing each TGG in detail, we present in Table 7.2 an overview to which degree GenTGG mappings can express the eMoflon::Neo test cases. For each TGG, we list the total number of rules, how many are fully expressible, and how many can almost fully be generated by GenTGG. As the comparison shows, GenTGG mappings can express a large extent of eMoflon::Neo test case TGGs. Only the TGG `FacebookToInstagram` cannot be expressed well by GenTGG since two rules cannot be expressed at all and because it contains four single-sided rules. Such single-sided rules are out of GenTGG's scope.

We also compared the lines of code of our mappings with those of the original TGGs. If we ignore blank lines and those only containing brackets, the GenTGG mappings require 80% fewer lines of code. This difference in lines of code can even be higher in cases where multiple variations of rules are required, like for the TGG `Families2Persons` or the one required for PEPML model extraction.

[7] `https://github.com/dwolters/gentgg/.../mappings/DocToJavaTweaked.etb`

Table 7.2: Coverage of evaluated TGGs

TGG	Total Number of Rules	Fully Expressable	Almost fully Expressable
`ClassInhHier2DB`	4	3	1
`CompanyToIT`	4	4	0
`FacebookToInstagram`	9	3	0
`FamiliesToPersons`	9	1	8
`JavaToDoc`	3	3	0

7.4.3 Feasibility Study: PEPML's Visual Syntax

We used an excerpt of the GenTGG mappings required for PEPML's visual syntax as a running example in this chapter (cf. Listing 7.1). As mentioned in Section 6.3, the resulting TGG rules shown in Figure 7.4 contain representatives of all four groups of rules required for PEPML's visual syntax. The rules required for the remaining parts of the visual syntax are structurally similar. Just the class names, associations and properties differ. The full GenTGG mappings for PEPML are available on GitHub[8].

As mentioned in Section 6.3, there is one case that GenTGG cannot fully cover. While the rule mapping `Rectangle` to `EducationComponent` (see Figure 6.21) can be generated based on a GenTGG mapping, it requires negative application conditions (NACs) for the transformation to be successful. Generating the rule and adding the NACs afterwards is cumbersome and error-prone since multiple variants of this rule are needed. Therefore, we allow adding NACs at the end of a GenTGG mapping.

Listing 7.4 shows the GenTGG mapping for the rule depicted in Figure 6.21. Lines 1 to 4 are GenTGG mappings. Line 5 specifies NACs for the rule to be generated. The remaining lines are eMSL pattern specifications used by the NACs. Since Line 4 defines a value mapping, two rule variants are created. One where `borderStyle` is "dashed" and `isMandatory` is "false", and one where `borderStyle` is "normal" and `isMandatory` is "true". The NACs specified in Line 5 are added to both rules.

Aside from the pattern required for NACs, GenTGG can be used to completely describe and generate the TGGs required for PEPML model extraction. GenTGG's heuristic of automatically creating association mapping rules for all compatible correspondences greatly reduces the effort required to define a TGG. If users add a new class mapping, compatible association mappings are generated immediately.

[8] https://github.com/dwolters/pepml/tree/main/tggs

Listing 7.4: Mapping between `Rectangle` and `EducationComponent` with NACs

```
1  Rectangle <=> EducationComponent
2  .name <=> .name
3  .borderStyle <=> .isMandatory
4  ["dashed"=false,"normal"=true]
5  forbid src(ContainsOtherRectangle) && trg(IsComposite)
6
7  pattern ContainsOtherRectangle {
8      r: Rectangle {
9          -connectedTo->r2 {
10             .startCap := "filled_diamond"
11         }
12     }
13     r2: Rectangle
14 }
15
16 pattern IsComposite {
17     ec: EducationComposite
18 }
```

7.4.4 Limitations

The mapping language does not replace writing TGG rules completely. It helps to create rules that originate in a single object on each side. Hence, rules requiring two independent patterns on one side cannot be expressed since the objects created via GenTGG are always connected directly or transitively with the other objects on that side. Also, rules that relate an association on one side to a node on the other cannot be expressed well. Furthermore, single-sided rules that only specify objects on one side cannot be expressed using GenTGG. The correspondences are all created based on heuristics and are mostly limited to one-to-one correspondences. It is only possible to create a one-to-many correspondence for an object when a property of an associated class is mapped. Other types of one-to-many correspondences must be expressed via additional handwritten rules.

Scope of GenTGG

Rules may have to be manipulated after generation, e.g. to add property constraints or extend the pattern. Regenerating the TGG would then overwrite these changes. For this purpose, the option to exclude rules from the TGG generation can be used. Users could specify and generate a first version of the rule using GenTGG, manipulate it manually afterwards, and then exclude it from future TGG generations. In such a way, the missing property constraints can be added to the rules generated for the TGG `Families2Persons`. Alternatively, GenTGG could be extended to support such constraints.

Rule Manipulation after Generation

Reduce Entry Costs of Using TGGs

While the GenTGG simplifies the creation of a TGG, knowledge about how TGGs work is still necessary. The standard heuristics lead to good results, but judging if additional rules are needed requires expertise. Otherwise, TGGs may not transform models as intended. In that sense, GenTGG reduces the entry cost of creating TGGs, but it does not substitute being able to write TGG rules manually.

While GenTGG does not cover the full expressiveness of TGGs, it is well-suited to map a language to its visual representation. Furthermore, the evaluation based on eMoflon::Neo's test cases illustrates its usefulness for other use cases as well.

7.5 Summary

We use model transformations based on TGGs to extract models from Miro boards. Writing the TGGs for this use case can be tedious and error-prone since many interconnected rules have to be written, often following a similar scheme. We developed an approach called GenTGG that reduces the effort of writing TGGs needed for our use case. Instead of writing TGG rules directly, we specify mappings between classes, properties and associations of the source and target metamodel. We explained how these mappings can be extended to a TGG by leveraging information in the metamodels and by applying heuristics. Generated TGGs can be enriched with manual TGG rules to cover complex mappings if necessary. Our approach only covers part of the expressiveness of TGGs but simplifies the definition of an important subset. The approach is particularly useful for our application scenario of mapping content from an online whiteboard to an instance of a modelling language. GenTGG is, however, not limited to this use case and can be beneficial to define mappings between any two modelling languages. We evaluated to which degree the test cases of eMoflon::Neo can be expressed using our approach to determine the expressiveness of GenTGG. The results show that the specification of 4 out of 5 TGGs used as test cases could be replaced with GenTGG mappings.

Chapter 8

Model-Driven Information Integration

In the previous chapters, we introduced a language to model professional education programmes and provided a way to model them visually with online whiteboards. However, education providers use many software tools for designing and managing such programmes. Our visual modelling approach is neither intended nor capable of replacing all of them. It is an addition to the toolset of education providers, which is intended for collaboration with others and to maintain a programme overview. This chapter gives examples of software tools containing semantically relevant information and presents a technique to integrate the distributed information into a single PEPML model. Thereby, the chapter addresses Research Question RQ3, concerned with integrating the information spread across tools used by education providers. Furthermore, it contributes to the *Modelling Language Validation* step of the language engineering process used to develop PEPML by describing an integration with a project management tool illustrating PEPML's work organization capabilities. The integrated PEPML model can serve as a tool-overarching perspective on professional education programmes and provide insights into the situational context, which we further utilize in Chapter 9.

Section 8.1 defines the requirements for our information integration approach, gives a solution overview and discusses related work. Section 8.2 focuses on integrating information from tools providing information via APIs. In particular, the section includes a feasibility study demonstrating the interplay of PEPML's work organization concepts with a tool for project management. Section 8.3 is concerned with using file-based information as sources for a PEPML model and uses Spreadsheets as an example to show feasibility.

8.1 Overview

This section provides an overview of the information integration approach presented in this chapter. It covers the requirements for our approach, a solution overview and related work.

8.1.1 Requirements

The following defines the Integration Requirements (IR) for our approach to integrate information from tools used to design and manage professional education programmes. These requirements are derived based on Research Question RQ3 and consider the modelling approach we presented in the previous chapters.

IR1 **Information Access** *It shall be possible to integrate information provided via APIs or as files.*

To integrate information, we must first be able to access it. For this, we have to differentiate between software tools with internal and file-based data storage. An API or file export is required to access information from tools with internal data storage. APIs allow tool vendors to provide controlled access to the information stored within their tools. However, not all tools provide an API but may offer a file export. The difference to general file-based storage is that such exports are often triggered manually. For tools with file-based storage, the files are automatically created when information is persisted. Both information access via APIs and files must be supported.

IR2 **Integration into a single PEPML Model** *It shall be possible to integrate information into a single PEPML model.*

Information integration can be bilateral, e.g. by adding information one tool provides to the other. This bilateral integration provides value in specific cases, but separate tools often manage information not immediately relevant to other tools. However, information integration is vital for education providers to gain an overarching perspective, providing insights into where information is stored and showing potential inconsistencies. With PEPML, we have introduced a language that can express such a perspective for professional education programmes. Hence, the information provided by the tools used by education providers shall be integrated into a single PEPML model.

IR3 **Information Excerpts** *It shall be possible to use an excerpt of the information provided by a tool.*

Not all information within a tool education providers use will be semantically relevant. Hence, we must be able to integrate partial information.

IR4 **Trace Information** *It shall be possible to trace integrated information back to its source.*

When a PEPML model is built based on multiple sources, we must be able to trace information back to its origin. This is especially important in case of conflicting information.

IR5 **Integration of Updates** *It shall be possible to update information integrated into a PEPML model.*

When information within a tool is changed, the corresponding information within a PEPML model has to be updated. Hence, it does not suffice to integrate information once.

These requirements above state what our approach shall address. However, certain aspects are out of the scope of this thesis. Requirement IR1 states the need to support information access via files and APIs. If tools neither have an API nor can provide information as files, machine-based information access is challenging. Web scrapping techniques are an option in the case of Web applications. For instance, Han and Tokuda [HT08] developed a technique to offer information stored in the UI of web applications as a service to others. For native applications, automated UI interactions can extract information but are highly specific to individual applications and UI technologies. For this thesis, tools without an API or file-based storage/export are out of scope.

Requirement IR2 implies a unidirectional transformation of information provided by the software tools into a single PEPML model. The reverse direction can also be interesting, e.g. adding information from the integrated model to a specific tool. We illustrate this reverse direction for our integration with the project management tool OpenProject in Section 8.2.2. In that specific case, the reverse direction is of interest for us to demonstrate the work organization capabilities of PEPML. Aside from that, the reverse direction is out of scope and subject to future work.

8.1.2 Solution Overview

Education providers use a broad range of software tools to design and manage professional education programmes, e.g. for project management, remote collaboration, content creation or evaluations. An overview of information stored within those tools and their relation is usually solely known by individual programme designers/managers or, at most, documented informally. In the following, we present our approach to integrating the information contained within those tools into a single PEPML model. This model can then provide a structured overview of the information about a programme to education providers. Furthermore, we can leverage such a model to determine situational context information programmatically and give education providers recommendations for improvement (see Chapter 9).

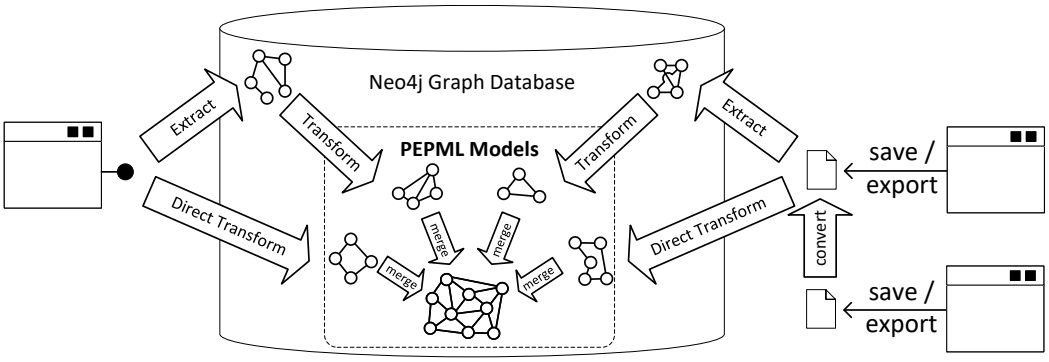

Figure 8.1: Overview of options for handling tools providing information via APIs or as files

Figure 8.1 gives an overview of how information can be integrated into a single PEPML model. Independent of whether information is exposed via an API or as a file (cf. IR1), we can proceed in two ways: (1) We stick to the information model of the API or file format, load the information into a Neo4j database, and use TGGs to transform it into a PEPML model. This is the same technical solution we use for obtaining PEPML models from Miro boards (see Chapter 6). (2) We directly transform the information into a PEPML model. In both cases, we suggest using our model merging technique to create a single PEPML model (cf. IR2) since it allows tracing the origin of information (cf. IR4) and supports information updates (cf. IR5).

Figure 8.2 shows the process of how to enable the integration of information from other tools. The first step is to inspect the file format or API and identify the information relevant to PEPML (cf. IR3). Next, users must decide whether to develop a direct transformer or leverage TGGs. Each approach

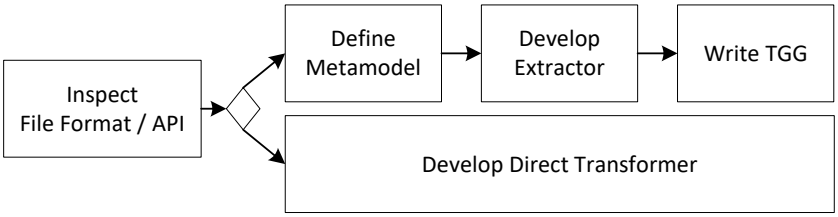

Figure 8.2: Process to enable information integration from other tools

has strengths and weaknesses. Direct transformers provide a high degree of flexibility and can be a good option if only limited information needs to be transformed. Defining a metamodel for the information, developing an extractor and writing a TGG seems more effort, but we argue it is not. A metamodel must only be defined for the relevant parts, and it allows to check if the extracted information is syntactically valid. Developing the extractor is the main effort, but we can do a mostly one-to-one extraction of the information and store it as a model conformant to the tool's information model in Neo4j. The main effort here is to translate object references to relationships for the model in Neo4j. Using TGGs instead of hardcoded transformation has the benefit that the transformation can be adapted more quickly, and we can rely on existing engines for transformation. If information is transformed backwards at any point, we can leverage the same TGGs and compute the delta between the new and the old information based on the models.

Enable Information Integration

The development of direct transformers can only be supported to a limited extent since the development is inherently specific to individual tools or formats. We can primarily assist with triggering the transformation and merging after a successful transformation. For extraction combined with TGG-based transformation, we can leverage eNeo.js as a transformation engine, reuse existing preprocessors, and simplify the TGG development with GenTGG. Moreover, existing extractors can be used as blueprints for developing new ones. In any case, manual development is necessary, but it just has to be done once for a tool/file format. In this chapter, we focus on the TGG-based approach and present two feasibility studies showing the integration of information from a project management tool and spreadsheets.

Assist in Development

A benefit of file-based data storage is the possibility to leverage file format converters. Assume we have a transformer to PEPML for a file format F_X, and the tool from which we want to integrate information uses the format F_Y. If a converter exists that can convert a file in format F_Y into a file in format F_X, we can reuse the existing transformer (see right part of Figure 8.1). Suitable converters can be determined automatically, as we explain in Section 8.3.2.

Leveraging File Format Conversion

Aside from being able to transform information from other formats so that it can be incorporated into a PEPML model, it is also necessary to have triggers that start the process. The simplest option is manually triggering the transformation and merging process. However, this means constant additional effort for users, which should be avoided. For file-based storage, the process can be triggered when file changes are discovered. For tools where we obtain the information via an API, we can only react to information changes when the API provides means to detect them. Some APIs allow registering listeners, e.g. webhooks, which are invoked when information is manipulated. We must resort to user-invoked transformations or polling if no such concept exists.

Triggering Information Integration

8.1.3 Related Work

For the information integration approach presented in this chapter, we leverage the same technique used to extract PEPML models from online whiteboards. In Section 6.1.3, we already covered TGGs and model merging. This related work section focuses on approaches dealing with multiple information sources and languages.

In [Ste17], Stevens discusses the implications of using bidirectional transformation not only between two but between n models. She calls this multiary transformation. In our work, we consider information from tools used by education providers as the basis for obtaining a PEPML model. We do not directly transform the information between tools but use a PEPML model as an intermediary. For instance, information from an online whiteboard is added to a PEPML model, from where information about tasks and their structure may be added to a project management tool (as we show in Section 8.2.2). Due to the availability of a common intermediate model, the transformation between the tools is more straightforward compared to the general case discussed by Stevens.

Multiary Transformations

Stünkel et al. [SKLR21] present an approach called *Comprehensive Systems* for multiary model transformation. They introduce a new concept called *commonality structure*, which describes how elements of the different modelling languages relate to one another. The benefit of their approach is that they can evaluate constraints over all models based on this structure. We describe for each tool individually how their provided information relates to PEPML but do not define information commonalities between tools. However, we can check consistency when merging PEPML models obtained by different tools.

Comprehensive Systems

Arifulina [Ari16] describes a core language for On-the-Fly markets to handle heterogeneous service descriptions. Service description languages are mapped to this core language, which allows uniform treatment of services described in different languages. We follow a similar idea by mapping the information provided by different tools to PEPML. However, PEPML is not a core language since it aims to cover all life cycle phases of professional education programmes on a high abstraction level. Hence, it adds an overarching perspective on the information provided by multiple tools and refers to them for more detailed information. For instance, PEPML stores information about tasks and their relation to a programme, but due dates or assignees are managed outside of PEPML, e.g. in a project management tool.

Core Language

For our information integration approach, we require the information provided by a tool to be stored as a model in a Neo4j database. This extraction of information into a model requires manual coding. Proper RESTful APIs support a concept called Hypermedia as the Engine of Application State (HATEOAS) [RAR13], which enables machines to discover an API on their own, e.g. using a library like Traverson[1]. If HATEOAS is supported properly, information extraction could be automated to a more considerable degree. Yet, not all information will be relevant, and a steering of the extraction is necessary. GraphQL[2] is an alternative to RESTful APIs. In contrast to a RESTful API, GraphQL allows querying information over a single endpoint. The information provided via this endpoint is described by a schema. As the name suggests, information exposed by a GraphQL API is already a graph, making extraction far more straightforward, and a generalized approach for our purposes seems feasible. Stünkel et al. [SvBRL20] even propose an approach to integrate data spread across multiple GraphQL APIs. Unfortunately, HTTP-based APIs that are not fully RESTful are still the most common, and we cannot assume that all relevant tools have a RESTful or GraphQL API.

HATEOAS

GraphQL

8.2 API-based Integration

This section highlights relevant tools for designing and managing professional education programmes, which can be accessed via an API. Based on the project management tool OpenProject, we show the feasibility of API-based information integration. Section 8.3 covers tools with file-based storage.

[1] https://github.com/traverson/traverson
[2] https://graphql.org

8.2.1 Relevant Tools and their APIs

The set of tools available to education providers is broad, but not all tools are relevant for our purposes. An extensive set of tools focuses on content creation, which is predominantly out of the scope of PEPML due to its abstraction level. In the following, we mainly focus on LMSs, project management tools and survey tools because they provide information relevant to PEPML.

Most education programmes use a form of LMS to manage participants, distribute content, perform assessments or handle submissions. LMSs are powerful tools, but their features differ significantly [CZ14]. According to WebTechSurvey[3], the open-source LMS Moodle[4] has a market share of nearly 33%. On the G2 Grid for open-source LMSs[5], Moodle is only outperformed by Canvas LMS[6]. Both LMSs provide an API, but the documentation for Moodle's API is sparse, and it does not follow modern API guidelines. However, both APIs allow retrieving participant information, course structures and other information relevant to PEPML. The landscape of commercial LMSs is richer, but on the G2 grid, only Google Classroom[7] outperforms Canvas LMS. Google Classroom has a more focused feature set than Moodle and targets school settings. Nonetheless, it provides a comprehensive and well-documented API allowing CRUD operations on information about courses, participants and materials. As these three examples show, wide-spread LMSs offer APIs providing access to information relevant to PEPML.

Learning Management Systems

The Experience API (xAPI) [Rus23a] is a project to track learning across multiple tools. Every learning event is captured as a statement consisting of a noun, a verb and an object, e.g. "Tessa submitted thesis". Such statements are sent to a Learning Record Store (LRS), which can share this information with other LRSs. Being able not just to track events within a single LMS but across learning tools is very powerful and can provide additional insights. Specific events, like participant registration or group assignments, can be relevant for PEPML models. While xAPI is out of scope for the moment since it provides a different kind of interaction paradigm than traditional APIs, it can be relevant in the future. Also, the xAPI support is limited in the LMSs presented above. Moodle and Canvas LMS have partial support for xAPI. Google Classroom currently does not support it at all.

xAPI

[3] https://webtechsurvey.com/technology/moodle
[4] https://moodle.com
[5] https://www.g2.com/categories/learning-management-system-lms#grid
[6] https://instructure.com/canvas
[7] https://edu.google.com/workspace-for-education/classroom

PEPML's *Work* package allows for representing tasks in PEPML models and attaching them to program entities. The task information is intentionally slim in PEPML, and it is intended to be complemented with information from other sources, such as project or task management tools. Project management tools have a higher emphasis on APIs than LMSs since they require a tighter integration into a company's workflow. Without an API and support for integration with other applications, project/task management tools risk staying competitive. Hence, obtaining information about tasks via an API is possible with all major project/task management tools. In the next section, we show the feasibility of integrating the information provided by a project management tool. For our feasibility study, we use OpenProject, an open-source project management tool with a well-designed and well-documented API.

Project/Task Management Tools

Other tools that can be relevant for designing and managing professional education programmes are survey tools like Qualtrics[8], Survey Monkey[9] or LimeSurvey[10]. The first two are commercial tools, while the latter is an open-source tool. The API of LimeSurvey is limited, but the commercial example tools provide comprehensive APIs, e.g. to obtain survey results or set up recipient lists for a survey. These tools can provide evaluation results that can be linked to PEPML models. Also, the information in a PEPML model can be used to set up surveys for evaluating a programme.

Survey Tools

8.2.2 Feasibility Study: OpenProject

In this section, we show the feasibility of integrating the information provided by the project management tool OpenProject into PEPML models. We also present how the linkage of tasks to PEPML entities can be used to create a consistent task structure within OpenProject. At the end of this section, we distil the requirements a project or task management tool has to fulfil to be used for such a purpose.

Retrieving Relevant Information

During the first step of the process depicted in Figure 8.2, we must identify which information provided by OpenProject is relevant for PEPML models. As part of the second step, we defined the metamodel shown in Figure 8.3, which this section uses as an example.

[8] https://qualtrics.com
[9] https://surveymonkey.com
[10] https://limesurvey.org

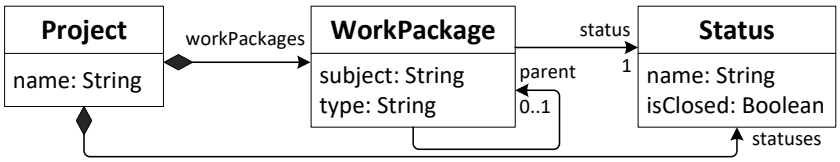

Figure 8.3: Metamodel for information provided by OpenProject

Each project in OpenProject can have work packages, which can be structured hierarchically. Types for work packages can be defined, e.g. *Task* or *Feature*. A work package has a lot of additional properties that are not part of PEPML, e.g. due dates or assignees. These properties are not shown in Figure 8.3 because this information is not used within PEPML. Like PEPML's *Work* package (see Section 4.2.2), OpenProject distinguishes between open and closed statuses. As part of the third step, we developed an extractor for OpenProject, available on GitHub[11]. This extractor is currently focused on the metamodel shown in Figure 8.3 but can be extended if additional information is considered relevant.

Mapping to PEPML

Listing 8.1 shows the GenTGG mappings necessary to map information from OpenProject to a PEPML model. These mappings are developed as part of the fourth step of the process shown in Figure 8.2. The mappings are mostly straightforward since the metamodel defined for OpenProject is structured similarly to the relevant part of PEPML's metamodel. Statuses have an equivalent class in PEPML. Every task within PEPML is mapped to a work package of this type. Moreover, PEPML entities are represented as work packages of type *Feature* in OpenProject.

A correct rule application order during transformation is critical. In PEPML, the classes `Task` and `Status` are both subclasses of `Entity` (see Section 4.2.2). Hence, the rules for transforming statuses and tasks must be ranked higher than the rule mapping entities to work packages. Otherwise, tasks and statuses could be treated as entities and be transformed into features.

The mappings shown in Listing 8.1 enable the import of tasks and statuses from OpenProject into a PEPML model. We interpret tasks not yet associated with a specific entity as general tasks within the scope of a programme. Within PEPML, users can associate tasks with entities, define task constraints or manipulate statuses. The latter can also be done within OpenProject.

[11] https://https://github.com/dwolters/openproject-pepml-integration

Listing 8.1: GenTGG mappings between OpenProject's and PEPML's metamodel

```
1   Project <=> EducationProgramme
2   -statuses <=> -entities
3   -workPackages <=> -entities
4
5   Status <=> Status
6   .name <=> .name
7   .isClosed <=> .isClosed
8
9   WorkPackage{.type="Task"} <=> Task
10  .subject <=> .name
11  <-parent <=> <-associatedTo
12  -status <=> -hasStatus
13
14  WorkPackage{.type="Feature"} <=> Entity
15  .subject <=> .name
16  -status <=> -hasStatus
17  <-parent <=> <-associatedTo
```

A PEPML model generated based on these GenTGG mappings can be merged with other PEPML models in the same fashion as we merge models from multiple Miro boards (see Section 6.2.5). Within OpenProject, we perform CRUD operations on tasks, and the changes can impact what is modelled on the online whiteboards, e.g. a task is shown on an online whiteboard, and the status is changed within OpenProject. When merging, we store the sources of all information in the merged PEPML model. We can use this to propagate changes to other sources. Changes are only propagated if the values provided by other sources match the previous value in OpenProject. Otherwise, users have to be made aware of conflicting values.

OpenProject's webhooks can be used to trigger the integration of new data. The webhooks are invoked with a customizable delay. Consequently, an individual change is not propagated right away but may be bundled together with other changes that occurred before the webhook is invoked.

From a PEPML Model to OpenProject

The mappings in Listing 8.1 can also be used for BWD and fill OpenProject based on a PEPML model. The current version translates all entities into work packages of type *Feature*. If only entities with associated tasks shall be translated into features, we must to add the `associatedTo` association to the pattern of the `Entity` class in Line 14.

To inject information back into OpenProject, we must determine what information changed and update OpenProject accordingly. The procedure to determine changes is similar to injecting changes to an online whiteboard. We compare the newly generated OpenProject model to the current OpenProject model (see Figure 8.4). We use custom properties to annotate which nodes and relationships have been added, updated or deleted. Based on these annotations, we can propagate the changes to OpenProject by calling the relevant API functions.

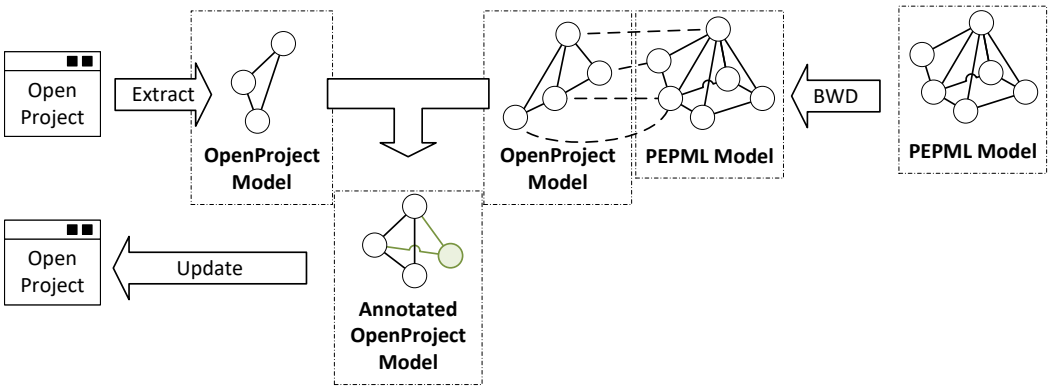

Figure 8.4: Propagating changes back to OpenProject

Task Constraints

The *Work* package, which we introduced in Section 4.2.2, not only allows attaching tasks to entities but also enables users to define task constraints. A task constraint ensures that a transition to a specific status is only possible when the constraint is fulfilled. We illustrated this with the example in Figure 4.7, where a task constraint ensures that the task "Agree on programme objectives" can only be closed if all learning objectives have a closed status, e.g. "Done" or "Rejected". An object diagram showing the abstract model at the beginning of the sequence is shown in Figure 4.6. In the following, we explain the interplay with OpenProject and how executable task constraints can be defined.

Figure 8.5.a shows how a status change in OpenProject is propagated to PEPML's tooling via a webhook. This webhook transmits the information about all status changes since the last webhook invocation. Based on this information, relevant task constraints are evaluated. As in the example in Figure 4.7.a, we assume that the constraint evaluates to false, and the task is put into the status "Tentative". In Figure 8.5.b, the learning objective

violating the constraint is set to a closed status, and then the task is again set to "Done". We assume for this example that both status changes happen within the delay of the webhook, resulting in only one webhook invocation for both changes. Now the constraint is fulfilled, and the task remains in status "Done". The statuses in the PEPML model are changed accordingly.

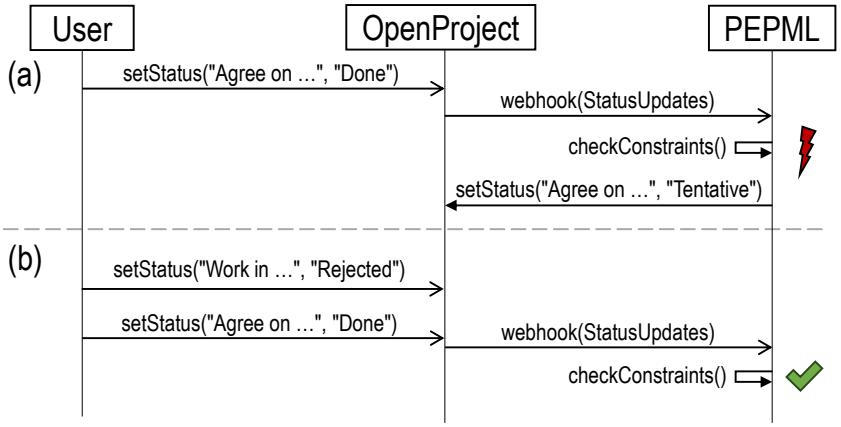

Figure 8.5: Example of a task constraint validation in combination with OpenProject

Figure 4.6 features a natural language description of a task constraint. Listing 8.2 contains a Cypher query that can be used to evaluate this constraint automatically. Line 1 gathers all objectives with a non-closed status. Line 2 removes objectives defined on a component level since the constraint shall only apply to programme-level objectives. Line 3 then returns the identifiers of objectives with open statuses. If the result is not empty, the constraint is not fulfilled, and the task has to be put into the failure status.

Listing 8.2: Cypher query to determine if any programme-level objective has an open status

```
1  MATCH (o:PEPML__Objectives {enamespace: $modelName})<-[:hasStatus]-(s
       ↪ :PEPML_Status {isClosed:false})
2  WHERE NOT EXISTS {(o)<-[:hasObjective]-(c:PEPML__EducationComponent {
       ↪ enamespace: $modelName})}
3  RETURN id(o)
```

We do not suggest defining executable constraints for all tasks since it requires technical expertise and additional effort. However, this investment is justifiable for important tasks or those reused in other programmes because it leads to higher-quality programmes. For reusing tasks, we present an approach to store tasks required in specific situations in Chapter 9. These tasks can then be recommended to education providers when these situations occur.

Requirements for Integration of Project Management Tools

OpenProject is just one example of a project management tool that could be integrated with PEPML. A project management tool must meet three main requirements to be integrated with our approach:

(1) A concept of tasks and subtasks must exist.
(2) Tasks must have a status.
(3) An API to read and manipulate tasks and statuses must exist.

Requirement (1) should be fulfilled by nearly every project management tool. In cases where very rudimentary to-do list apps are used for project management, the subtasks feature may not exist. Requirement (2) varies in the degree of fulfilment between project management tools. Any project management tool that supports tasks allows closing tasks. However, it may not always be the case that one can differentiate between statuses other than "open" and "closed". Nonetheless, integration is still feasible. Just fewer options for statuses exist. Requirement (3) is likely to be fulfilled by all project management tools with a sufficient user base. However, for smaller project management tools or simple to-do list applications, an API may not be provided.

8.3 File-based Integration

In the following, we explain how tools with file-based data storage can be used as sources to add information to PEPML models. As part of this, we mention file formats relevant to education programmes and discuss file format conversion options. Afterwards, we use spreadsheets to show the feasibility of integrating file-based information into PEPML models.

8.3.1 File Format and Structure

File Format Processing

The file format has a significant influence when tools provide information as files. Working with proprietary file formats, especially binary ones, is difficult if they are not documented, and no libraries exist to process files in such formats. Such libraries typically exist for well-known open or standardized formats. Furthermore, plenty of custom file formats are based on JSON or XML. If no library exists to process the custom format, developers can at least work with the libraries for the underlying one.

Only a few widely-known file formats exist for Instructional Design, and those that do exist are very low-level. For instance, SCORM [Rus23b] allows the packaging of e-learning content so that it can be exchanged between different LMSs. A SCORM package can be seen as a low-level representation of an education component. Information about the sequence of activities could be extracted from a SCORM package. IMS-LD [IMS03] is theoretically a relevant format since it includes information relevant to our purposes, but the format is not supported by any major LMS. Moodle describes an XML format for quiz questions and glossaries[12]. Otherwise, Moodle and other LMSs allow file exports, typically in the CSV or another spreadsheet format. Spreadsheets or other office documents are often used to describe information about learning goals or the structure of programmes or individual components.

File Formats

Depending on the file format and how it is used, we can distinguish between structured, semi-structured and unstructured information. We consider every file where we can rely on the structure and know its semantics as a structured file. A BPMN model provided as an XMI file is an example of a structured file. Similarly, spreadsheets can be fully structured if the semantics of the different columns and rows are known, and no cell contains unstructured information. What is considered semi-structured or unstructured can vary. For this thesis, we call files semi-structured when we can only partially rely on a structure. A Word-based template to describe an education component would be an example of a semi-structured file. We consider such files to be unstructured if there is no predefined structure or we cannot rely on it.

Structuredness

We must proceed differently depending on how much we know about the structure of a file and to which degree we can rely on it. For structured files, we can proceed analogously to information extracted from APIs. For both unstructured and semi-structured information, we need to handle the unstructured parts. Natural language processing can extract structured information from the unstructured parts. However, this requires extensive expertise. Possible alternatives are existing Large Language Models (LLMs), like ChatGPT[13], that could extract structured information from unstructured parts. For this, the intended file format and the unstructured information must be provided as a prompt. While we can check on a syntactical level if the output of the LLM is valid, it is not guaranteed that the correct information is extracted. In this thesis, we focus on structured information, and handling semi-/unstructured information is subject to future work.

Handling Unstructured Information

[12] https://docs.moodle.org/403/de/Moodle_XML-Format
[13] https://chat.openai.com

8.3.2 File Format Conversion

As explained in Section 8.1.2, file conversion can help to integrate information from a more diverse set of file formats than just those for which transformations to PEPML have been developed. In the following, we first mention examples of relevant conversion tools and then refer to an approach to automatically select a suitable converter or even converter chains.

File Format Converters

A broad range of conversion tools exist for standardized or widely-known file formats. Among the most versatile file conversion tools is Pandoc[14]. It can convert files between over 40 different formats. It converts files purely on a structural level without requiring to know the semantics of these structures. The open-source office suite LibreOffice[15] allows file conversion between supported formats. The project unoserver[16] goes a step further and provides a web service for office document conversion based on LibreOffice. The tool Rapper[17] can convert between different RDF serialization formats like RDF+XML or Turtle. Even more powerful for RDF conversion is Apache Jena[18], which also supports JSON-LD, a variant of the JSON format for linked data. VLETools.com[19] introduces a lightweight custom format to describe quizzes and glossaries and provides a converter to a Moodle XML file.

Automatically Determine Conversion Chains

In [WHKE17], we presented an approach to extract semantic data from websites, automatically explore conversion options, and determine all services that can process this (converted) data. The service and converter repository and the technique to automatically discover suitable services based on data conversion chains can be reused for this thesis. Every tool to transform a specific file format into a PEPML model can be registered as a service. Based on information in a given format, the approach can identify if a transformation service exists for this format. If not, the approach can explore whether the information can be converted into a format for which a transformation service exists. The following section explains how to use spreadsheets as sources to add information to PEPML models. We also discuss the role of converters in handling different spreadsheet formats.

[14] https://pandoc.org
[15] https://libreoffice.org
[16] https://github.com/unoconv/unoserver
[17] https://librdf.org/raptor/rapper.html
[18] https://jena.apache.org
[19] https://vletools.com

8.3.3 Feasibility Study: Spreadsheets

Spreadsheets are used in nearly every domain for managing information. While spreadsheets force users to organize information in columns and rows, their meanings can significantly differ. Yet, row-based information organization is a typical form of using spreadsheets. Except for a header row, each row represents a specific type of entry, and each column defines a particular property of this entry. The header row provides names for the respective properties. Typical examples of information organized in rows are file exports, e.g. participant lists or assessment results. In the following, we show the feasibility of integrating such information into a PEPML model.

Several different formats for spreadsheets exist. The most common are Excel Workbooks (*.xlsx*), the 97-2003 Excel Workbooks (*.xls*), Comma-separated Values (*.csv*) and the Open Document Format for Spreadsheets (*.ods*). The open-source office suite LibreOffice can be used to convert between these formats. Consequently, providing separate extractors or direct transformers for each format is unnecessary.

While conversion between different formats is possible, there are differences in their capabilities. We focus on their commonalities to ensure proper conversion. We developed an extractor for Excel Workbooks that is available on GitHub[20]. This extractor focuses on workbooks where the first row of every sheet provides names for each column and where the remaining rows represent the information. Furthermore, we only consider plain values (no formulas) in table cells and assume that there are no merged cells[21]. While this sounds restrictive, such spreadsheets are very common, especially when information is exported as a spreadsheet. The extractor automatically derives a metamodel for the information stored in the spreadsheet and extracts the contained information into a compliant model stored in Neo4j. The following provides a concrete example illustrating the use of our extractor and how the extracted model can be transformed to PEPML using TGGs.

We use the Excel Workbook shown in Table 8.1 as an example to illustrate the feasibility of our approach. The workbook contains two sheets. The first sheet is called “Participants” and contains a list of programme participants and the project group to which they belong. Each participant only appears once on this sheet. Since multiple participants can be in the same group, groups can appear multiple times on this sheet. The second sheet is called

[20] https://https://github.com/dwolters/excel-pepml-integration
[21] Merged cells span multiple rows or columns.

"Assignments" and contains a list of project groups and the project topic they are working on. On this sheet, groups only appear once, but projects can appear multiple times. How often information appears in a spreadsheet is important when we design the TGG to transform it into a PEPML model.

Since the first row of each sheet provides the names of the data contained in the respective column, we use these names to access the columns instead of their index (A, B, C, ...). Excel proceeds similarly when an area is explicitly defined as a table. Our extractor for spreadsheets creates a node in Neo4j for the workbook, each sheet and every row except the first one. We use a lower camel case notation to deal with blanks in column names. For instance, the value of the column "Project Group" would be available as the property `projectGroup` on the respective node.

Table 8.1: Example Excel Workbook consisting of two sheets

Sheet: *Participants*

Name	**Project Group**
John	Group 1
Marie	Group 2
Frank	Group 1
Mitch	Group 2
Emma	Group 3
Tessa	Group 3

Sheet: *Assignments*

Group Name	**Project Title**
Group 1	Scrum vs Kanban
Group 2	Scrum vs Kanban
Group 3	Scrum vs SixSigma

Our extractor is generic and can handle any Excel Workbook that fulfils the constraints mentioned earlier (header row in every sheet, no formulas, no merged cells and data as rows). Instead of developing a metamodel manually for every spreadsheet, the extractor can generate a metamodel automatically. The upper part of Figure 8.6 shows the generic part of our metamodel for spreadsheets. The subclasses of `Row` are generated based on a sheet's name and columns defined in the sheet's header row (see Table 8.1).

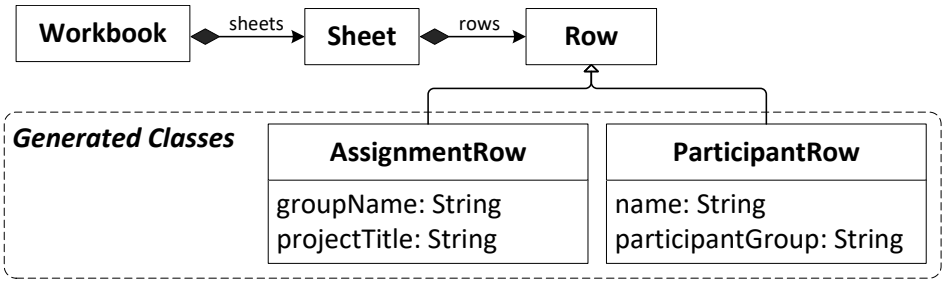

Figure 8.6: Metamodel for example Excel Workbook in Table 8.1

Listing 8.3: GenTGG mappings for the example Excel Workbook

```
1 Workbook <=> EducationProgramme
2
3 ParticipantsRow{<-rows-Sheet?{.name="Participants"}} <=> Participant
4 .name <=> .name
5 .projectGroup <=> -inGroup->Group!.name
6
7 AssignmentsRow{<-rows-Sheet?{.name="Assignments"}} <=> Group
8 .teamName <=> .name
9 .projectName <=> -worksOn->Project?.name
```

Listing 8.3 shows the GenTGG mappings to convert our example Excel Workbook into a PEPML model. Line 1 maps `Workbook` to `EducationProgramme`. The resulting rule creates an empty object of `EducationProgramme`. Additional information is added when the PEPML model is merged with other PEPML models. An `EducationProgramme` object is necessary to attach entities to a programme properly.

Since certain information appears multiple times, we have to use modifiers to avoid duplicate creations of entities. Every `ParticipantsRow` is transformed into `Participant`. For plain unidirectional transformation, the pattern in Line 3 could be omitted because the class `ParticipantsRow` already indicates to which sheet the row belongs. Since the class `Sheet` is not mapped to any PEPML entity, we require this pattern for BWD. The same holds for the pattern defined in Line 7. The "?" operator ensures that we have one rule creating a new `Sheet` object and another rule assuming a `Sheet` object exists. If the operator is not used, only the latter rule is created, and rows would not be added to a sheet during BWD since no rule would create a `Sheet` object.

In Line 5, we exploit that we can map a class property to an associated class's property. By default, an object of the associated class would be created. In this case, we use the "!" operator to state that we expect the object of the associated class to exist. We do this because we create objects of class `Group` based on the other sheet.

Lines 7-8 are analogous to Lines 3-4 and handle the creation of a `Group` object based on a row of the "Assignments" sheet. Since groups are only listed once on the second sheet, there is no risk of creating multiple objects for the same group. Unfortunately, no sheet uniquely lists all available projects, and we have to avoid creating multiple objects for the same project. We use the "?" operator in Line 9 to ensure that we have a rule that creates a project and one that can reuse an existing one.

With the TGG generated based on the GenTGG mappings in Listing 8.3, it is possible to transform the example spreadsheet into a PEPML model and vice versa. For the transformation to work, we have to specify a rule order where the rules that expect an object of `Project` or `Sheet` to exist are ranked higher than those creating new objects of the respective classes. Furthermore, we must specify that the rules creating an object of `Project` or `Sheet` are only applied to individual matches. Such transformation options can be provided to our eNeo.js transformation engine. If these rules were applied to all possible matches at once, we would get a `Project` object for each row of the "Assignments" sheet during FWD or a `Sheet` object for each row during BWD.

We can also serialize a spreadsheet model again into an Excel Workbook. If we do this based on a spreadsheet model obtained based on BWD, we cannot ensure a column or row order since this information is not stored within a PEPML model.

Based on the example, we showed how to enable bidirectional transformation for specific types of spreadsheets. The extractor we developed is generic and not limited to the example presented in this section. The example also shows the power of GenTGG since no manually specified TGG rules were needed besides the GenTGG mappings. Furthermore, it highlights what has to be considered when dealing with sources that do not provide a unique list of certain objects.

8.4 Summary

In this chapter, we explained how we could enrich a PEPML model based on information stored in tools used by education providers to develop and manage professional education programmes. We considered obtaining information from tools based on their API or the files they use to store information. Moreover, we provided examples of applications and file formats containing information relevant to PEPML. Integrating the obtained information is analogous to integrating information from online whiteboards. In two feasibility studies, we showed the integration information from the project management tool OpenProject and a common form of spreadsheet. Thereby, we show the possibility of building up PEPML models from other sources than just online whiteboards. Consequently, education providers are not forced to use PEPML's visual syntax but can still benefit from PEPML models by using them to check consistency, create a portfolio or provide situational context.

Chapter 9

Situation-specific Recommendations

In the previous chapters, we explained PEPML and how PEPML models can be visually modelled in collaborative online whiteboards or by compiling information from other tools used to manage information about professional education programmes. This chapter introduces an approach that takes PEPML models as situational context to provide situation-specific recommendations to education providers to increase the quality of a programme. It addresses Research Question RQ4, concerned with storing and recalling knowledge about designing and managing professional education programmes. Furthermore, it concludes the *Modelling Language Validation* step by showing that PEPML models can provide situational context information.

Section 9.1 defines the requirements, gives a solution overview and discusses related work. Section 9.2 explains how to define situations in which knowledge can be relevant and how these descriptions can be operationalized to automatically determine if a situation occurs. Section 9.3 describes how to define knowledge items that can be stored within our knowledge base.

9.1 Overview

This section provides an overview of our approach to giving situation-specific recommendations to providers of professional education programmes presented in this chapter. It covers the requirements for our approach, a solution overview and related work.

9.1.1 Requirements

The following Recommendation Requirements (RR) are based on Research Question RQ4 and the characteristics of professional education programmes.

RR1 **Knowledge Storage**: *It shall be possible to store knowledge relevant to designing and managing professional education programmes.*

Programme designers and managers acquire new knowledge with every new programme and iteration. This knowledge has to be externalized so that it is not lost when a designer or manager leaves an education provider and to make it accessible to others. Knowledge can be very diverse and relate to the characteristics of a programme, e.g. the product itself, processes, people involved or tools. Aside from this, there can be more general knowledge, like best practices, values or hints.

RR2 **Tie Knowledge to Applicable Situations**: *It shall be possible to tie knowledge to situations in which it is relevant.*

The situation determines which knowledge is relevant. Hence, it is not sufficient to store the knowledge, but it must also be tied to applicable situations.

RR3 **Model-driven Situational Awareness**: *It shall be possible to determine the current situation based on a PEPML model of a professional education programme.*

Designers and managers ideally have an overview of the current situation, but they cannot be queried about every aspect of it to identify which knowledge is relevant. With PEPML, we introduced a language to model professional education programmes. Ideally, situations can be determined automatically based on the model of a programme.

RR4 **Semi-automatic Model Enrichment**: *It shall be possible to ask education providers targeted questions to determine situational information missing from a PEPML model and add the answer to the model.*

Education providers may not know which information in a model is crucial to describe a situation. If certain situational information is absent from a model, education providers have to be asked to provide the missing information. The answers have to be stored within the model of the programme to provide a more detailed model of the situational context.

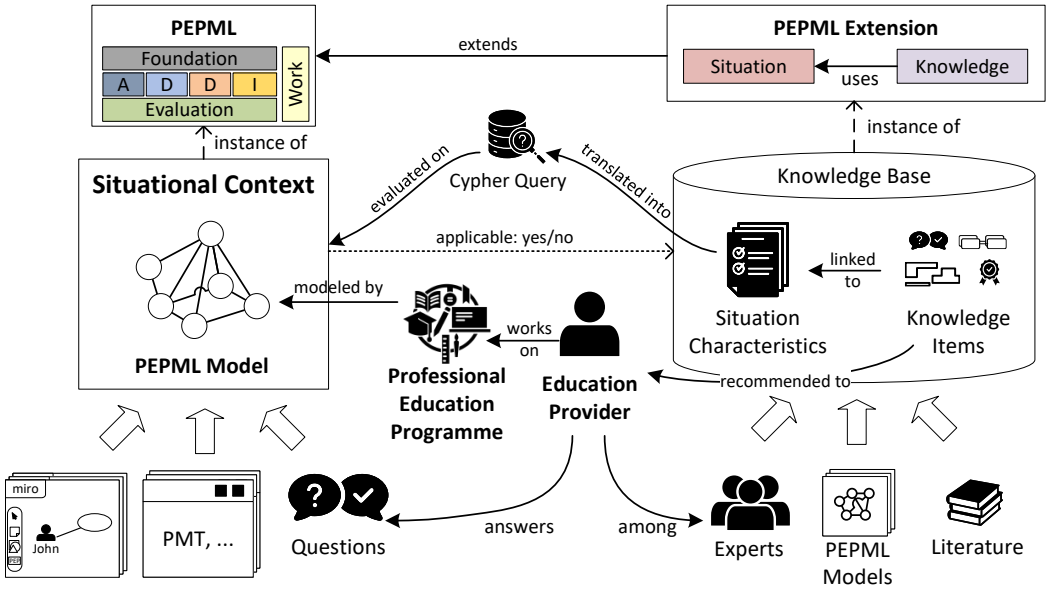

Figure 9.1: Approach to provide situation-specific recommendations

9.1.2 Solution Overview

The following presents our approach for recommending education provider situation-specific knowledge for designing and managing professional education programmes. This approach was first introduced in [WE22b]. A visual overview of the approach is depicted in Figure 9.1.

For our approach to be aware of the situation (cf. RR3), we use PEPML models of professional education programmes as representations of the situational context. As we explained throughout the previous chapters, PEPML models can be modelled using our Miro-based tool support or we can transform information extracted from other tools that contain information about a programme. In this chapter, we introduce a question-based approach as an additional source to semi-automatically add missing information about situational factors to PEPML models (cf. RR4).

PEPML models represent Situational Context

Knowledge about designing and managing professional education programmes is stored in a knowledge base which addresses RR1. We developed two additional PEPML packages to describe the contents of the knowledge base. The *Knowledge* package enables the specification of different types of knowledge items. The most basic is a textual recommendation, e.g. of a best practice. More detailed knowledge items can be described using PEPML. For instance, a PEPML-based knowledge item could be a suggestion for adding a specific education component. The knowledge base can be filled by experts or based on existing PEPML models of other programmes or literature.

Knowledge Items

Each knowledge item can be linked to situations in which it applies (cf. RR2). These situations are described using the *Situation* package. A situation description defines situational factors in terms of specific values for properties, relations between entities or a mixture of both. Such descriptions are translated into queries that are afterwardss evaluated on the PEPML model, serving as situational context. If the query returns a positive result, the attached knowledge items are recommended to education providers.

Determine Relevant Knowledge

For textual knowledge items, users must apply the recommendation manually. In the case of PEPML-based knowledge items, the PEPML model of the professional education programme can automatically be enriched with relevant items when users want to follow the recommendation. These can be content elements like education components or method parts like tasks. In the case of tasks, we can support their enactment through our integration with project management tools (see Section 8.2.2) and task constraints (see Section 4.2.2).

Applying Knowledge

If the situational characteristics of a question-based knowledge item apply, users are asked to answer the respective question. Answers are directly added to the PEPML model. Afterwards, the model can be queried again to identify if the added information allows any further recommendations. Question-based knowledge items can be defined to ensure that important situational factors are set within the model, e.g. the size of the audience.

Semi-Automatic Model Enrichment

In Section 9.2, we explain how to describe situations and derive queries that can be executed on PEPML models that serve a situational context. Subsequently, in Section 9.3, we explain how to describe knowledge items that can be suggested in applicable situations.

9.1.3 Related Work

Related work for this chapter covers Situational Method Engineering (SME) approaches and knowledge bases for Instructional Design.

Henderson-Sellers et al. [HSRÅR14] summarize the field of SME and present an approach for software engineering in general. Fazal-Baqaie [FB16] also presents an SME approach for general software engineering that follows the idea of service-oriented computing. Geisen [Gei15] uses a feedback loop to adapt a software engineering method to the current situation. The approach from Spijkerman [Spi15] allows the development of domain-specific software engineering methods. It builds upon the meta-method MetaMe from Sauer [Sau11], enabling the description of such methods. The SME approach of Grieger [Gri16] focuses on software migration projects, and Jovanovikj [Jov20]

Situational Method Enginnering

on co-migrating the tests in migration projects. SME approaches also exist outside of software engineering. Gottschalk [Got22] developed an SME approach for business model development methods. Tsai and Zdravkovic [TZ21] present an SME approach for digital business ecosystems. The approach presented in this chapter follows the core ideas of SME: Store existing knowledge in a repository, identify situations in which the knowledge is applicable and assist in its application. However, our approach focuses on making recommendations while a programme is being designed/managed rather than composing an engineering method upfront. Hence, we rather follow an incremental approach to method engineering.

Beyond Software Enginnering

We need a way to store knowledge about designing and managing professional education programmes to give situation-specific recommendations. Ontologies are often used to organize knowledge by describing concepts and relationships between them. Rahayu et al. [RFK22] describe different use cases for ontologies in Instructional Design. While the focus of their particular study is on using ontologies for the personalization of e-learning, they list modelling of the curriculum, data integration, and descriptions of the domains and learning activities as further use cases. These further use cases are the focus of PEPML models. We use PEPML models to provide recommendations to education providers. Vidal-Castro et al. [VCSP12] developed an ontology to store information about Instructional Design methods so that this information can be used in supporting tools for Instructional Design. Unfortunately, their ontology is not openly available, but it could have been used as an information source for the approach presented in this chapter. Psyché et al. [PBNM05] developed an approach that links Instructional Design theories with learning designs. They use IMS-LD descriptions as context to give recommendations based on their ontology. As mentioned in Section 3.3, IMS-LD has not gained traction in practice. Furthermore, in contrast to PEPML, IMS-LD is on a lower abstraction level and does not cover all life cycle phases.

Ontologies for Instructional Design Knowledge

Aside from using ontologies to represent knowledge, solution patterns[1] are often used to describe solutions to reoccurring problems. In software engineering, solution patterns like those of the Gang of Four [GHJV95] describe reusable structures that can be applied in specific situations. Similarly, solution patterns also exist in the domain of Instructional Design [FU02, Ped12]. While ontologies can be used for pure knowledge organization, solution patterns couple existing solutions to specific problems. Hence, existing solution pattern

Solution Patterns

[1] The correct term is just "pattern", but we already use the term "pattern" in this thesis to refer to graph patterns. Therefore, we use "solution pattern" to distinguish these terms.

collections can be used to fill the knowledge base presented in this chapter. The programme designers or managers typically determine if a solution pattern fits the current situation. With our approach, we can complement this by using PEPML models to determine applicability. The occurrence of a specific problem alone does not necessarily lead to a solution pattern application. There can be multiple solution patterns for the same or similar problems and applying specific solution patterns results in a trade-off. The implications, positive and negative, of applying a solution pattern are part of its description. Therefore, we recommend this information to users of our approach and let them decide if they want to apply it.

9.2 Describing Situations

We couple knowledge to certain situations to only recommend it when it is relevant (cf. RR2). Such situations are described by defining the situational factors that have to be present in a PEPML model. In the following, we explain how situations can be described and how these descriptions can be operationalized to determine automatically if a situation occurs (cf. RR3).

9.2.1 Metamodel to Describe Situations

We enable the description of a situation with the PEPML extension shown in Figure 9.2. PEPML itself is used to describe situations by defining `Situation` as a subclass of `EducationProgramme`. Hence, any PEPML model can be interpreted as a specification of a situation with concrete characteristics, e.g. entities with specific property values and relations. Conditions can be used to define more flexible situations. These can define constraints on entities or relations, like defining a property value to be in a certain range. The custom `forbid` property denotes that the respective entity or relation shall not be present. If an entity is marked as forbidden, all relations pointing to this entity must be forbidden as well since we cannot have dangling edges.

PEPML-based Situation Descriptions

Describing situations with PEPML, forbidden elements and conditions does not require deep technical expertise and can already describe a broad range of situations. More technically advanced users can also define parts of the Cypher query manually using a query condition. Thereby, they can leverage the full expressiveness of Cypher and can describe any situation that a PEPML model can represent.

Situation Descriptions based on Partial Queries

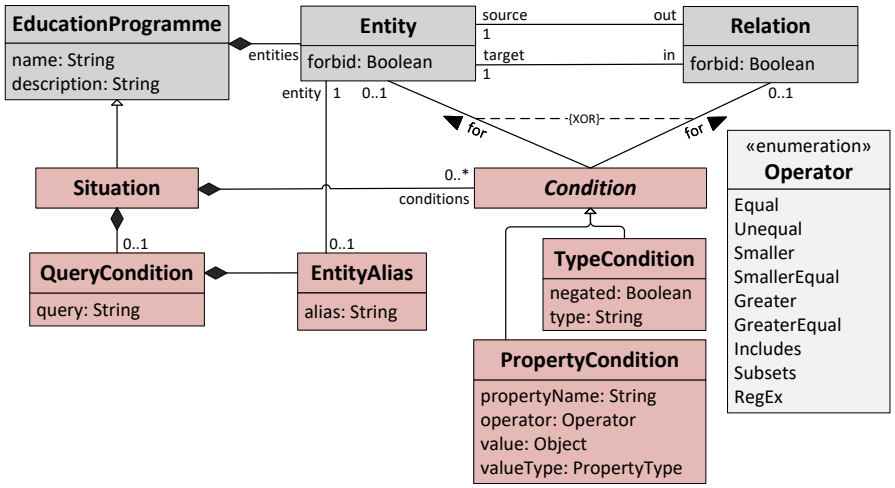

Figure 9.2: PEPML extension to describe situations

Figure 9.3 depicts an example of a situation description in the abstract syntax. We use the «forbid» stereotype to depict elements where the property `forbid` is set to `true`. The same kind of notation is used within graph transformation approaches like Henshin [ABJ+10] or TGG [WA21]. The example uses an instance of `EducationComponent` to describe a teaching/learning component that is delivered synchronously online and contains a group exercise. The condition specifies that the exercise includes multiple groups. Using the «forbid» stereotype, we denote that no tool has been assigned to the exercise.

Example Situation Description

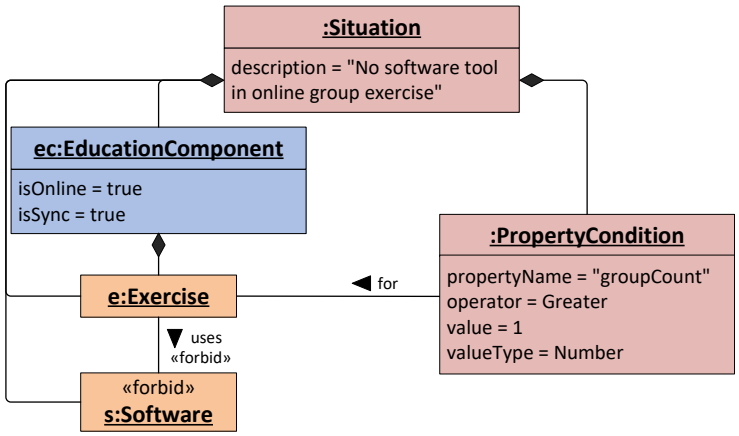

Figure 9.3: Example situation describing that no software tool has been assigned for an online group exercise

When determining if a situation occurs, we follow the Liskov substitution principle [LW94] and assume that any instance of a subclass can also be interpreted as an instance of its superclass. The class `EducationComponent` has two subclasses `EducationComposite` and `TimePlanningComposite` (see

Liskov Substitution Principle

Section 4.2.4). Any instance of these subclasses could also represent the education component in Figure 9.3. We can use type conditions to disable the effects of the Liskov substitution principle. In the case of the example, we could add a negated type condition to exclude subclasses of `EducationComponent`.

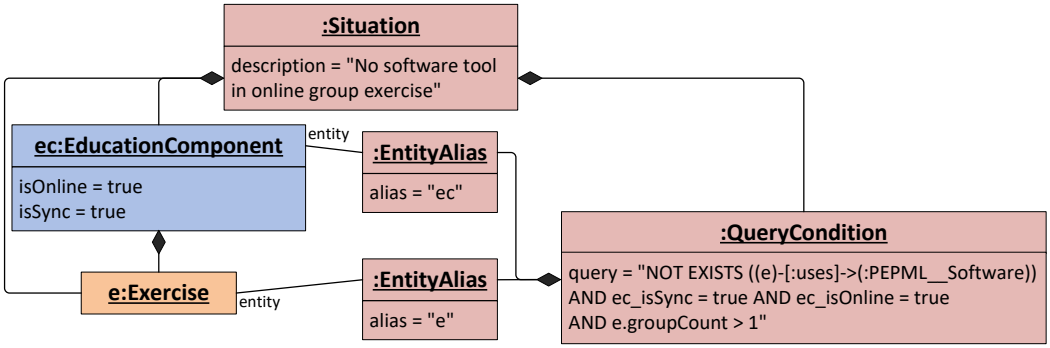

Figure 9.4: Equivalent situation description to Figure 9.3 using a `QueryCondition`

Query Condition Example

Figure 9.4 defines the same situation as in Figure 9.3 but uses a query condition instead of forbidden entities and a property condition. A query condition solely defines the WHERE clause of a Cypher query. Each entity is associated with an instance of `EntityAlias` to specify which names are used for these entities within the query.

9.2.2 Operationalization of Situation Descriptions

Operationalization to Cypher queries

Situation characteristics described as instances of the metamodel depicted in Figure 9.2 are operationalized by translating them into Cypher queries. These queries are evaluated on PEPML models and if the returned results are not empty, the described situations occur.

MATCH Clause Generation

During operationalization, we must differentiate between the regular PEPML parts[2], forbidden elements, inherited property values and conditions. We explain the operationalization based on the example given in Figure 9.3, which is translated into the Cypher query shown in Listing 9.1. All non-forbidden PEPML parts are listed in the MATCH clause of the generated query. The `enamespace` property ensures that the entities belong to the same model. Non-inherited property values could be listed alongside the `enamespace` property. The example only uses properties for which the values are inherited from ancestor elements if not set by the entity itself. OPTIONAL MATCH clauses, like in Line 2, are added to gather inherited values.

[2]With regular PEPML parts, we refer to instances of classes described by the metamodel presented in Chapter 4 that are not tagged with the «forbid» stereotype.

Listing 9.1: Operationalized Situation Description

```
1  MATCH (ec:PEPML__EducationComponent {enamespace: $modelName})-[:
       ↪ contains]->(e:PEPML__Exercise {enamespace: $modelName})
2  OPTIONAL MATCH (ec)<-[:contains*]-(ec2:PEPML__EducationComposite {
       ↪ enamespace: $modelName})
3  WITH ec, e, coalesce(ec.isSync, head(collect(ec2.isSync))) as
       ↪ ec_isSync, coalesce(ec.isOnline,head(collect(ec2.isOnline)))
       ↪  as ec_isOnline
4  WHERE NOT EXISTS ((e)-[:uses]->(:PEPML__Software))
5  AND ec_isSync = true AND ec_isOnline = true AND e.groupCount > 1
6  RETURN id(ec) as ec, id(e) AS e
```

WITH Clause Generation

The WITH clause aggregates the results and provides access to entities and inherited property values (see Line 3). If query conditions are used, the names assigned to the entities in the WITH clause are based on the assigned aliases. Without query conditions, internal names suffice since the remaining clauses are also generated.

WHERE Clause Generation

Forbidden elements and conditions are converted into a WHERE clause. The former leads to adding embedded subqueries that shall return an empty result (see Line 4). The latter are regular conditions on property values. For properties with inherited values, it is crucial to use the aggregated value specified in the WITH clause and not the property provided by the entity. This is similar to the queries generated for decorators (see Section 6.2.7). If query conditions are used, the WHERE clause is fully specified by the query condition.

Determine Situation Occurence

Once a query is generated for a situation description, it can be evaluated on a given PEPML model. The query returns the identifiers of the PEPML entities that are supposed to exist in the described situation. Hence, every result describes an occurrence of this situation. The following section explains how knowledge items can be attached to situations and how these identifiers can be used to attach additional elements to the identified entities.

9.3 Knowledge Base Items

In this section, we explain how we describe knowledge relevant to designing and managing professional education programmes. The metamodel shown in Figure 9.5 defines three ways to describe such: Textual descriptions, questions and PEPML models. These types are all knowledge items that can be attached to relevant situations. In the following, we explain each type.

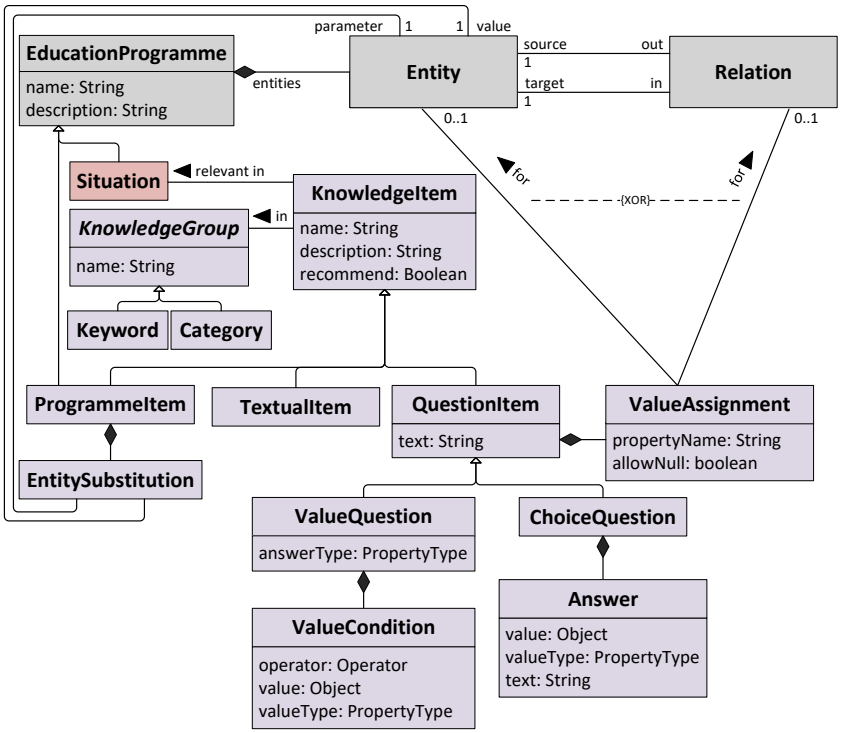

Figure 9.5: Metamodel to describe situation-specific knowledge

9.3.1 Textual Knowledge Items

The most basic type of knowledge item is a textual description of the relevant knowledge. It can provide general hints or recommendations without defining concrete additions/changes to a PEPML model. For instance, Instructional Design patterns, like those presented in [Ped12], could be represented by such items. Subclasses of `TextualItem` can be added to prescribe a structure, e.g. to add fields for describing benefits and drawbacks when following this recommendation or to refer to literature for further information.

Figure 9.6 shows two examples of textual knowledge items relevant to the situation described in Figure 9.3. One item recommends the usage of Miro to gather results of online group exercises and the other one emphasizes that the task description needs to be easily accessible when groups move to breakout rooms. Both examples are based on our own experience of running online group exercises. Subsequently, we show how the knowledge item on the left of Figure 9.6 could be represented based on PEPML.

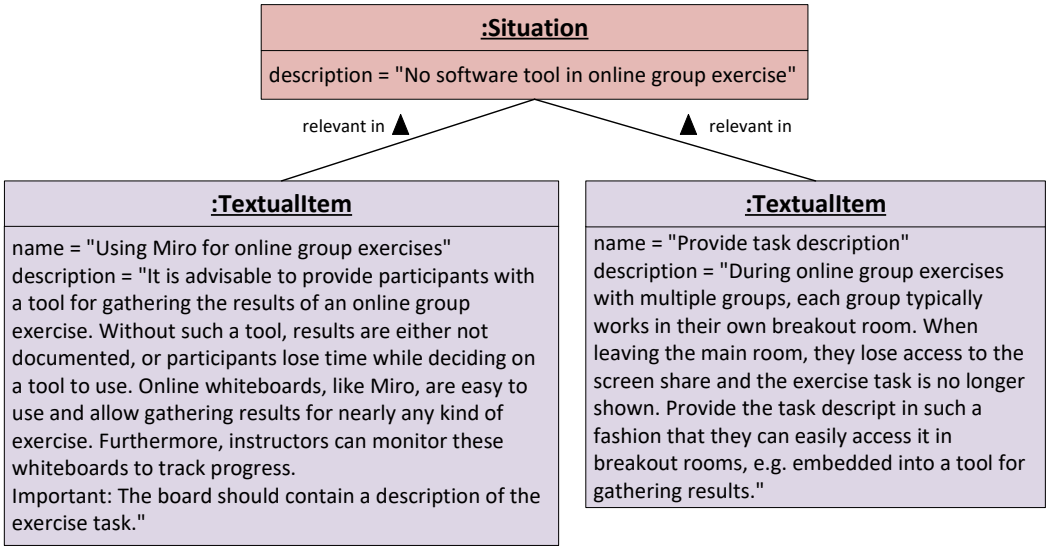

Figure 9.6: Examples of textual knowledge items

9.3.2 PEPML-based Knowledge Items

We leverage PEPML not only to model the situational context but also to describe knowledge items. Such PEPML-based knowledge items are represented by the class `ProgrammeItem`. We distinguish between parameterized and self-contained knowledge items based on PEPML. The former contains parameters that are substituted based on the entities identified by the Cypher query generated for the situation description. A knowledge item without parameters is self-contained. We focus on parameterized PEPML-based knowledge items in the following and mention self-contained items at the end of this section.

Knowledge Items based on PEPML

PEPML-based knowledge items can describe entities that are relevant to a given situation. The query created based on a situation description identifies entities in a PEPML model that describes a professional education programme. These identified entities can be referenced by PEPML-based knowledge items so that we can attach additional elements to them. The entity within the knowledge item is called a `parameter`, and the entity identified by the query is the `value`. The linkage between the `parameter` and the `value` is defined using the class `EntitySubstitution`.

Parameters

Figure 9.7 gives an example of a parameterized knowledge item based on PEPML that could be used instead of the textual knowledge items shown in Figure 9.6. The right part of Figure 9.7 is the same situation description as in Figure 9.3. The left part of Figure 9.7 defines a PEPML-based knowledge item that refers to the exercise identified by the situation description. The knowledge item suggests adding Miro as a software tool as well as tasks to

ensure that the board is prepared and the link is distributed before the exercise. If a user applies this knowledge item, the entities associated with the exercise on the left of the figure would be added to the exercise identified by the query for the situation described on the right. As the example shows, we can define tasks within knowledge items to guide users during enactment. By adding such method parts, we can help education providers develop methods incrementally while designing and managing their programmes.

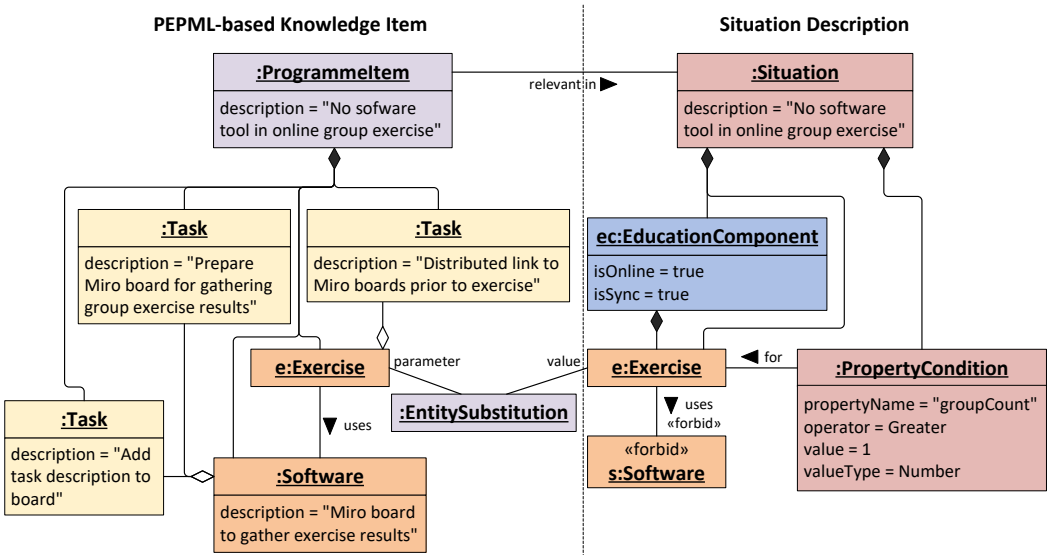

Figure 9.7: Example of a parameterized PEPML-based knowledge item

Self-contained PEPML-based Knowledge Items

When a PEPML-based knowledge item does not define any entity substitutions, it is self-contained. This means that when it is added to a model of professional education programmes, it does not connect to any existing entities. Such items are mainly intended to describe portfolio items, e.g. existing education components that can be incorporated into other programmes.

9.3.3 Questions

Questions to Add Missing Information

Models of professional education programmes evolve and they will not immediately contain values for all relevant situational factors. Education providers may also need to be made aware of which information is important to add. To deal with this, we allow to define questions to add missing information. As shown in Figure 9.5, we allow two types of questions: `ChoiceQuestion` and `ValueQuestion`. The former defines a set of predefined answers and the user can select one of them. The latter offers an input field to provide an answer compatible with the data type specified by the `answerType` property. In the following, we give an example of a value question.

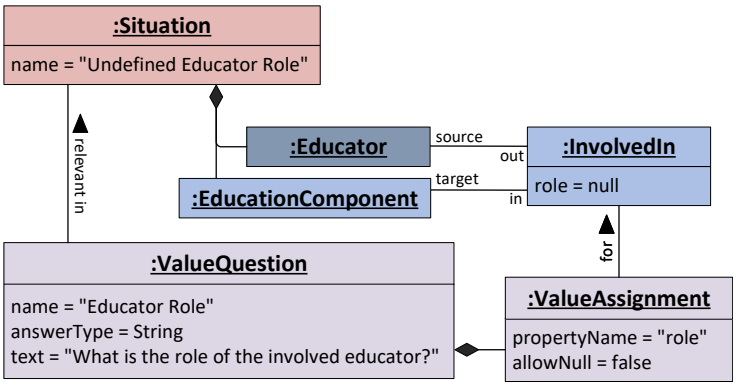

Figure 9.8: Example of a situation-specific question

Figure 9.8 shows an example of a question that is asked when the role of an educator is unspecified. The `answerType` property defines that the answer must be a `String`, which results in a text field being offered to the user. The question also defines a `ValueAssigment`, which allows the storing of the answer within a property of a PEPML entity or relation that is part of the attached situation. In the example, the answer would be stored in the `role` property of the `InvolvedIn` relation between `Educator` and `EducationComponent`. The `answerType` must match the data type of the property to which the value is assigned. The `ValueCondition` can be used to define constraints on the provided value, e.g. using a regular expression to allow only specific strings.

Storing Answers in PEPML Models

Especially in the beginning of developing bespoke professional education programmes, it is vital to ask the right questions. With the approach presented here, we are not only able to define questions but also can tie them to the situation in which they should be asked. Since answers are added to the models, we can semi-automatically enrich the PEPML with missing information, e.g. about situational factors.

9.3.4 Clustering Items

Knowledge about designing and managing professional education programmes is extensive. To allow a better overview, we allow clustering knowledge items into groups. For this purpose, the metamodel in Figure 9.5 includes the class `KnowledgeGroup` and its subclasses `Keyword` and `Category`. Additional subclasses can be added, e.g. the level at which the knowledge item is relevant (programme, component, activity) or programme delivery modes (face-to-face, blended, online). Based on knowledge groups, we can offer users filters and search capabilities.

9.3.5 Recommending Knowledge Items

To avoid overwhelming users with recommendations, we allow classifying knowledge items as recommendations or suggestions. If the `recommend` property of a knowledge item is set to true, it is a recommendation. Otherwise, it is a suggestion. Recommendations are actively presented to the users, while suggestions can be browsed upon request. For instance, if a component spans multiple hours without any breaks, we would recommend adding a break after at most two hours. A suggestion for the same situation would be to offer the audience the possibility to stand up during the longer sessions.

Recommendation vs. Suggestion

Decisions on whether users follow or ignore a recommendation must be stored since we cannot always deduce this based on the PEPML model. We propose to use PEPML's `Decision` class to store decisions about recommendations in PEPML models. As part of this decision, we store the affected entities and do not recommend the same item again for them if the decision has a closed status.

Capturing Decisions

9.3.6 Paths to Filling the Knowledge Base

Figure 9.1 shows the three primary sources for filling the knowledge base: experts, PEPML models and literature. It is out of the thesis' scope to provide a filled knowledge base, but we discuss in this section how it can be filled.

In [GWGK05], Gagné et al. write that each "designer brings to the process [of Instructional Design] his and her understanding of the principles and events that affect learning, and how to best structure instructions." While not every programme designer or manager may be trained in the field of Instructional Design, they experience what works and what does not. Hence, each designer and manager becomes an expert, at least for the programmes they are designing/delivering. The knowledge gained typically evolves around the content of those programmes, the people involved, the processes that were enacted or the resources that were used.

Experts (on their Programmes)

As visualized in Figure 9.9, we propose a reflection phase after every design and delivery phase to externalize important knowledge. On the one hand, designers and managers should reflect on the performed tasks. This reflection can be the basis for identifying important or unnecessary tasks and defining processes for upcoming iterations of a programme or the design of new programmes. On the other hand, reflection on reusable parts of the programme is required. For instance, to identify which education components, objectives or tools can be relevant for the knowledge base.

Reflection after Design and Delivery

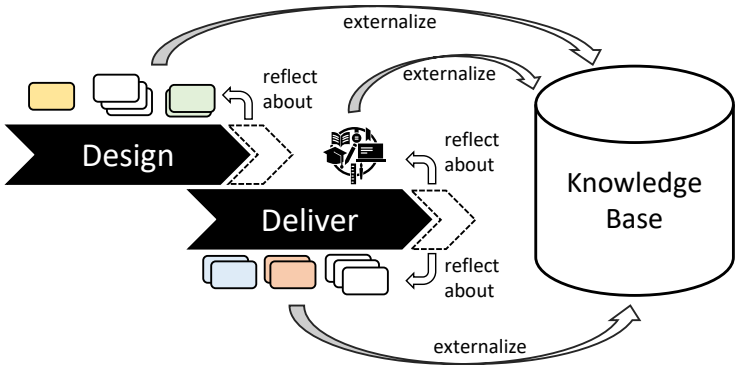

Figure 9.9: Suggested reflection after every design and delivery

PEPML Models as a Source

Externalization is simplified if a PEPML model of the programme exists. Parts of the model can be generalized and directly added to the knowledge base. Due to the integration with project management tools and the possibility of tracking tasks in relation to entities, it can be identified which tasks were necessary for an entity. For instance, we can reuse the education components and the tasks required to manage them.

General vs. Situation-specific Knowledge

The knowledge base can also be filled based on Instructional Design literature. For instance, Mayer's 12 principles of multimedia learning [May17] can be recommended for asynchronous, digital learning or solution patterns from [Ped12] could be recommended for planning activities. However, we must distinguish between general and situation-specific knowledge. If knowledge items are purely attached to types of entities, they are rather general knowledge for this entity type. Knowledge is situation-specific if multiple factors are taken into account, e.g. delivery mode, entity type and topic. In the case of general knowledge, the knowledge items should be marked as a suggestion rather than a recommendation.

9.4 Summary

This chapter introduced an approach to defining knowledge items in the form of questions, textually or based on PEPML. We showed how such knowledge items can be attached to situations in which they are relevant. Applicable situations can be defined using PEPML combined with a self-defined query or conditions defined on PEPML entities. We explained how such situation descriptions can be transformed into queries that can be executed on the PEPML model describing a programme. If a query for a situation description yields a non-empty result, it means the situation can be observed in the model,

and the attached knowledge items are recommended. Also, we showed that knowledge items can be questions to enrich a PEPML model with information about situational factors semi-automatically. We illustrated the approach based on multiple examples.

This chapter focused on providing a technical infrastructure to store and recall situation-specific knowledge (see RQ4). We illustrated that PEPML models can represent the situational context information, which concludes the *Modelling Language Validation* step of the language engineering process used for developing PEPML. Filling the knowledge base is out of the scope of this thesis and subject of future work. PEPML will evolve when the knowledge base is filled since additional properties, entities and relations will be required. Partially, such additions can be made in an ad hoc fashion due to PEPML's extensibility. However, these additions will have to be consolidated frequently, either globally for PEPML or for individual education providers. Furthermore, new types of knowledge items may be necessary in the future, e.g. additional question types or textual items with predefined structures.

Chapter 10

Conclusion and Outlook

This chapter concludes this thesis by first providing a summary of its contributions. Afterwards, we revisit the research questions introduced at the beginning of this thesis and explain how the contributions address these questions. Lastly, we provide an outlook on future work.

10.1 Contribution Summary

Modelling Language for Professional Education Programmes

With PEPML, we introduced a modelling language that enables remote collaborative design of professional education programmes in online whiteboards and provides the basis for work and knowledge management. In contrast to other educational modelling languages, PEPML covers all life cycle phases of such programmes and can track work items in relation to programme elements. PEPML has viewpoints for expressing dependencies between programme elements and the temporal structure of a programme. The language is designed to be extensible by introducing new subclasses, relations between PEPML entities or additional class properties.

Miro-based Visual Modelling Tool Support

Besides a whiteboard-compatible syntax, we introduced visual modelling tool support for PEPML based on the online whiteboard Miro. Thereby, we go beyond using Miro for informal modelling and make it a primary tool for visual modelling. We add features to detect errors in models and use TGG-based model transformations to transform Miro board content into PEPML models. We use model merging to create a joint PEPML model based on multiple sources. Thereby, we enable collaboration across multiple boards and support complexity management by allowing users to split complex models into smaller ones. Due to the usage of TGGs to couple Miro items to PEPML elements, we can support additional visual syntaxes when needed by adding further

TGGs. For example, we introduced a canvas model to plan activities of a PEPML education component. Our tooling is customizable and can be reused to support visual modelling based on Miro for other visual languages as well.

Generating Triple Graph Grammars

The TGGs required for mapping Miro items to PEPML entities are complex and repetitive. With GenTGG, we introduced a language to describe mappings between classes, properties and associations of two metamodels. Based on these mappings and by taking metamodel information into account, we can generate the necessary TGGs based on a far more succinct format compared to manually written TGGs. While GenTGG does not cover the full expressiveness of manually written TGGs, all rules required for this thesis could be generated based on GenTGG mappings. Only the negative application conditions for two rules had to be specified manually. GenTGG significantly reduced the effort for creating the TGGs for PEPML model extraction, and the mappings are easier to maintain due to their succinct format. We evaluated the expressiveness of GenTGG by using it to express existing TGGs used as test cases for eMoflon::Neo [WA21]. This evaluation showed GenTGG's potential beyond its use for PEPML model extraction.

Lightweight TGG Engine

For operationalizing TGGs and enacting model transformation, we developed a new lightweight TGG engine called eNeo.js. In contrast to other TGG engines, it is not coupled to any Eclipse-based tooling, which allows easier integration into other tools, and it is better suited to be used in web-based projects. It also allows TGG operationalization at runtime without having to redeploy the tooling. We use this to enable experienced users to adapt existing TGGs or introduce new ones at runtime.

Versatile Information Integration

Education providers use many tools for designing and managing professional education programmes. The technical infrastructure used to obtain PEPML models based on Miro boards is versatile and can be used to build such models based on information provided by other tools than just online whiteboards. To do this, tools with semantically relevant information must provide an API giving access to the tool's data or store information in an open file format. GenTGG or manually written TGGs can then be used to map a tool's data model or the used file format to PEPML. Based on the (generated) TGG, we can then transform the relevant information into a PEPML model representing this information. The newly created PEPML model can then be merged with existing ones, e.g. those obtained from Miro boards. We provided examples of how information from a project management tool and spreadsheets can be transformed into PEPML models.

We use PEPML to represent situational context information and provide situation-specific recommendations to education providers. These recommendations include textual hints or questions to add missing information or additions to the PEPML models. For instance, content items or method parts for designing and managing professional education programmes can be recommended. Since PEPML models can also be built based on information from tools other than online whiteboards, users do not necessarily need to adopt our visual modelling approach to benefit from situation-specific recommendations. We introduced a knowledge base that can store knowledge tied to situations where it is relevant. These situations are described in such a way that they can be translated into queries that can be evaluated on PEPML models to determine relevant knowledge automatically.

Situation-specific Recommendations

10.2 Research Question Revisited

At the beginning of this thesis, we introduced four research questions. In the following, we revisit each question and explain how it has been addressed.

Research Question 1: How can models be used for collaboration, work organization and situational context while designing and managing professional education programmes?

This research question mentions three purposes (collaboration, work organization and situational context) for which models of professional education programmes shall be used. Our analysis showed that existing educational modelling languages cannot properly be used for all these purposes. In particular, work organization was out of the scope of all existing languages. As a result, we developed PEPML, a modelling language for professional education programmes that specifically addresses these purposes.

We created PEPML with the intention of supporting collaborative design on online whiteboards. Although online whiteboards support real-time collaboration, their visual elements do not have predefined semantics. To address this limitation, we introduced a whiteboard-compatible visual syntax for PEPML specified using TGGs. These TGGs not only impart semantics to specific elements but also enable PEPML model extraction from online whiteboards and the detection of syntactical errors. Besides online whiteboards, we support other tools used for designing and managing professional education programs as sources to create PEPML models by leveraging the same technical infrastructure used for extracting models from whiteboards. As

a result, our approach facilitates collaboration that is not limited to a single online whiteboard but can span multiple boards and tools.

To assist in work organization, PEPML allows organizing tasks that have to be performed by education providers in relation to programme elements. Open tasks can directly be visualized in PEPML models, allowing education providers to get an overview of pending tasks based on a programme's design. PEPML models are intended as an addition to existing work organization tools. The information within these models can be used to structure work items within such tools consistently. We showed this based on an example integration with the project management tool OpenProject. Additionally, PEPML allows defining constraints to ensure that tasks can only be closed if the intended outcome is reflected in the PEPML model.

Often, relevant situational factors are only known by programme designers or managers. In this thesis, we showed that PEPML models can also serve as situational context. Situational information can directly be modelled or obtained by integrating information from relevant tools. Only missing information is then obtained by inquiring the programme designers or managers. Since PEPML models can describe professional education programmes across all life cycle phases, they provide a good basis for storing situational information.

Research Question 2: How can online whiteboards be used as primary tools for designing and managing professional education programmes collaboratively?

Current online whiteboards only support informal modelling. They may support shapes specific to major modelling languages like UML or BPMN but do not offer any features to extract actual models of these languages or support any form of syntax validation. With the visual modelling tool support for PEPML based on the online whiteboard Miro, we showed that online whiteboards can be extended to provide such features. Instead of hardcoding the transformations of board content to PEPML models, we utilized TGG-based model transformations. We widened the range of TGG application scenarios by introducing preprocessors that extend the board content so that TGGs can properly transform it into a PEPML model. This combination of preprocessors and TGG-based model transformations is powerful enough to cover all application scenarios described in this thesis. Since the TGGs used for model transformation are maintained outside of the code of the Miro extension, our technical infrastructure can be reused to support other modelling languages and can be adapted to other online whiteboards as well.

Research Question 3: How to integrate information about a professional education programme that is spread across multiple tools?

Integrating information from multiple tools can be done in the same fashion as we integrate information from multiple Miro boards. We transform information from a tool into a PEPML model and merge it with models obtained by other tools to get an integrated version of the model. The technical infrastructure from our Miro-based visual modelling tool support can be reused for this purpose, and the creation of the necessary TGGs can be simplified by using GenTGG. For this to work, a tool must offer its data through an API or use file-based storage. An API is preferable since it allows more fine-grained access, partial updates and, ideally, notifications about data changes. Once transformations to PEPML are defined for an API or file format, they can be reused by others. In this thesis, we provide example transformations for task information provided by the project management tool OpenProject and for common types of spreadsheets. Integrating information from multiple tools into a single PEPML model helps to get an overview of where information is stored. Furthermore, the model provides more detailed situational information.

Research Question 4: How can situation-specific knowledge about designing and managing professional education programmes be stored and recalled?

We defined an extension of PEPML, allowing the definition of knowledge items and situations in which they are applicable. Situations are described in terms of situational factors that should be present in a PEPML model. We transform situation descriptions into Cypher queries to automatically determine if certain situations occur. Due to the extensibility of PEPML, additional properties, relations and subclasses can be added if the current elements do not suffice to describe the situation. Knowledge items relevant to a given situation are recommended to users. This knowledge can be described textually or in the form of partial PEPML models that can be incorporated into the model describing the professional education programme. This approach can be used to incrementally develop design and management methods due to PEPML's ability to describe method parts, e.g. tasks or roles. Since situational information within a model is likely to be incomplete, we also allow to define questions to add missing information, which enables semi-automatic model completion.

10.3 Future Work

Future work is divided into two subsections: The first subsection covers aspects regarding the Model-Driven Engineering techniques used and developed in this thesis. The second subsection focuses on further ways to improve the situation-specific recommendations for education providers.

10.3.1 Model-Driven Engineering Aspects

Supporting other Online Whiteboards

We presented a Miro-based approach for visual modelling. However, our analysis showed that other online whiteboards like Mural or LucidSpark would have been suitable alternatives. The general architecture of our approach and a great extent of its implementation can be reused to develop visual modelling tool support based on other whiteboards.

Better Explanation of Errors

Syntactical problems are reported to users when detected during extraction, preprocessing, transformation or merging. Problematic elements can then be highlighted on the Miro boards. Unclear containments of items or violations of multiplicities can clearly be articulated as problems. However, when elements are not transformed, we cannot inform users why this is the case. The reason could be an incomplete TGG or a wrong application order. Automatically analysing the TGG for possible shortcomings or exploring other rule orders could provide users with more insights into the problem.

Extension of GenTGG

GenTGG is expressive enough to generate nearly all rules relevant to this thesis. We only needed to manually specify NACs for one rule of each of PEPML's viewpoints to ensure proper transformation. GenTGG's syntax could be extended to cover type restrictions and define patterns that shall not exist for the mapping to apply. With such information, we could even generate the required NACs.

Supporting other TGG Languages

eMSL is a good fit for this thesis to describe TGGs and metamodels since it has been developed with model storage and transformation in a Neo4j graph database in mind. Other TGG engines like eMoflon::IBEX [WARV19] have their own language to describe TGGs. Further, GenTGG generators can be developed to generate TGGs described in another TGG language.

Generate Rule Application Order

When generating TGGs based on GenTGG mappings, we exploit metamodel information to enrich the generated rules. If a rule is created, which is a subset of another rule, users must define a rule application order, ensuring that the more specific rule is applied first. Such a rule order could automatically be determined to a certain degree by analysing the generated TGG.

TGG engines like eMoflon::IBEX [WARV19] and eMoflon::Neo [WA21] are based on Eclipse. With eNeo.js, we provide an alternative that works without Eclipse, is more lightweight in its use and is better suited to work in web-based scenarios. Currently, eNeo.js can only operationalize TGGs for FWD and BWD. The existing engines are more powerful and allow further types of operationalization, e.g. to handle concurrent model synchronization. In the future, eNeo.js could be extended to also cover these advanced operationalizations.

Extending eNeo.js

eMoflon::IBEX and eMoflon::Neo require that a rule application order is defined via source code. The model transformation control centre in our Miro app allows defining model transformation options, like the rule application order, via a UI. Together with the runtime-based operationalization, we can develop TGGs and customize their execution at runtime. However, the model transformation options are stored within the app. TGG languages like eMSL should be extended to also cover information like the rule application order or the number of allowed applications per rule.

Store Transformation Options in TGGs

In this thesis, we focus on detecting problems and pinpointing users to the problem's source. The resolution is made manually. Weidmann's fault-tolerant model synchronization approach [Wei22] tries to find a consistent solution automatically. We decided against using this approach because it might lead to the deletion of relevant elements. In the future, such automated resolutions could be offered as potential solutions to users. Additionally, being able to analyse the model transformation similar to the VICToRy debugger [WYA+22] could further assist users in problem resolution.

Further Assistance in Problem Resolution

10.3.2 Situation-specific Recommendations

We showed how PEPML can be used to provide situational context for providing situation-specific recommendations. Furthermore, PEPML is designed in such a way that it can be used in collaborative online whiteboards. The language is extensible and will evolve, e.g. to allow describing additional situational factors. Over time, such extensions can be incorporated into the main language if they are considered to be of general relevance.

Incorporating Extensions into PEPML

Several possible directions exist in which PEPML could be extended. For instance, an additional viewpoint focusing on cost or information access management could be added. Moreover, additional client roles like legal, procurement or the work council could be added. While the focus of this thesis has been on professional education programmes, extensions for other types of programmes could be developed as well.

Extending PEPML

Filling the Knowledge Base & User Studies

We provide examples of items that can be added to the knowledge base of our approach throughout this thesis. While these examples are suitable for understanding the concepts around our approach, filling the knowledge base was out of scope. Furthermore, we describe the technological basis for situation-specific recommendations. In the future, a knowledge base could be filled, and user studies could be conducted to evaluate practical benefits.

Further Integrations & Marketplace

We present integrations with OpenProject and common types of spreadsheets in this thesis. Additional integrations, especially with LMSs, will be needed in the future. Ideally, these integrations are paired with a marketplace to exchange such integrations with interested parties.

Learner Analytics

In this thesis, we focussed on support for designing and managing professional education programmes. PEPML provides basic support for defining evaluation strategies, which could be extended to incorporate advanced learner analytics in a future thesis. With xAPI [Rus23a], a standard exists to track learning across multiple platforms and settings. Such information could be correlated with the contents of PEPML models to provide more in-depth insights into learning paths and reached learning outcomes.

AI-based Co-Designer

In a future thesis, our approach could be paired with generative AI solutions like ChatGPT to realize an AI-based co-designer for professional education programmes. ChatGPT can already be used for creating lesson plans or providing rough structures for education programmes. However, the output is based on probability and is not checked for syntactical and semantical correctness. Also, the amount of provided context information greatly influences the quality of the output. Such context information could be provided based on PEPML models. Furthermore, we could demand a (partial) PEPML model as output, on which we can perform syntactical and semantical checks.

References

[ABC+21] Lena Ahner, Yasemin Bayrak, Imogen Cleaver, Maximilian L. Ge, Jens Neuhüttler, and Florian Urmetzer. Impacts of Professional Education Measures on the Digital Transformation in Organisations. In *Advances in the Human Side of Service Engineering*, volume 266 of *Lecture Notes in Networks and Systems*, pages 173–180. Springer, 2021.

[ABJ+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Model Driven Engineering Languages and Systems (MODELS'10)*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2010.

[ADJ+17] Anthony Anjorin, Zinovy Diskin, Frédéric Jouault, Hsiang-Shang Ko, Erhan Leblebici, and Bernhard Westfechtel. Benchmarx reloaded: A practical benchmark framework for bidirectional transformations. In *International Workshop on Bidirectional Transformations (Bx'17)*, volume 1827 of *CEUR Workshop Proceedings*, pages 15–30. CEUR-WS.org, 2017.

[AKA+01] Lorin W. Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Raths, and Merlin C. Wittrock. *A Revision of Bloom's Taxonomy of Educational Objectives*. Addison-Wesley, 2001.

[AMAC21] Amana Ahmmad, Imaan Moughal, Cristina Adriana Alexandru, and Aurora Constantin. Systematic Review of Online Collaborative Whiteboard Platforms for Higher Education. In *University of Edinburgh Learning and Teaching Conference*, 2021.

[Anj18] Anthony Anjorin. An Introduction to Triple Graph Grammars as an Implementation of the Delta-Lens Framework. In *Bidirectional Transformations*, pages 29–72. Springer, 2018.

[APH14] Tapio Auvinen, Juha Paavola, and Juha Hartikainen. STOPS: A Graph-based Study Planning and Curriculum Development Tool. In *Koli Calling International Conference on Computing Education Research (Koli Calling'14)*, pages 25–34. ACM, 2014.

[Ari16] Svetlana Arifulina. *Solving Heterogeneity for a Successful Service Market*. PhD thesis, Paderborn University, 2016.

[AS12] Michael W. Allen and Richard Sites. *Leaving ADDIE for SAM*. American Society for Training and Development, 2012.

[BAHT21] Emilio Brandao, Marco Adelfio, Shea Hagy, and Liane Thuvander. Collaborative Pedagogy for Co-creation and Community Outreach: An Experience from Architectural Education in Social Inclusion Using the Miro Tool. In *Advances in Human Dynamics for the Development of Contemporary Societies*, volume 277 of *Lecture Notes in Networks and Systems*, pages 118–126. Springer, 2021.

[BBCR+08] Luca Botturi, Daniel Burgos, Manuel Caeiro-Rodríguez, Michael Derntl, Rob Koper, Patrick Parrish, Tim Sodhi, and Colin Tattersal. Comparing Visual Instructional Design Languages: A Case Study. In *Handbook of Visual Languages for Instructional Design: Theories and Practices*, pages 315–344. IGI Global, 2008.

[BBE21] Saçak Begüm, Aras Bozkurt, and Wagner Ellen. Instructional Design vs Learning Design: Trends and Patterns in Scholarly Landscape. In *Association for Educational Communications and Technology (AECT'21)*, 2021.

[BBH+08] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. Motivation in Software Engineering: A Systematic Literature Review. *Information and Software Technology*, 50(9):860–878, 2008.

[BBvB+01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development, 2001. URL: http://www.agilemanifesto.org/.

[BCW17] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2017.

[BGE14] Dennis Bokermann, Christian Gerth, and Gregor Engels. Use Your Best Device! Enabling Device Changes at Runtime. In *Business Process Management (BPM'14)*, volume 8659 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 2014.

[Big96] John Biggs. Enhancing Teaching through Constructive Alignment. *Higher Education*, 32(3):347–364, 1996.

[Bit22] Bitkom e. V. IT-Fachkräftelücke wird größer: 96.000 offene Jobs, 2022. (Accessed 2023-11-08). URL: https://www.bitkom.org/Presse/Presseinformation/IT-Fachkraefteluecke-wird-groesser.

[BK14] Robert Maribe Branch and Theodore J. Kopcha. Instructional Design Models. In *Handbook of Research on Educational Communications and Technology*, pages 77–87. Springer, 2014.

[BN19] Mateus Brito and Isabel Nunes. ATID Web Authoring Tool for Instructional Design. In *Workshops do Congresso Brasileiro de Informática na Educação (WCBIE'19)*. Brazilian Computer Society, 2019.

[Bot06] Luca Botturi. E^{2}ML: A Visual Language for the Design of Instruction. *Educational Technology Research and Development*, 54(3):265–293, 2006.

[Bot08] Luca Botturi. E^{2}ML. In *Handbook of Visual Languages for Instructional Design*, pages 111–131. IGI Global, 2008.

[Bra09] Robert Maribe Branch. *Instructional Design: The ADDIE Approach*. Springer, 2009.

[BT23] David Bernal Tejedor. Werkzeugunterstützung für grafische Modellierungssprachen basierend auf kollaborativen Online-Whiteboards. Bachelor's thesis, Paderborn University, 2023.

[BvdWBM08] Willem Bekkers, Inge van de Weerd, Sjaak Brinkkemper, and Alain Mahieu. The Influence of Situational Factors in Software Product Management: An Empirical Study. In *International Workshop on Software Product Management (IWSPM'08)*, pages 41–48. IEEE, 2008.

[CGH08] Qi Chen, John Grundy, and John Hosking. SUMLOW: Early Design-Stage Sketching of UML Diagrams on an E-whiteboard. *Software: Practice and Experience*, 38(9):961–994, 2008.

[CH06] Krzysztof Czarnecki and Simon Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[CLA14] Manuel Caeiro, Martín Llamas, and Luis Anido. PoEML: Modeling Learning Units through Perspectives. *Computer Standards & Interfaces*, 36(2):380–396, 2014.

[Con09] Gráinne Conole. The Role of Mediating Artefacts in Learning Design. In *Handbook of Research on Learning Design and Learning Objects: Issues, Applications, and Technologies*, pages 188–208. IGI Global, 2009.

[CR19] Manuel Caeiro-Rodriguez. Making Teaching and Learning Visible. In *Technological Ecosystems for Enhancing Multiculturality (TEEM'19)*, pages 265–274. ACM, 2019.

[CZ14] Nadire Cavus and Teyang Zabadi. A Comparison of Open Source Learning Management Systems. *Procedia - Social and Behavioral Sciences*, 143:521–526, 2014.

[Dar17] Dark Horse Innovation. *Digital Innovation Playbook*. Murmann, 2017.

[DCC05] Walter Dick, Lou Carey, and James O. Carey. *The Systematic Design of Instruction*. Pearson Education, 2005.

[DGC17] Zinovy Diskin, Abel Gómez, and Jordi Cabot. Traceability Mappings as a Fundamental Instrument in Model Transformations. In *Fundamental Approaches to Software Engineering*, volume 10202 of *Lecture Notes in Computer Science*, pages 247–263. Springer, 2017.

[DMP08] Michael Derntl and Renate Motschnig-Pitrik. coUML: A Visual Language for Modeling Cooperative Environments. In *Handbook of Visual Languages for Instructional Design: Theories and Practices*, pages 154–182. IGI Global, 2008.

[Dor81] George T Doran. There's a S.M.A.R.T. Way to Write Management's Goals and Objectives. *Management Review*, 70(11):35–36, 1981.

[Dou08] Ian Douglas. Performance Case Modeling. In *Handbook of Visual Languages for Instructional Design: Theories and Practices*, pages 208–223. IGI Global, 2008.

[EHHS00] Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML. In *The Unified Modeling Language (UML'00)*, volume 1939 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2000.

[ES10] Gregor Engels and Stefan Sauer. A Meta-Method for Defining Software Engineering Methods. In *Graph Transformations and Model-Driven Engineering*, volume 5765 of *Lecture Notes in Computer Science*, pages 411–440. Springer, 2010.

[FB16] Masud Fazal-Baqaie. *Project-specific Software Engineering Methods: Composition, Enactment, and Quality Assurance*. PhD thesis, Paderborn University, 2016.

[FFVM04] Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno. An Introduction to UML Profiles. *UML and Model Engineering*, V(2), 2004.

[FKSH09] Jan Friedrich, Marco Kuhrmann, Marc Sihling, and Ulrike Hammerschall. *Das V-Modell XT*. Springer, 2009.

[FLM+09] Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. In *Enterprise, Business-Process and Information Systems Modeling*, Lecture Notes in Business Information Processing, pages 353–366. Springer, 2009.

[FU02] Sally Fincher and Ian Utting. Pedagogical Patterns: Their Place in the Genre. In *Innovation and Technology in Computer Science Education (ITiCSE'02)*. ACM, 2002.

[Gei15] Silke Geisen. *Selbst-adaptive Software-Engineering-Methoden*. PhD thesis, Paderborn University, 2015.

[GFV13] Helen Gibson, Joe Faith, and Paul Vickers. A Survey of Two-Dimensional Graph Layout Techniques for Information Visualisation. *Information Visualization*, 12(3-4):324–357, 2013.

[GGM15] Timothy Goddard, David Griffiths, and Wang Mi. Why has IMS Learning Design not Led to the Advances which were Hoped for? In *The Art & Science of Learning Design*, Technology Enhanced Learning, pages 121–136. SensePublishers, 2015.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Got22] Sebastian Gottschalk. *Situation-specific Development of Business Models within Software Ecosystems*. PhD thesis, Paderborn University, 2022.

[Gri16] Marvin Grieger. *Model-Driven Software Modernization: Concept-based Engineering of Situation-specific Methods*. PhD thesis, Paderborn University, 2016.

[GW06] Holger Giese and Robert Wagner. Incremental Model Synchronization with Triple Graph Grammars. In *Model Driven Engineering Languages and Systems (MODELS'06)*, volume 4199 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2006.

[GW09] Holger Giese and Robert Wagner. From Model Transformation to Incremental Bidirectional Model Synchronization. *Software & Systems Modeling (SoSyM)*, 8(1):21–43, 2009.

[GWGK05] Robert M. Gagné, Walter W. Wager, Katharine C. Golas, and John M. Keller. *Principles of Instructional Design*. Wadsworth, 2005.

[Har05] Rosemary Harrison. *Learning and Development*. CIPD Publishing, 2005.

[HBO94] Frank Harmsen, Sjaak Brinkkemper, and Han Oei. Situational Method Engineering for Information System Project Approaches. In *Methods and Associated Tools for the Information Systems Life Cycle*, pages 169–194, 1994.

[HBVW08] Tracy Hall, Sarah Beecham, June Verner, and David Wilson. The Impact of Staff Turnover on Software Projects: The Importance of Understanding What Makes Software Practitioners

Tick. In *Computer Personnel Doctoral Consortium and Research (SIGMIS CPR'08)*, pages 30–39. ACM, 2008.

[HLG+13] Stephan Hildebrandt, Leen Lambers, Holger Giese, Jan Rieke, Joel Greenyer, Wilhelm Schäfer, Marius Lauder, Anthony Anjorin, and Andy Schürr. A Survey of Triple Graph Grammar Tools. *Electronic Communications of the EASST*, 57, 2013.

[HMRS02] Robert Heinich, Michael Molenda, James D. Russel, and Sharon E. Smaldino. *Instructional Media and Technology for Learning*. Pearson Education, 2002.

[HNB+14] Frank Hermann, Nico Nachtigall, Benjamin Braatz, Thomas Engel, and Susann Gottmann. Solving the FIXML2Code-case study with HenshinTGG. In *Transformation Tool Contest (TTC'14)*, volume 1305 of *CEUR Workshop Proceedings*, pages 32–46. CEUR-WS.org, 2014.

[HR10] Brian Henderson-Sellers and Jolita Ralyté. Situational Method Engineering: State-of-the-Art Review. *Journal of Universal Computer Science (JUCS)*, 16(3):424–478, 2010.

[HSRÅR14] Brian Henderson-Sellers, Jolita Ralyté, Pär J. Ågerfalk, and Matti Rossi. *Situational Method Engineering*. Springer, 2014.

[HT08] H. Han and T. Tokuda. A Method for Integration of Web Applications Based on Information Extraction. In *International Conference on Web Engineering (ICWE'08)*, pages 189–195. IEEE, 2008.

[IBA02] Ivan Ivanov, Jean Bézivin, and Mehmet Aksit. Technological Spaces: An Initial Appraisal. In *Distributed Objects and Applications (DOA'02)*, pages 1–6, 2002.

[IMS03] IMS Global Learning Consortium. Learning Design Specification - Version 1, 2003. URL: http://www.imsglobal.org/learningdesign/.

[JMG+08] Francisco Jurado, Ana I. Molina, William J. Giraldo, Miguel A. Redondo, and Manuel Ortega. Using CIAN for Specifying Collaborative Scripts in Learning Design. In *Cooperative Design, Visualization, and Engineering (CDVE'08)*, pages 204–211. Springer, 2008.

[Jov20] Ivan Jovanovikj. *Validation of Software Migration: Model-Driven Co-Migration of Test Cases*. PhD thesis, Paderborn University, 2020.

[KC13] Bilal Karasneh and Michel R.V. Chaudron. Img2UML: A System for Extracting UML Models from Images. In *Software Engineering and Advanced Applications (SEAA'13)*, pages 134–137. IEEE, 2013.

[KDMM09] Eva Kyndt, Filip Dochy, Maya Michielsen, and Bastiaan Moeyaert. Employee Retention: Organisational and Personal Perspectives. *Vocations and Learning*, 2(3):195–215, 2009.

[Kee22] Michael Keeling. The Psychology of Architecture Decision Records. *IEEE Software*, 39(6):114–117, 2022.

[Kir98] Donald L. Kirkpatrick. Evaluating Training Programs: The Four Levels. In *Evaluating Corporate Training: Models and Issues*, pages 95–112. Springer, 1998.

[KKP+09] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design Guidelines for Domain-specific Languages. In *Workshop on Domain-Specific Modeling (DSM'09)*, 2009.

[KLKS10] Felix Klar, Marius Lauder, Alexander Königs, and Andy Schürr. Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. In *Graph Transformations and Model-Driven Engineering*, pages 141–174. Springer, 2010.

[KLR+12] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Model Transformation By-Example: A Survey of the First Wave. In *Conceptual Modelling and Its Theoretical Foundations*, pages 197–215. Springer, 2012.

[Kre17] Ralf T. Kreutzer. Treiber und Hintergründe der digitalen Transformation. In *Digitale Transformation von Geschäftsmodellen: Grundlagen, Instrumente und Best Practices*, pages 33–58. Springer, 2017.

[LAS17] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Intermodel Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques. In *Fundamental Approaches to Software Engineering*, volume 10202 of *Lecture Notes in Computer Science*, pages 191–207. Springer, 2017.

[LGB07] Juan de Lara, Esther Guerra, and Paolo Bottoni. Triple Patterns: Compact Specifications for the Generation of Operational Triple Graph Grammar Rules. *Electronic Communications of the EASST*, 6, 2007.

[Li22] Ling Li. Reskilling and Upskilling the Future-ready Workforce for Industry 4.0 and Beyond. *Information Systems Frontiers*, 2022.

[LW94] Barbara H. Liskov and Jeannette M. Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, 1994.

[LZXL21] Qingchuan Li, Jiaxin Zhang, Xin Xie, and Yan Luximon. How Shared Online Whiteboard Supports Online Collaborative Design Activities: A Social Interaction Perspective. In *Advances*

in Creativity, Innovation, Entrepreneurship and Communication of Design, volume 276 of *Lecture Notes in Networks and Systems*, pages 285–293. Springer, 2021.

[Mac06] Keith Mackrell. In my Opinion. *Management Today*, page 12, 2006.

[May17] Richard E. Mayer. Using Multimedia for e-Learning. *Journal of Computer Assisted Learning*, 33(5):403–423, 2017.

[ME22] Claude Müller and Jennifer Erlemann. Educational Design for Digital Learning with myScripting. In *Shaping the Digital Transformation of the Education Ecosystem in Europe (EDEN Conference'22)*, pages 128–134, 2022.

[MF20] Dennis Mancl and Steven D. Fraser. COVID-19's Influence on the Future of Agile. In *Agile Processes in Software Engineering and Extreme Programming - Workshops*, volume 396 of *Lecture Notes in Business Information Processing*, pages 309–316. Springer, 2020.

[MJdlC+09] Ana Isabel Molina, Francisco Jurado, Ignacio de la Cruz, Miguel Angel Redondo, and Manuel Ortega. Tools to Support the Design, Execution and Visualization of Instructional Designs. In *Cooperative Design, Visualization, and Engineering*, volume 5738 of *Lecture Notes in Computer Science*, pages 232–235. Springer, 2009.

[Moo09] Daniel Moody. The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.

[MVMMV15] Stéphanie Mailles-Viard Metz, Philippe Marin, and Émilie Vayre. The Shared Online Whiteboard: An Assistance Tool to Synchronous Collaborative Design. *European Review of Applied Psychology*, 65(5):253–265, 2015.

[NS14] Isabel Dillmann Nunes and Ulrich Schiel. Using High Level Activities Net for Learning Analytics of Instructional Design. In *International Conference on Advanced Learning Technologies (ICALT'14)*, pages 383–385. IEEE, 2014.

[Obj08] Object Management Group (OMG). Software & Systems Process Engineering Metamodel - Version 2.0, 2008. URL: https://www.omg.org/spec/SPEM/2.0.

[Obj11] Object Management Group (OMG). Business Process Model and Notation (BPMN) - Version 2.0, 2011. URL: http://www.omg.org/spec/BPMN/2.0.

[Obj14] Object Management Group (OMG). Object Constraint Language - Version 2.4, 2014. URL: https://www.omg.org/spec/OCL/2.4.

[Obj15] Object Management Group (OMG). Meta Object Facility Specification - Version 2.5, 2015. URL: https://www.omg.org/spec/MOF/2.5/.

[Obj17] Object Management Group (OMG). Unified Modeling Language - Version 2.5.1, 2017. URL: https://www.omg.org/spec/UML/2.5.1.

[Obj19] Object Management Group (OMG). System Modeling Language Specification - Version 1.6, 2019. URL: https://www.omg.org/spec/SysML/1.6.

[OP10] Alexander Osterwalder and Yves Pigneur. *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. Wiley, 2010.

[OPBS14] Alexander Osterwalder, Yves Pigneur, Greg Bernarda, and Alan Smith. *Value Proposition Design: How to Create Products and Services Customers Want*. Wiley, 2014.

[Paq04] Gilbert Paquette. *Instructional Engineering in Networked Environments*. Pfeiffer, 2004.

[PBNM05] Valéry Psyché, Jacqueline Bourdeau, Roger Nkambou, and Riichiro Mizoguchi. Making Learning Design Standards Work with an Ontology of Educational Theories. In *Artificial Intelligence in Education (AIED'05)*, pages 539–546, 2005.

[PdlTL+05] Gilbert Paquette, Ileana de la Teja, Michel Léonard, Karin Lundgren-Cayrol, and Olga Marino. An Instructional Engineering Method and Tool for the Design of Units of Learning. In *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*, pages 161–184. Springer, 2005.

[Ped12] Pedagogical Patterns Editorial Board. *Pedagogical Patterns: Advice for Educators*. Joseph Bergin Software Tools, 2012.

[Pet13] Marian Petre. UML in Practice. In *International Conference on Software Engineering (ICSE'13)*, pages 722–731. IEEE, 2013.

[PHW+19] Till Post, Aaron Heuermann, Stefan Wiesner, Detlef Olschewski, Wolfgang Maaß, Rüdiger Klatt, Philipp Jussen, Sherif Ragab, Roman Senderek, Benedikt Höckmayr, Thomas Schulz, Kyrill Meyer, Ewald Heinen, Christian Hocken, Simon Fischer, Christoph Lattemann, Beke Redlich, Katrin Schlimm, Christoph Ziegler, Michael Falkus, Christopher Rechtien, Markus Schröder, Bernhard Kube, Jens Pöppelbuß, Manuel Wiesche, Christian Bartelheimer, Daniel Beverungen, Hedda Lüttenberg, Verena Wolf, Franziska Bongers, Corinna Winkler, Jan Hendrik Schumann, Mahei Manhai Li, Jonas Brinker, Simon Hagen, Friedemann Kammler, Giuseppe Strina, Philipp Ernst, and Michael Falkus. *DIN SPEC 33453:*

Entwicklung digitaler Dienstleistungssysteme (Development of Digital Service Systems). 2019.

[PL06] Gilbert Paquette and Michel Léonard. The Educational Modeling of a Collaborative Game using MOT+LD. In *International Conference on Advanced Learning Technologies (ICALT'06)*, pages 1156–1157. IEEE, 2006.

[PLLC11] Gilbert Paquette, Michel Léonard, and Karin Lundgren-Cayrol. The MOT+Visual Language for Knowledge-Based Instructional Design. In *Instructional Design: Concepts, Methodologies, Tools and Applications*, pages 697–717. IGI Global, 2011.

[Plo04] Gordon D. Plotkin. The Origins of Structural Operational Semantics. *The Journal of Logic and Algebraic Programming*, 60-61:3–15, 2004.

[POB00] Richard F. Paige, Jonathan S. Ostroff, and Phillip J. Brooke. Principles for Modeling Language Design. *Information and Software Technology*, 42(10):665–675, 2000.

[PP21] Bhavik K. Pathak and Shailendra C. Palvia. Taxonomy of Higher Education Delivery Modes: A Conceptual Framework. *Journal of Information Technology Case and Application Research (JITCAR)*, 23(1):36–45, 2021.

[RAR13] Leonard Richardson, Mike Amundsen, and Sam Ruby. *RESTful Web APIs*. O'Reilly, 2013.

[RFK22] Nur W. Rahayu, Ridi Ferdiana, and Sri S. Kusumawardani. A Systematic Review of Ontology Use in E-Learning Recommender System. *Computers and Education: Artificial Intelligence*, 3:100047, 2022.

[Rus23a] Rustici Software LLC. Experience API (xAPI) Specification, 2023. (Accessed 2023-11-08). URL: https://xapi.com/specification/.

[Rus23b] Rustici Software LLC. SCORM, 2023. (Accessed 2023-11-08). URL: https://scorm.com/.

[Sau11] Stefan Sauer. *Systematic Development of Model-based Software Engineering Methods*. PhD thesis, Paderborn University, 2011.

[SBPM08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley, 2 edition, 2008.

[Sca21] Scaled Agile, Inc. Achieving Business Agility with SAFe® 5, 2021. URL: https://www.scaledagile.com/resources/safe-whitepaper/.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *Graph-Theoretic Concepts in Computer Science*, pages 151–163. Springer, 1994.

[Sch19] Maximilian Schmidt. EMSL: A Uniform and Modular Language for Model Management. Bachelor's thesis, Paderborn University, 2019.

[SCTR22] Darja Smite, Emily Laue Christensen, Paolo Tell, and Daniel Russo. The Future Workplace: Characterizing the Spectrum of Hybrid Work Arrangements for Software Teams. *IEEE Software*, 40(2):1–9, 2022.

[SH17] Stephan Seifermann and Jorg Henß. Comparison of QVT-O and Henshin-TGG for Synchronization of Concrete Syntax Models. In *International Workshop on Bidirectional Transformations (Bx'17)*, 2017.

[SKLR21] Patrick Stünkel, Harald König, Yngve Lamo, and Adrian Rutle. Comprehensive Systems: A formal foundation for Multi-Model Consistency Management. *Formal Aspects of Computing*, 33(6):1067–1114, 2021.

[SM20] Saskia Stillhart and Timothee Moos. Whiteboard-to-Model Compiler (miro2cml). Study research project, OST Ostschweizer Fachhochschule, 2020.

[Spi15] Michael Spijkerman. *Situationsgerechte Methodenweiterentwicklung auf Basis von MetaMe am Beispiel der Server-System-Entwicklung*. PhD thesis, Paderborn University, 2015.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973.

[Ste17] Perdita Stevens. Bidirectional Transformations in the Large. In *Model Driven Engineering Languages and Systems (MODELS'17)*, pages 1–11, 2017.

[SvBRL20] Patrick Stünkel, Ole von Bargen, Adrian Rutle, and Yngve Lamo. GraphQL Federation: A Model-Based Approach. *The Journal of Object Technology*, 19(2):18:1–21, 2020.

[SvdALS23] Bernhard Schäfer, Han van der Aa, Henrik Leopold, and Heiner Stuckenschmidt. Sketch2Process: End-to-End BPMN Sketch Recognition Based on Neural Networks. *IEEE Transactions on Software Engineering*, 49(4):2621–2641, 2023.

[SZS23] Mohammadreza Sharbaf, Bahman Zamani, and Gerson Sunyé. Conflict Management Techniques for Model Merging: A Systematic Mapping Review. *Software & Systems Modeling (SoSyM)*, 22(3):1031–1079, 2023.

[TdMFS+17] Douglas Afonso Tenorio de Menezes, Diogo Lima Florencio, Renato Ely Domingues Silva, Isabel Dillmann Nunes, Ulrich Schiel, and Marcus Salerno de Aquino. DaVID — A Model of Data Visualization for the Instructional Design. In *International Conference on Advanced Learning Technologies (ICALT'17)*, pages 281–285. IEEE, 2017.

[TGGLH21] Andrew A. Tawfik, Jessica Gatewood, Jaclyn J. Gish-Lieberman, and Andrew J. Hampton. Toward a Definition of Learning Experience Design. *Technology, Knowledge and Learning*, 27(1):309–334, 2021.

[Tsi18] George J. Tsinarakis. Modeling Task Dependencies in Project Management Using Petri Nets with Arc Extensions. In *Mediterranean Conference on Control and Automation (MED'18)*. IEEE, 2018.

[TZ21] Chen Hsi Tsai and Jelena Zdravkovic. A Foundation for Design, Analysis, and Management of Digital Business Ecosystem through Situational Method Engineering. In *The Practice of Enterprise Modeling*, volume 432 of *Lecture Notes in Business Information Processing*, pages 134–149. Springer, 2021.

[UNE11] UNESCO Institute for Statistics. International Standard Classification of Education: ISCED 2011, 2011.

[US 17] US Army. Army Learning Policy and Systems. Technical report, 2017.

[VBD+13] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser, and Guido Wachsmuth. *DSL Engineering: Designing, Implementing and Using Domain-specific Languages*. dslbook.org, 2013.

[VCSP12] Christian Vidal-Castro, Miguel-Ángel Sicilia, and Manuel Prieto. Representing Instructional Design Methods using Ontologies and Rules. *Knowledge-Based Systems*, 33:180–194, 2012.

[vdLH19] Dirk van der Linden and Irit Hadar. A Systematic Literature Review of Applications of the Physics of Notations. *IEEE Transactions on Software Engineering*, 45(8):736–759, 2019.

[VJC17] Boban Vesin, Rodi Jolak, and Michel R.V. Chaudron. OctoUML: An Environment for Exploratory and Collaborative Software Design. In *International Conference on Software Engineering Companion (ICSE-C'19)*, pages 7–10, 2017.

[VW74] Adriaan Van Wijngaarden. The Generative Power of Two-Level Grammars. In *Automata, Languages and Programming*, volume 14 of *Lecture Notes in Computer Science*, pages 9–16. Springer, 1974.

[WA21] Nils Weidmann and Anthony Anjorin. eMoflon::Neo - Consistency and Model Management with Graph Databases. In *International Workshop on Bidirectional Transformations (Bx'21)*, volume 2999 of *CEUR Workshop Proceedings*, pages 54–64. CEUR-WS.org, 2021.

[WARV19] Nils Weidmann, Anthony Anjorin, Patrick Robrecht, and Gergely Varró. Incremental (unidirectional) model transformation with eMoflon::IBeX. In *Graph Transformation*, pages 131–140. Springer, 2019.

[WB19] Dennis Wagner and Marc Bleß. *Agile Spiele - kurz & gut: Für Agile Coaches und Scrum Master*. O'Reilly, 2019.

[WE22a] Dennis Wolters and Gregor Engels. Model-Driven Design and Management of Professional Education Programmes. In *International Conference on Software Business (ICSOB'22) Companion Proceedings*, volume 3316 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[WE22b] Dennis Wolters and Gregor Engels. Towards Situational Process Management for Professional Education Programmes. In *Product-Focused Software Process Improvement (PROFES'22)*, volume 13709 of *Lecture Notes in Computer Science*, pages 235–242. Springer, 2022.

[WE23] Dennis Wolters and Gregor Engels. Model-Driven Collaborative Design of Professional Education Programmes With Extended Online Whiteboards. In *Model-Based Software and Systems Engineering (MODELSWARD'23)*, pages 133–142. SciTePress, 2023.

[Wei22] Nils Weidmann. *Fault-Tolerant Consistency Management in Model-Driven Engineering*. PhD thesis, Paderborn University, 2022.

[WGE16] Dennis Wolters, Christian Gerth, and Gregor Engels. Modeling Cross-Device Systems with Use Case Diagrams. In *Conference on Advanced Information Systems Engineering (CAiSE'16) Forum*, volume 1612 of *CEUR Workshop Proceedings*, pages 89–96. CEUR-WS.org, 2016.

[WGE17] Dennis Wolters, Christian Gerth, and Gregor Engels. Visual Requirements Modeling for Cross-Device Systems. *Computer Science and Information Systems (ComSIS)*, 14(2):517–536, 2017.

[WHKE17] Dennis Wolters, Stefan Heindorf, Jonas Kirchhoff, and Gregor Engels. Linking Services to Websites by Leveraging Semantic Data. In *International Conference on Web Services (ICWS'17)*, pages 668–675. IEEE, 2017.

[WHKE18] Dennis Wolters, Stefan Heindorf, Jonas Kirchhoff, and Gregor Engels. Semantic Data Mediator: Linking Services to Websites. In *Service-Oriented Computing (ICSOC'17) Workshops*, volume 10797 of *Lecture Notes in Computer Science*, pages 388–392. Springer, 2018.

[WKE19] Dennis Wolters, Jonas Kirchhoff, and Gregor Engels. Specifying Web Interfaces for Command-Line Applications Based on OpenAPI. In *Service-Oriented Computing (ICSOC'19) Workshops*, volume 12019 of *Lecture Notes in Computer Science*, pages 30–41. Springer, 2019.

[WKGE16a] Dennis Wolters, Jonas Kirchhoff, Christian Gerth, and Gregor Engels. Cross-Device Integration of Android Apps. In *Service-Oriented Computing (ICSOC'16)*, volume 9936 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2016.

[WKGE16b] Dennis Wolters, Jonas Kirchhoff, Christian Gerth, and Gregor Engels. XDAI-A: Framework for Enabling Cross-Device Integration of Android Apps. In *Service-Oriented Computing (ICSOC'16) Workshops*, volume 10380 of *Lecture Notes in Computer Science*, pages 203–206. Springer, 2016.

[Wol18] Dennis Wolters. Einsatz von Classroom-Response-Systemen und Peer Instruction in der Veranstaltung Grundlagen von Datenbanken. *die hochschullehre*, 4/2018:277–298, 2018.

[WSE18] Marcus Wolf, Arlett Semm, and Christian Erfurth. Digital Transformation in Companies - Challenges and Success Factors. In *Innovations for Community Services*, Communications in Computer and Information Science, pages 178–193. Springer, 2018.

[WSG19] Dustin Wüest, Norbert Seyff, and Martin Glinz. FlexiSketch: A Lightweight Sketching and Metamodeling Approach for End-Users. *Software & Systems Modeling (SoSyM)*, 18(2):1513–1541, 2019.

[WYA+22] Nils Weidmann, Enes Yigitbas, Anthony Anjorin, Ankita Srivastava, and Jane Jose. Human-in-the-Loop Large-Scale Model Transformations with the VICToRy Debugger. *The Journal of Object Technology*, 21(3):3:1–15, 2022.

List of Figures

List of Tables

List of Listings

List of Acronyms

ADDIE	Analysis, Design, Development, Implementation, Evaluation
ADR	Architecture Decision Record
AI	Artificial Intelligence
API	Application Programming interface
BMC	Business Model Canvas
BPMN	Business Model and Notation
BWD	Backward Transformation
BX	Bidirectional Transformation
CO	Contribution
CC	Comparison Criteria
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DSL	Domain-Specific Language
EMF	Eclipse Modelling Framework
eMSL	eMoflon Specification Language
ER	Expressiveness Requirement
FWD	Forward Transformation
GPL	General Purpose Language
HATEOAS	Hypermedia as the Engine of Application State
HSV	Hue, Saturation, Value
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
ILP	Integer Linear Programming
IR	Integration Requirement
ISCED	International Standard Classification of Education
JSON	JavaScript Object Notation
LD	Learning Design
LLM	Large Language Model

LMS	Learning Management Systems
LR	Language Requirement
LRS	Learning Record Store
MDE	Model-Driven Enginnering
ML	Machine Learning
MOF	Meta Object Facility
MOOC	Massive Open Online Course
NAC	Negative Application Condition
OCL	Object Constraint Language
OMG	Object Management Group
OTS	Off The Shelf
PCM	Performance Case Modelling
PEPML	Professional Education Programme Modelling Language
PMT	Project Management Tool
REST	Representational State Transfer
RGB	Red, Green, Blue
RQ	Research Question
RR	Recommendation Requirement
RUP	Rational Unified Process
SAFe	Scaled Agile Framework
SAM	Successive Approximation Model
SDK	Software Development Kit
SME	Situational Method Engineering
SPEM	Software & Systems Process Engineering Metamodel
TGG	Triple Graph Grammar
TR	Tool Requirement
UI	User interface
UML	Unified Modelling Language
XMI	XML Metadata Interchange

Appendix A

Expressiveness Requirements Coverage

Approach	ER1	ER2	ER3	ER4	ER5	ER6	ER7
IMS-LD [IMS03]	+	+	+	o	-	-	+
Learn-CIAN [MJdlC+09]	-	+	-	-	-	-	-
MOT+(LD) [PLLC11]	o	+	o	-	-	-	+
E²ML [Bot06]	+	+	-	o	-	-	o
HLAN [NS14]	-	-	-	-	-	-	+
STOPS [APH14]	+	-	-	-	-	-	+
PoEML [CLA14]	+	+	+	-	-	-	+
MyScripting [ME22]	+	+	o	+	-	-	+
UML [Obj17]	-	o	o	+	-	-	+
PCM [Dou08]	+	+	o	+	-	-	+
coUML [DMP08]	+	+	+	+	-	-	+

Approach	ER8	ER9	ER10	ER11	ER12	ER13	ER14
IMS-LD [IMS03]	+	+	+	+	+	+	o
Learn-CIAN [MJdlC+09]	-	+	o	+	+	-	-
MOT+(LD) [PLLC11]	+	+	+	-	+	-	o
E²ML [Bot06]	o	+	+	-	+	+	o
HLAN [NS14]	+	+	-	-	+	-	-
STOPS [APH14]	+	-	+	-	-	-	-
PoEML [CLA14]	+	+	+	+	+	+	o
MyScripting [ME22]	-	+	o	+	+	+	+
UML [Obj17]	+	+	+	-	o	o	o
PCM [Dou08]	o	-	+	-	-	-	-
coUML [DMP08]	+	+	+	+	+	-	o

Appendix B

List of Compared Online Whiteboard Solutions

Name	Source	Link	Reason for Exclusion
Adobe Connect	G2	adobe.com	Video Conferencing
Aha!	G2	aha.io	Focus on product development
Allo	G2	allo.io	Dashboard/Project Management
Ayoa	G2	ayoa.com	Work Management Platform
Bitpaper	[AMAC21]	bitpaper.io	Focus on Freehand + Math Formulas
Bluescape Workspace	G2	bluescape.com	-
BrightIdea	Web	brightidea.com	Idea Management
Canva Whiteboard	Web	canva.com	-
ClickUp	G2	clickup.com	Not a Whiteboard, Project Management
Collaboard	G2	collaboard.app	-
Conceptboard	G2	conceptboard.com	-
Creately	G2	creately.com	Work Management Platform
CYBEROffice	G2	cyberhorizon.com	Video Conferencing
DEON	G2	deon.de	only partially web-based
draft.io	Web	draft.io	-
Explain Everything	G2	explaineverything.com	Focus on Freehand
ezTalks	G2	eztalks.com	Video Conferencing
Fibery	G2	fibery.io	Project Management
FigJam	G2	figma.com	-
FlowMap	Web	flowmapp.com	Focus on UX design

Name	Source	Link	Reason for Exclusion
Google Jamboard	G2	jamboard.google.com	-
GroupBoard	[AMAC21]	groupboard.com	Focus on Freehand, Outdated
Hoylu	G2	hoylu.com	Project Management
InVision Freehand	G2	freehandapp.com	-
iObeya	G2	iobeya.com	Project Management
Klaxoon Board	G2	klaxoon.com	-
Limnu	G2	limnu.com	Focus on Freehand
LiveWebinar	G2	livewebinar.com	Webinar Platform
Lucidspark	G2	lucidspark.com	-
Microsoft Whiteboard	G2	whiteboard.office.com	-
midlap	G2	midlap.com	Project Management
Milanote	Web	milanote.com	-
MiniTab Workspace	Web	minitab.com	Not web-based, Focus on statistical analysis
Miro	G2	miro.com	-
moqups	Web	moqups.com	Focus on UI/UX Design
Mural	G2	mural.co	-
NoteBookCast	Web	notebookcast.com	Focus on Freehand
Padlet	G2	padlet.com	Only restricted layouts
Productivity Lab Whiteboard	Web	productivitylab.de	-
scribblar.com	[AMAC21]	scribblar.com	Outdated, Focus on Freehand
Sketchboard	[AMAC21]	sketchboard.io	-

Name	Source	Link	Reason for Exclusion
Stormboard	G2	stormboard.com	-
Tutorialspoint Whiteboard	Web	tutorialspoint.com	Non-collaborative
Tutorsbox	[AMAC21]	tutorsbox.io	Focus on Freehand
Twiddla	[AMAC21]	twiddla.com	Focus on Freehand
VISPA	Web	vispa.io	3D Collaboration Space
WebEx Whiteboard	G2	webex.com	-
Weje	G2	weje.io	-
Whiteboard Fox	Web	whiteboardfox.com	Focus on Freehand
Whiteboard Team	Web	whiteboard.team	-
Whiteboard.chat	Web	whiteboard.chat	Focus on Freehand
Whiteboard.fi	Web	whiteboard.fi	Focus on Freehand
Ziteboard	[AMAC21]	ziteboard.com	Focus on Freehand