



DISSERTATION | DOCTORAL THESIS

Titel | Title

A Dynamic Adaptation Framework for Haptic Modelling
Methods: Creating Digital Twins from Design Thinking

verfasst von | submitted by

Dipl.-Ing. Christian Muck BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of
Doktor der Wirtschaftswissenschaften (Dr.rer.oec.)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree
programme code as it appears on the
student record sheet:

UA 796 305 175

Dissertationsgebiet lt. Studienblatt | Field of
study as it appears on the student record
sheet:

Wirtschaftsinformatik

Betreut von | Supervisor:

o. Univ.-Prof. Dr.techn. Dimitris Karagiannis

Declaration of Authorship

I declare that I have written this thesis independently and only with the support of the resources listed. The ideas and thoughts of other authors, which have been taken over in spirit or part, have been cited accordingly, and the sources have been quoted.

Date and Signature: _____

Acknowledgements

This thesis and the underlying research were supervised by Univ.-Prof. Dr.techn. Dimitris Karagiannis, head of the Research Group Knowledge Engineering at the Faculty of Computer Science of the University of Vienna. I thank Prof. Karagiannis for his guidance and support and for enabling me to do my research. His ideas and countless discussions drove the progress of my research and shaped the results. My thesis would not have been possible without his input, guidance, and the freedom and experience he provided. Apart from the scientific scope, working with Prof. Karagiannis enabled me to grow professionally.

Further, I thank my past and present colleagues from the Research Group Knowledge Engineering and the OMiLAB NPO. They supported me with countless discussions and input, shaping my thesis to its final form. From my work at the Research Group Knowledge Engineering, I thank Victoria Döller for the discussions regarding conceptual modelling and Danial Amlashi for the technical discussions. From the OMiLAB NPO, I want to highlight Dr. Wilfrid Utz and Patrik Burzynski for their valuable input and technical support throughout my thesis.

Last but not least, I want to thank my friends and family, who supported me during this time through motivating words and understanding. Here, I want to highlight Julia for being there for me and helping me personally.

Abstract

Designing innovative solutions to complex problems involves tackling "wicked problems" where no apparent solutions exist. This necessitates creative solution-finding techniques in physical co-creation workshops, as they are frequently employed in design thinking, bringing together experts with various backgrounds to collaborate and communicate effectively. These solution-finding techniques result in haptic representations of the designed idea. While creating these results, the haptic modelling methods facilitate the participants' communication and exploration of ideas.

While haptic techniques used in small teams facilitate co-creation and foster creativity, they lack the ability to automatically capture the produced haptic designs in a changeable and processable way. This limitation hinders the later adaptation of results and the understanding of stakeholders who could not participate. Conversely, purely digital online tools enable global cooperation and provide a digital representation of the work, but they hinder the co-creation and creative flow that is crucial in workshops.

This thesis addresses this issue by introducing the concept of digital twins from design thinking, allowing the automatic creation of a digital and processable representation of the haptic model. Central to this approach is conceptual modelling, allowing ideas to be represented in a structured and computer-understandable way. This method enhances physical workshops by enabling the automatic creation of a digital twin representation of the physical artefacts. Digital models not only represent the ideas but also facilitate automated processing, subsequent adaptation, and easy sharing.

Conceptual modelling structures knowledge in an organized manner, but applying it in the context of physical workshops presents challenges. These include dynamically adapting the metamodel to the changing domain typical of such workshops. Additionally, the haptic models must be mapped to the dynamically changing metamodel of the modelling tool to enable the automated transformation. In this context, the modelling tool must be able to adapt to different workshops and support varying functionality. Finally, a computer-understandable representation is essential for processing the models, further supporting the users.

This thesis introduces a framework for dynamically adapting a modelling environment to support haptic modelling in workshops and digital twins from design thinking. Dynamically adapting means that the metamodel and the modelling environment can be adapted during its usage to fit the current workshop's needs.

Throughout this thesis, a conceptual design and a proof-of-concept prototype were developed. The prototype was tested in guided workshops and evaluated using questionnaires to gain insights into its applicability. The encouraging results demonstrate that combining haptic and digital conceptual modelling is feasible and beneficial.

Kurzfassung

Die Findung von innovativen Lösungen für komplexe Probleme ist eine herausfordernde Aufgabe, da es für diese keine offensichtlichen Antworten gibt. Kreativtechniken, wie sie oft in physischen Design Thinking Workshops verwendet werden helfen, da sie effektive Zusammenarbeit von Experten mit diversen Hintergründen ermöglichen. Diese Kreativtechniken resultieren in einer haptischen Darstellung der innovativen Idee. Während der Erstellung dieser Darstellung, erleichtern die haptische Modellierung die Kommunikation und die Erkundung von Ideen durch die Teilnehmer*innen.

Physische Workshops ermöglichen gute Zusammenarbeit in kleinen Gruppen, um innovative Lösungen zu finden. Hier fehlt es aber an Möglichkeiten, diese Ideen effektiv, verarbeitbar und veränderbar festzuhalten. Dadurch können die physischen Resultate der Workshops, später nur umständlich automatisiert verarbeitet oder angepasst werden und es erschwert Menschen, die nicht an dem Workshop teilnehmen konnten, die Idee ausreichend zu verstehen. Die Verwendung von webbasierten Tools ermöglicht eine globale Zusammenarbeit und ein digitales Resultat. Aber eine rein digitale Zusammenarbeit erschweren den kreativen Fluss, welcher wichtig ist, um innovative Lösungen zu finden.

Um dieses Problem zu lösen, stellt diese Arbeit Digitale Zwillinge für Design Thinking vor. Das Fundament dieses Ansatzes basiert auf toolgestützter, konzeptioneller Modellierung, welche verwendet wird, um die physischen Resultate eines Workshops digital darzustellen. Modelle können erweitert, automatisiert verarbeitet und leicht verteilt werden. Um diesen Ansatz anwendbar zu machen, wird das digitale Modell als Kern des digitalen Zwillings automatisch vom physischen Artefakt abgeleitet.

Konzeptionelle Modellierung wird zur strukturieren Darstellung von Wissen verwendet. Diese in kreativen Workshops einzusetzen, birgt Herausforderungen, da die abzubildende Domäne zu Beginn des Workshops noch nicht feststeht und sich dynamisch ändert. Diese Änderungen müssen während der Modellierung im Metamodell beachtet werden, vor allem da die physischen Artefakte dem Metamodell zugeordnet werden müssen. Ohne diese Zuordnung kann die automatisierte Verarbeitung nur unzureichend unterstützt werden.

In dieser Arbeit wird ein Rahmenwerk für die dynamische Anpassung einer Modellierungsumgebung zur Unterstützung haptischer Modellierung in Workshops und digitaler Zwillinge für Design Thinking vorgestellt. Dynamische Anpassung bedeutet, dass das Metamodell und die Modellierungsumgebung während ihrer Nutzung an die aktuellen Bedürfnisse des Workshops angepasst werden können.

Diese Arbeit stellt die Konzeptionalisierung des Ansatzes und ein Prototyp vor, welche in geleiteten Workshops und mit Fragebögen evaluiert wurde. Die gefunden Resultate sind vielversprechend was die Kombination von physischer und digitaler Modellierung und deren Mehrwert betrifft.

Contents

Acknowledgements	i
Abstract	iii
Kurzfassung	v
List of Tables	xi
List of Figures	xiii
List of Algorithms	xvii
Listings	xix
1. Introduction	1
1.1. Thesis Contribution	3
1.2. Related Work	5
1.3. Research Approach	7
1.3.1. Research Objectives	7
1.3.2. Design Science Research	9
1.3.3. Literature Review	11
1.4. Addressee of the Thesis	13
2. Fundamentals	15
2.1. Design Thinking	15
2.1.1. Design Thinking Processes	18
2.1.2. Tangible Design Thinking Techniques	21
2.2. Digital Twin	25
2.2.1. Repurposing Digital Twins for Design Thinking	27
2.2.2. Tools for Creating and Using Digital Representations in the Context of Design Thinking	29
2.3. Conceptual Modelling	31
2.3.1. Diagrams	35
2.3.2. Modelling Language and Methods	36
2.3.3. Metamodelling	40
2.4. Artificial Intelligence Technologies	55
2.4.1. Ontology	55
2.4.2. Object Detection	57

2.4.3. Text Recognition	66
2.4.4. Sentiment Analysis	66
3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling	69
3.1. Digital Twin in the Context of Design Thinking	72
3.2. Added Value of Tangible Modelling	74
3.3. A Tangible Design Thinking Technique Example	75
3.4. Adaptation at Runtime	76
3.4.1. Adapt Domain Concepts	77
3.4.2. Extend the Modelling Method Capabilities	78
3.5. Handling of an Adaptive Modelling Environment	83
3.6. Summary	86
4. Requirements for the Adaptive Modelling Environment	87
4.1. Origin Areas	88
4.1.1. Application Domain	88
4.1.2. (User) Interaction	90
4.1.3. Modelling Method	91
4.2. Requirements for Adaptive Modelling through a Literature Review	92
4.2.1. Literature Review Procedure	93
4.2.2. Requirements from the Literature Review	96
4.3. Requirements for the Design Thinking Digital Twin	109
4.3.1. Requirements for Using Design Thinking Digital Twins in Workshops	115
4.4. Consolidated Requirements	118
4.4.1. Adaptable Modelling Environment	120
4.4.2. Adaptability	120
4.4.3. Automated Adaptation	121
4.4.4. Avoid Redundancy	121
4.4.5. Comprehensible Notation	121
4.4.6. Configurable Mechanisms	122
4.4.7. Consistency	122
4.4.8. Extendability	123
4.4.9. Flexibility	123
4.4.10. Fundamental Concepts	124
4.4.11. Implementation Agnostic	125
4.4.12. Integrability	125
4.4.13. Longevity	125
4.4.14. Meta Model Provenance	125
4.4.15. Model Value	126
4.4.16. Multi View Support	126
4.4.17. Ontological Complete and Clear	126
4.4.18. Processability	127
4.4.19. Reusability	127

4.4.20. Scalability	127
4.4.21. Simplicity	127
4.4.22. Tooling	128
4.4.23. Usability	128
5. The Adaptive Modelling Environment	129
5.1. Adaptive Modelling Environment Procedure	129
5.1.1. Interaction Setting for the Adaptive Modelling Environment	129
5.1.2. Instance Based Metamodelling within the Adaptive Modelling Environment	132
5.1.3. Using the Adaptive Modelling Environment in Physical Workshops	134
5.2. A Metamodel for the <i>Adaptive Modelling Environment</i>	141
5.2.1. Fundamental Structures in Design Thinking Techniques	142
5.2.2. A Metamodel for the Adaptive Modelling Environment	151
5.2.3. A Implementation-Oriented Metamodel for the Adaptive Modelling Environment	153
5.3. Modelling Tool Related Capabilities	157
5.3.1. Adaptation Capability	157
5.3.2. Providing Comprehensible Notation	163
5.3.3. Extending the Modelling Method	164
5.3.4. Configure Mechanisms	165
5.3.5. Ensure Consistency	166
5.3.6. Instance Modelling	166
5.3.7. Linking Multiple Models	167
5.3.8. Processable Representation	168
5.3.9. Logging Changes	169
5.3.10. Model Sharing	169
5.4. Design Thinking Digital Twin Capabilities for Using the <i>Adaptive Modelling Environment</i> in Workshops	170
5.4.1. Transforming Tangible Models into Digital Twins	170
5.4.2. Analyse Feedback of Conceptual Models	197
5.4.3. Model Transformation to Enhance captured Knowledge	200
5.5. Conceptual Architecture of the <i>Adaptive Modelling Environment</i>	210
5.5.1. Modelling Tool	212
5.5.2. User Interface	213
5.5.3. Provider Component	214
5.5.4. Object Identification	214
5.5.5. Relation Identification Component	215
5.5.6. Concept Enrichment Component	215
5.5.7. Evaluation Component	216
5.5.8. Model Transformation	216
5.5.9. Preparation Component	216

6. Evaluation with the Scene2Model Prototype	219
6.1. The Scene2Model Proof-of-Concept Prototype	220
6.1.1. Used Technologies for Implementing the Scene2Model Prototype	220
6.1.2. Usage Phases of the Scene2Model Prototype	222
6.2. Questionnaire Based Evaluation	229
6.2.1. Second Workshop Evaluation	230
6.2.2. Third Workshop Evaluation	245
7. Conclusion and Outlook	257
7.1. Conclusion	257
7.2. Outlook	260
Bibliography	263
Acronyms	280
A. Appendix	281
A.1. Knowledge Base	281
A.1.1. Assignment of Abstraction Layer to Recognised Objects with RDF	281
A.1.2. Assignment of Domain Tags to Recognised Object with RDF	283
A.2. A Metamodel for Design Thinking Techniques	284
A.3. A Implementation-Oriented Metamodel for <i>Adaptive Modelling Environment</i>	284
A.4. Used Technologies and Methods	287
A.4.1. ADOxx	287
A.4.2. CoreNLP	287
A.4.3. DarkMark	287
A.4.4. Darknet	288
A.4.5. EasyOCR	288
A.4.6. FFmpeg	288
A.4.7. GraphDB	288
A.4.8. imgaug	289
A.4.9. lxml	289
A.4.10. Olive Microservice Framework	289
A.4.11. Python opencv	289
A.4.12. Yolo	290
A.4.13. Designing Metamodels with CoChaCo	290
A.5. Workshop Materials and Evaluation	291
A.5.1. Pre-Defined Smart Mobility Scenario	291
A.5.2. Questionnaire for the Second Workshop Evaluation	293
A.5.3. Questionnaire for the Second Workshop: Open Question Answers	302
A.5.4. Questionnaire for the Third Workshop Evaluation	303

List of Tables

2.1. Overview of different kinds of digital twins and their features	28
4.1. List of the overall found publications in the given platforms based on the provided search terms	95
4.2. List of publications after the screening of the titles and abstracts	97
4.3. Table with an overview of the requirements found in the literature review	110
4.4. Table with an overview of the requirements derived from the <i>Adaptive Modelling Environment</i> as introduced in chapter 3	114
4.5. Table with an overview of the requirements to process semantic for the <i>Adaptive Modelling Environment</i> as introduced in chapter 3	118
4.6. Table with an overview of the requirements consolidated from the literature (see section 4.2) and the requirements identified for the design thinking digital twin (see section 4.3)	119
5.1. List of design thinking techniques which were analysed to build the abstract metamodel	143
5.2. List of the available comparison operators for the pseudo rules and their explanation	203

List of Figures

2.1. Visualisation of the Double Diamond Model from the Design Council [1] (published under the <i>CC BY 4.0 license</i>)	21
2.2. Modelling method components form the metamodeling framework of [2, p.3])	37
2.3. Visualisation of a three-layered model hierarchies	41
2.4. Conceptual representation of the usage of the ontology	56
2.5. Example hierarchy of classes and an instance	57
2.6. Visual representation of the application of a kernel. The representation is based on a lecture by Jeremy Howard titled Lesson 4: Practical Deep Learning for Coders, which is available on YouTube. [3]	62
2.7. Visual representation of the application of the pooling operation. The representation is based on a lecture by Jeremy Howard titled Lesson 4: Practical Deep Learning for Coders, which is available on YouTube. [3] . .	63
3.1. The <i>Adaptive Modelling Environment</i> (the figure contains pictures from SAP Scenes TM [7])	70
3.2. Concept of directly posting models to social media platforms and retrieving analysed feedback (the figure contains pictures from SAP Scenes TM [7]) .	81
3.3. Example metamodel for extending the modelling environment, which is an extension to the metamodel introduced in [4, 5] (visualisation done with CoChaCo, described in appendix A.4.13 or [6])	82
3.4. Example possibilities for interacting with the adaptive modelling environ- ment (the figure contains pictures from SAP Scenes TM [7] and pixabay.com)	84
5.1. Visual representation of physical and digital models and their encoded spatial information	131
5.2. Visualisation of an example for instance based metamodeling	133
5.3. Visualisation of the roles involved in using the <i>Adaptive Modelling Envir- onment</i> in workshops	135
5.4. Visualisation of the guideline for using the <i>Adaptive Modelling Environment</i> in workshops	138
5.5. Fundamental structure of design thinking techniques: Concept Types . . .	145
5.6. Fundamental structure of design thinking techniques: Concept Description	146
5.7. Fundamental structure of design thinking techniques: Nested Canvas . . .	147
5.8. Fundamental structure of design thinking techniques: Sequence	148
5.9. Fundamental structure of design thinking techniques: Tree	149
5.10. Fundamental structure of design thinking techniques: Layer	150

List of Figures

5.11. Fundamental structure of design thinking techniques: Method Defined Layout	151
5.12. Fundamental structure of design thinking techniques: Unrestricted	151
5.13. Abstract metamodel for design thinking techniques	153
5.14. Implementation-oriented metamodel for the <i>Adaptive Modelling Environment</i>	154
5.15. Visualisation of an example of 3 abstraction layers for recognising physical objects	173
5.16. Visualisation of the two concepts of how relations can be recognised in the physical objects	175
5.17. Visualisation of how the coordinate system, the bounding box and the bounding box's parameters are used in this thesis.	175
5.18. Conceptual representation of the artefacts which are needed as input for the modelling environment to use the machine learning based object identification approach	192
5.19. Graphical representation of the training process to create a trained CNN .	193
5.20. Example of tags created with ArUco [8]	196
5.21. Conceptual architecture for the <i>Adaptive Modelling Environment</i>	211
5.22. Conceptual architecture for the <i>Adaptive Modelling Environment</i> , enhanced with instances for the components	212
6.1. Visual Representation of usage phases of the prototype (the figure contains pictures from SAP Scenes TM [7])	222
6.2. Example of automated image augmentation with the <i>imgaug</i> library (the figure contains pictures from SAP Scenes TM [7])	225
6.3. Visualisation of the recognition workflow using pre-defined objects for relation identification (the figure contains pictures from SAP Scenes TM [7])	226
6.4. Bar charts on how knowledgeable the participants were about design thinking and conceptual modelling	234
6.5. Bar charts of the answer to the question <i>Reflecting on the concepts that were needed during the working session had a positive influence on the created idea/design.</i> in the second evaluation	235
6.6. Bar charts of the answer to the question <i>Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.</i> in the second evaluation	236
6.7. Bar charts of the answer to the question <i>The added value from adopting the figure outweighs the additional effort.</i> in the second evaluation	236
6.8. Bar charts of the answer to the question <i>The digital model outperforms a photograph in capturing the knowledge collected in the workshop.</i> in the second evaluation	237
6.9. Bar charts of the answer to the question <i>Combining paper figures and digital models improves the overall experience.</i> in the second evaluation . .	237
6.10. Bar charts of the answer to the question <i>The tangible paper figures foster creativity and cooperation in the workshops.</i> in the second evaluation . . .	238

6.11. Bar charts of the question <i>I was able to represent all the notational concepts that I wanted to represent.</i> compared between workshops with and without pre-defined figures	239
6.12. Bar charts of the question <i>The existing concepts in the tool were not sufficient for modelling the desired scenario(s).</i> compared between workshops with and without pre-defined figures	240
6.13. Bar charts of the question <i>It was easily possible to add additional concepts in the tool according to my needs.</i> compared between workshops with and without pre-defined figures	241
6.14. Bar charts of the question <i>Based on the time I spent using the tool, I would require more time to feel familiar with its use.</i> compared between workshops with and without pre-defined figures	242
6.15. Bar charts of the question <i>These instructional videos helped me when using Scene2Model.</i> , evaluating the teaching videos provided to the participants.	243
6.16. Pie charts for the answers to two questions regarding the video	244
6.17. Bar charts of the answers to the questions <i>How knowledgeable are you in the topic of design thinking?</i> and <i>How knowledgeable are you in the topic of conceptual modelling?</i> in the third evaluation.	249
6.18. Bar charts of the answer to the question <i>The haptic design thinking environment was beneficial in generating innovative scenarios.</i> in the third evaluation.	250
6.19. Bar charts of the answer to the question <i>The Scene2Model tool effectively supported the digitalisation of the created scenarios in the haptic design thinking environment.</i> in the third evaluation.	250
6.20. Bar charts of the answer to the question <i>Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.</i> in the third evaluation.	251
6.21. Bar charts of the answer to the question <i>The added value from adapting the figure outweighs the additional effort.</i> in the third evaluation.	252
6.22. Bar charts of the answer to the question <i>The digital model outperforms a photograph in capturing the knowledge collected in the workshop.</i> in the third evaluation.	253
6.23. Bar charts of the answer to the question <i>The tangible paper figures foster creativity and cooperation in the workshops.</i> in the third evaluation.	253
6.24. Bar charts of the answer to the question <i>Within the workshop and the Scene2Model tool, I was able to represent all the notational concepts that I wanted to represent.</i> in the third evaluation.	254
6.25. Bar charts of the answer to the question <i>Based on the time I spent using the tool, I would require more time to feel familiar with its use.</i> in the third evaluation.	255
A.1. Abstract metamodel for design thinking techniques in landscape format	285

List of Figures

A.2. Implementation-oriented metamodel for <i>Adaptive Modelling Environment</i> in landscape format	286
A.3. Overview of the important classes and relations offered by the CoChaCo modelling method	291

List of Algorithms

1.	Pseudocode for the CNN based identification of physical objects	179
2.	Pseudocode for the <i>consolidateFoundObjects()</i> function from Algorithm 1. The functions used in this algorithm can be found in Algorithm 3 and Algorithm 4.	181
3.	Pseudocode for the <i>calculateOverlapArea()</i> function, which is used in Al- gorithm 2	182
4.	Pseudocode for the <i>removeDuplicates()</i> function, which is used in Algorithm 2185	
5.	Pseudocode for identifying relations from physical objects and their position	188
6.	Pseudocode for setting the endpoints of an identified relation from Algorithm 5189	
7.	Pseudocode for identifying relation based on proximity and pre-defined knowledge	191
8.	Pseudocode for the tag-based identification of physical objects	197
9.	Pseudocode for the automated feedback analysis	199
10.	Pseudocode for the <i>gatherFeedback()</i> function of the automated feedback analysis	200

Listings

5.1. Example from Figure 5.15 insert in knowledge base	177
5.2. Reasoned abstraction layer assignement to the knowledge base defined in Listing 5.1	178
5.3. Examples of assigning domain tags, based on the example concepts from section 5.4.1.1.1	184
5.4. Model Mapping	204
5.5. Object Mapping	204
5.6. Relation Mapping	205
5.7. Attribute Mapping	205
5.8. Relation to Object Mapping	206
5.9. Object to Relation Mapping	206
5.10. Conditional Object Mapping	207
5.11. Conditional Attribute Mapping	208
5.12. Conditional Relation Mapping	209
A.1. RDF knowledge base example for the abstraction layers of recognised Objects (example based on Figure 5.15)	281
A.2. SPARQL query to get the abstraction layer of a concrete object	282
A.3. SPARQL query to get the abstraction layer of a class	282
A.4. RDF knowledge base example for the assignment of domain tags to concepts (example based on Figure 5.15)	283
A.5. SPARQL query to get all domain tags	283
A.6. SPARQL query to get all domain tags of <i>Engineer</i>	284

1. Introduction

Recent and ongoing technological progress offers many innovative and exciting opportunities for enterprises, which must keep up and adapt to stay relevant. This is a daunting challenge, as new technologies must be applied to increasingly complex environments due to their integrated and fast-changing nature.

The complexity stems not only from using novel technologies but also strongly involves human influence, both from within and external to the organisation. For the staff, new ways of working must be found, which can be facilitated by technology, like it was done during the pandemic. For an organisation's connection to external stakeholders or organisations, new ways of interaction evolve. Both are based on using technologies in innovative ways to improve their own business.

To tackle these tasks, not only the technology must be understood, but the environment, the real problems, and the domain as a whole must be thoroughly comprehended [9]. Without this understanding, the problem can not be solved entirely. Such problems are known as “wicked problems” [10, 11], and have been research topics since the 1970s. However, they are still relevant today [12, 13]. They are problems that are not clearly defined, are influenced by various stakeholders with different and sometimes opposing interests, and the available information is incomplete. Therefore, for them, there does not exist a best solution, saving the problem for everyone.

Even if a best solution cannot be found, these problems must be addressed. Designers are already successfully tackling this problem [11]. However, only some problems can be tackled by an experienced and gifted designer. Therefore, support is needed to tackle such problems, independent of their experience with design. Here, design thinking comes into play as an approach supporting and guiding the designing of innovative solutions by providing a mindset, procedures and a toolset [9, 14].

These are necessary because innovative ideas do not form randomly and independently in an expert mind but result from hard work involving many experts with various backgrounds [9]. Design thinking was developed from the need to support these experts and tackle wicked problems [11]. Involving multiple experts allows them to cover various fields and domains but creates a need to support their communication and establish a shared understanding of the problem and the solution approach [15]. Therefore, design thinking uses physical workshops, where tangible creative techniques support the participants in externalising their understanding, exploring different possibilities and communicating. Having visualisations of ideas also helps to make explicit knowledge implicit [16]. The workshop result is a tangible representation of the idea and a shared understanding by the participants. Using physical objects for the representation additionally allows the rearrangement of aspects of the idea and the exploration of different options [14].

1. Introduction

However, as the workshop size should be restricted so as not to hinder the cooperation [17, 18], this means that other involved stakeholders do not have access to this knowledge that easily. Mainly because what can be usually shared is a picture of the physical representation and maybe some textual description. Here, an adaptable representation would improve later utilisation compared to a static picture.

Having a static picture of the physical representation cannot be used as profoundly during later implementation work the idea [19]. Further, having machine-interpretable representation not only allows one to adapt it later on but to relate the representations in the beginning, which are on a high-abstraction level, with artefacts created during the implementation [20]. This way, the progress of design and implementation can be linked to provide better insights.

Therefore, not only is the idea creation based on expert communication in small groups important, but it is also essential to know how this knowledge can be captured and shared with other stakeholders. One well-established approach for describing knowledge in a human and machine-understandable way is conceptual modelling [21]. It often uses visual modelling languages to describe systems under study on an abstraction level tailored to its users. Conceptual modelling is already used in various areas like software engineering [22], enterprise modelling [23], teaching stochastic [24] or design thinking [25, 26].

Conceptual modelling is used to design, implement and analyse organisations [27]. It facilitates the documentation of important aspects of organisations, like processes, and supports users' understanding of the system. The expressiveness of the models, created with conceptual modelling languages, can be increased if the language uses concepts from the domain it is used for [28, 29, 30]. Such languages are then called domain-specific modelling languages.

The understandability of these models is based on the specification of the modelling language, as it defines the concepts which can be used to describe the system under study and their meaning [31]. Metamodels are used as a core for the modelling language definition [32, 33]. This core definition of the language makes the models understandable to humans and machines who are familiar with it. The metamodel can also be used as the foundation for a digital modelling tool [2], further increasing the model value for its users [34]. The modelling tool must be fitted to the users rather than the other way around [35].

Therefore, this thesis combines design thinking and its usage in physical workshops with digital conceptual modelling to support the design of innovative solutions. These two approaches fit together, as both use abstraction and visual representation to capture and communicate knowledge. However, the two approaches should not be used after one another but integrated to improve overall support.

One crucial aspect supports the users in creating the digital model, as they were already created with the physical representation. This is important because people do not want to model everything from scratch but prefer to generate them from existing data [36]. An example from the domain of software engineering is that many software engineers do not want to create UML models but find sketches helpful during the design [22], as they can be used to communicate in teams and offer freedom in their design. The usefulness of such

sketches can be further improved if they can be saved and used during the implementation to analyse how well the overall idea is implemented.

Therefore, one crucial goal when combining physical design thinking workshops with digital conceptual modelling is that the digital models are created automatically from the physical representation so that the users do not have to rebuild them manually.

An instant digital representation enables utilisation during the workshop by allowing one to easily share a comprehensible and enriched representation. It can also be used as input for intelligent systems, triggering new participant discussions.

Conceptual modelling methods are often rigorously defined so that the models can be better understood and processed. However, there are situations where users need flexibility and the ability to represent ideas without learning a rigorous method [37]. The representations in design thinking belong to this group, as the solutions in its domain are not fully known during the workshop. In such cases, the modelling method must be dynamically tailored to the users' needs to fit their needs better [28]. This includes the visual representation and the meaning, attributes, and relations within the model, leading to adaptations of the metamodel. The creation of the metamodel and derived models should not be separated but enabled together. Therefore, even if models are available, the metamodel should be adaptable [38]. Further, as a digital tool is used, the changes to the models and metamodels must also be seamlessly applied to the tool [39].

Even though not limited to, the visual representation of the used concepts is essential in the context of design thinking, as they provide the semantics to the human users. Therefore, in this context, not only abstract visualisation should be used, but some that have meaning in the domain and for the users [40].

It should also be mentioned that designing a sophisticated and rigorous modelling method is a complex and time-consuming task [41]. Therefore, expertise in designing modelling methods is needed, which cannot be assumed by the participants in the workshop. Therefore, the goal is not to let the participants define such a modelling method for the domain representation but to create a modelling method and the models on a higher abstraction. The metamodel created in this context provides context for the created models. This also means that more than providing a metamodeling platform to the participants is needed, as they need more freedom and guidance in creating the models with the corresponding metamodels.

1.1. Thesis Contribution

This section introduces the contribution of this thesis and briefly discusses the core concepts to tackle the problems and goals introduced above. This concept is discussed more in chapter 3. The introduction will provide an overview of what this thesis will accomplish and provide context for the related work (see section 1.2, the research approach (see section 1.3 and the fundamentals (see chapter 2), which are discussed below.

The novelty of this thesis is the design for a digital environment facilitating synergies between digital and haptic modelling. In this context, digital modelling means the creation of digital conceptual models (see section 2.3) within a modelling tool. Haptic modelling

1. Introduction

means the creation of haptic or physical representation (e.g., using paper figures or sticky notes) during a physical design thinking workshop (see section 2.1). We will also use physical modelling or haptic modelling as synonyms for haptic modelling.

Therefore, the notion of digital twins (see section 2.2) will be repurposed for design thinking to create *Digital Twins from Design Thinking* (see section 3.1). Within digital twins from design thinking, the haptic model (which will also be called tangible in this thesis) will be automatically transformed into a digital model, making it directly available within and after the workshop. Functionality based on the digital model, which is the core of the digital twin, then influences the idea created within the workshop.

The digital twin can further be used straightforwardly after a workshop for documentation and sharing it with stakeholders, but they cannot participate in the workshop. The model can also evolve alongside the idea and can be related and transformed into models from other modelling languages to add the necessary information needed to implement the created idea.

In this context, the transformation from the tangible to the digital model is a key element, as it reduces the repetitive and manual labour of the participants so that they can focus on the essential aspects.

This thesis introduces a framework for supporting haptic modelling methods used in physical workshops. This framework enables the creation of the digital twins from design thinking and will be embedded in the design of a modelling environment and how to use it. One core aspect of this framework is its dynamic adaptability, meaning that both the haptic and digital models can be adapted during its usage. For the digital modelling, this includes the metamodel and the modelling environment for the digital modelling environment.

The environment used to create the digital twin from design thinking will be based on conceptual modelling as its core and called *Adaptive Modelling Environment*. Adaptive in this context means that the metamodel and the modelling method can be adapted during its usage, enabling a tailored domain representation. However, the modelling environment can also be adapted in the way it works and which functionality it offers further to tailor it to the needs of the workshop. For example, users can choose different ways to transform tangible models into digital ones automatically.

This thesis further distinguishes between the modelling environment and the modelling tool. The modelling tool is an application focused on creating and managing conceptual models. This includes the model editor, saving models, exporting pictures and similar. The modelling tool is a core element of the modelling environment, which offers additions to the modelling tool. The modelling environment has additional components for interacting with the physical world or other applications. This decoupling from the modelling tool and the additional components was made to keep the modelling environment flexible and adaptable to different situations. The modelling environments and their components are described in more detail in chapter 5.

This thesis also introduces how these concepts can be used in and after workshops in the context of the digital twin from design thinking and the *Adaptive Modelling Environment*. This includes a prototypical implementation, which was used to evaluate it in workshops

using questionnaires.

Therefore, the contribution of this thesis is the design of the *Adaptive Modelling Environment* for the usage in the context of digital twin from design thinking. The novel aspect is the combination of tangible and digital modelling.

1.2. Related Work

This section focuses on approaches and tools in the context of this thesis topic and has some similarities. The goal is to point out similar approaches and why this thesis differs. Therefore, the section does not contain literature on concepts used as a foundation. These topics are introduced in the *Fundamentals* section (see section 2).

First of all, we found no other approach which supports the creation of tangible representation in tangible workshops, together with an automated transformation into digital, conceptual models, so that the digital model can be used instantly within the workshop and afterwards.

Modelling tools exist to create digital versions of the physical representations manually. MELCA [42] and its enhanced version CuTiDe [25] are conceptual modelling tools which enable users to create diagrammatic conceptual models using a GUI. Here, not only can different digital diagrams be created, but they can also be linked to each other, and the tools implement functionality supporting the creation. However, these tools do not support working in physical workshops, and they are also not concerned with dynamically adapting the modelling methods to specific workshops. Therefore, the tools can be used as they are or reimplemented.

The AOAME tool introduced by Laurenzi et al. [43] is a web-based modelling tool which uses ontologies to define the metamodel and support an adaptation of modelling methods during its usage. It was initially designed to offer modelling methods from the enterprise modelling domain, like BPMN, but was also used in the context of design thinking [26]. This tool offers a GUI in which models can be created, and adaptations to the metamodel can be made. Therefore, it allows the user to adapt the modelling method on the fly and enables the definition of rules on the metamodel that can be applied to the models.

This approach does not focus on supporting the usage in a physical workshop and how the models with the adapted metamodels can be shared with other stakeholders. Therefore, it supports tailoring the modelling language to the specific needs but does not focus on using it together.

In the context of creating sketches and visual representations digitally in a team, there also exist collaborative online tools like *Mural*¹ or *Sketchboard*². With such tools, online workshops can be held, where multiple people can work together to create a visualisation of an idea. Even though they enable collaboration over a considerable distance, they do not support physical workshops, and they do not create models but diagrams. Therefore,

¹<https://www.mural.co/>, visited: 24.08.2023

²<https://sketchboard.io/>, visited 24.08.2023

1. Introduction

they do not capture meta information, but the semantics rely on the interpretation of the human users and are encoded in graphical positions, labels and added textual notes.

Wüest et al. [19] introduce the FlexiSketch tool, which uses touch screens to allow users to draw digital sketches, allowing freedom in creating the visualisation. The software then recognises the different drawn objects, which can be enhanced with metamodel information during usage and afterwards reused. FlexiSketch can be used in a physical workshop, but the participants would use touchscreen devices to draw the sketches and, therefore, do not support physical representations. Further, the tools offer free drawing, which usually leads to diagrams that are not fully semantic-rich, as abstract forms are often drawn. For this thesis, semantic-rich pictures should be used in the physical and digital representation to increase user visual interpretation.

Gumienny et al. [44] introducing Teleboard, a digital environment based on digital whiteboards and video streams to enable workshops over vast distances. With this tool, a physical workshop should be emulated, allowing people to draw with their hands and share it with people in the same room or distant locations. The authors did not mention whether a metamodel could be assigned, but they focused on long-distance collaboration. However, this solution depends on sophisticated hardware, such as digital whiteboards, which must be available in each location. Further, such digital whiteboards do not necessarily support the idea that people work simultaneously on the same digital whiteboard if they are in the same room.

Regarding the creation of metamodels out of example models, López-Fernández et al. [45] introduce a method, using multiple circles of creating multiple example models through domain experts and creating a metamodel and modelling language out of it by involving modelling language engineers. In this thesis, we also use the model instances to derive or map them to metamodels, but we follow a more lightweight approach, similar to the one introduced by Wüest et al. [19]. Therefore, the focus is not on creating a highly sophisticated and rigorous modelling method but on capturing additional information through the metamodel. The gathered metamodel may be used to create a more sophisticated modelling method, but this is not the sole purpose of the approach introduced in this thesis.

Last but not least, in the context of creating models with metamodel, also multi-level modelling [46, 47] should be mentioned. Shortly put, in multi-level modelling, multiple ontological metamodel levels can be created on top of the model. These ontological metamodel levels are abstractions of the system under study, further describing the system under study. Ontological in this context means that the focus is on defining abstractions of the domain regarding classes, attributes or relations and not on language constructs. In multi-level modelling, the models and metamodels fuse to a common artefact, allowing a more sophisticated domain description but simultaneously increasing complexity. Therefore, this approach is helpful for experts to describe a system under study rigorously. However, in the context of this thesis, where expert knowledge and experience in metamodelling can not be presumed, there are better tools than such a complex approach. The models and metamodels created in the context of this thesis focus on a flexible and lightweight description.

1.3. Research Approach

This section will introduce the research approach used within this thesis. The core research approach is oriented toward Design Science. Additionally, a literature review was conducted to specify the requirements. Therefore, this section will first focus on design science research and then introduce the literature review approach.

1.3.1. Research Objectives

This section will introduce the research goal of this thesis. This will be done by filling out the problem statement template introduced by Wieringa [48] and then formulating and explaining the research questions.

The problem statement template introduced by Wieringa [48] helps to define the problem, its context and the strived-for goal. It provides four important aspects that must be filled out to define the research objectives:

- problem context
- (re)design an artefact
- requirements
- goal

Below is the filled-out template for this thesis, with the filled-in parts marked in bold. Afterwards, the parts introduced in the template are described.

Improve **knowledge acquisition, enrichment and processing in physical workshops analysing fast-evolving domains**

(problem context)

by applying **digital twins from design thinking based on the adaptive modelling environment**

(artefact)

that **satisfies the usage of haptic modelling in physical workshops and the automated creation of digital twins enabling the capturing and processing of the created designs**

(requirement)

in order to **externalise, enrich, process and communicate the knowledge within the workshop and among all involved stakeholders.**

(goal)

This thesis aims to design a digital modelling environment supporting physical workshops in which design thinking is used to solve complex problems in not fully defined domains. In it, tangible techniques (as introduced in section 2.1 support the design, facilitating cooperation and cooperation among the participants. The results are captured

1. Introduction

in tangible representations, for example, made out of paper figures or sticky notes. In this context, the digital environment enables the automated creation of digital twins from design thinking derived from physical representations. The design thinking digital twin supports the participants during and after the workshop by offering a processable representation of the idea, which can be utilised during and after the workshops. Chapter 3 introduces the idea of digital twins from design thinking in more detail.

Conceptual modelling (as introduced in section 2.3) will be used as the foundation of the design thinking digital twin, enabling a semantic-rich and processable representation. As conceptual modelling is often used to rigorously describe systems under study with pre-defined modelling methods, using it in the context of this thesis, where the domain is not fully known, raises new requirements for modelling and supportive tools. The used modelling methods and the underlying metamodel must adapt to the changing requirements of the workshops to fully capture all aspects and can be used as the foundation for design thinking digital twins.

The research questions below have been formulated to narrow down the focus of this thesis's research. They focus on the digital environment rather than improving tangible techniques. The tangible techniques are used as they are for this research, and the digital environment is designed to support them. Changes to the tangible technique are only made if necessary for applying the digital environment. Therefore, the research questions focus on conceptual modelling and its supportive tools and how they can be used in the context of this thesis.

RQ1: Which requirements must an adaptive modelling environment fulfil to support a dynamic adaptation of modelling methods and the tool during the usage?

Creating domain-specific modelling methods or languages is a complex task on its own, for which much research has already been done. The first research question focuses on the requirements for creating and adapting modelling methods that apply to an adaptive modelling environment. This is needed so that the created modelling methods can create value for its users. Not only the initial creation is important, but also how (especially domain-specific) languages can grow together with their represented domain, as motivated by Karagiannis [28]. [49] and [50] argue that it is also essential to consider adaptations which are only applied by a sub-group of users, which should also be sharable with other stakeholders.

RQ2: How can metamodel adaptation mechanisms of an adaptive modelling environment be used by users during the usage of the modelling environment?

This research question focuses on the interaction of the users with the adaptive modelling environment, which is used by domain experts, who often have limited knowledge about metamodeling or modelling method creation/adaptation. Allowing them to dynamically adapt the provided modelling languages to meet their needs better poses the question of how they are interacting with and how much information they need about metamodeling.

RQ3: Which internal aspects of a modelling environment (e.g., the metamodel) must be considered and held consistent if the adaptation of the modelling envir-

onment is possible at runtime? How can these adaptations be implemented to ensure that no unwanted states of the tool and metamodel can occur?

The question focuses on the internal structure of an adaptive modelling environment and how the offered mechanisms can help ensure that only wanted states can be reached after an adaptation. This is for functionality available by default or added to tailor the functionality to specific cases. In the context of this question, the analysis focuses on how the implemented modelling method's internal structure can be structured and adapted to fit the needs of the adaptive modelling environment used for creating design thinking digital twins.

RQ4: *What is a fitting architecture for the adaptive modelling environment and enables the usage in the context of design thinking digital twin and the corresponding metamodel adaptation at runtime?*

This research question focuses on the information system's architecture and other design decisions needed to enable the adaptive modelling environment and its usage in the context of design thinking digital twins. The individual components are identified and how they must interact to allow the usage with the combination of the physical workshop and the digital environment, especially with the automated transformation from the physical to digital representations.

RQ5: *How can a newly created or adapted modelling method be shared with other instances of the adaptive modelling environment, enabling the user to share it?*

This research question focuses on how the modelling methods created or adapted from the design thinking digital twin can be shared with other users of the same environment. This includes how the models can be transferred from one instance to another, and the information from the metamodel can be shared. This is important, as the models should be able to contain the same information in each instance in which they are imported.

Additionally, this research question focuses on how the knowledge in the digital models can be shared and reused in other contexts to enable enrichment and support later implementation. For example, how can the design thinking of digital twin models be related to concepts or parts of modelling created with other modelling methods? Alternatively, how can the digital models' knowledge be transformed to be reused in other modelling methods?

1.3.2. Design Science Research

The conducted research is based on *Design Science* as introduced by Hevner et al. [51], and the research process is inspired by Peffers et al. [52].

Design Science is a fundamental research approach used in the information system (IS) domain to create innovative artefacts [51]. It is used to guide the design of artefacts to solve before-identified problems. To distinguish design science from industrial design, it must be used to find solutions to relevant problems for which no established solutions exist. Further, the design must be evaluated, rigorous methods must be used, different

1. Introduction

solutions must be compared, communication must be made, and a contribution to research must be made.

Peffers et al. [52] further introduced a six-step design science process. It starts with identifying the problem and defining the objectives of the artefact that should be created. Then, it is designed, implemented and demonstrated in a fitting context. Afterwards, it should be evaluated and communicated to scientific communities.

In this thesis, the *Adaptive Modelling Environment* is designed, and a proof of concept prototype is implemented. The thesis includes the resulting artefacts, such as the conceptual and structural design and an implemented instance, and discusses how to work with it. The resulting prototype enables usage in physical workshops and creates conceptual models as the core for the digital twin from design thinking. The usage procedure is also part of the overall design and this thesis. The design of these novel results is the core of this thesis, and therefore *Design Science* was used as an applicable research approach.

In order to meet the above-introduced criteria for design science and its process, the research was structured and conducted as described below.

The research started with identifying the concrete problem that should be solved, which is how the design of innovative ideas can be supported, allowing the use of both tangible and digital means in physical workshops. The relevance and motivation for this thesis are discussed in chapter 1.

The objectives, requirements and possible approaches for the designed solutions were gathered by analysing the problem domain and searching the literature. The analysis of the problem domain was done by reading the literature, within the *OMiLAB Community of Practice* (cf. [53]) and the EU-funded *Digitrans*³. OMiLAB is a global community focused on value created through conceptual models and how they can be applied in various domains. The network consists of independent nodes with their own research focus but who participate in knowledge exchange. One aspect focuses also on supporting the design and evaluation of innovative ideas in laboratory environments [54]. Digitrans focused on equipping small and medium-sized enterprises with knowledge and tools for their digital transformation. This included using physical workshops to design solutions to complex problems.

A systematic literature review was conducted after conceptual modelling was identified as a fundamental element of the solution approach. The literature review procedure is described in section 1.3.3. The literature review supported the identification of requirements for adapting modelling methods during their use.

The results of specifying the problem and the identified requirements are described in chapter 3 and chapter 4.

Multiple iterations were carried out for the prototype's design and implementation, analysing different methods and technologies that can be used. The design contains the architecture, the capabilities and the metamodel and is described in chapter 5. Technical details of the prototype are introduced in chapter 6.

The prototype was demonstrated and used on multiple occasions, such as teaching,

³<https://dtp.interreg-danube.eu/approved-projects/digitrans> (accessed: 09-06-2024)

scientific conferences, and scientific cooperations within the OMiLAB community. These were used to improve the design and the prototype and communicate the artefacts within scientific communities. Here, the analysis was conducted by observing the usage and having discussions.

The communication and dissemination activities within OMiLAB are not limited to exchange with and cooperative publications. Here, technical documentation, descriptions and cases for a more general audience are also shared. This was used to communicate the idea, design, and prototype further.

Three evaluations with questionnaires were conducted to improve the gathering of feedback. In the starting phases with an early prototype of the research, workshops with the partners from the EU-funded Digitrans project and with students from the University of Vienna were conducted and evaluated with questionnaires to evaluate the applicability of the idea. Later, with an improved design and prototype, two workshops were held within two different instalments of OMiLAB's Nemo⁴ summer school series. Here, the participants used the prototype and filled out questionnaires. More information on the evaluation can be found in chapter 6.

The conducted research leads to the design and prototypical implementation of an environment to create and use digital twins from design thinking. This allows the combination of physical workshops using tangible methods with a digital representation, which is automatically derived from the tangible method. The designed architecture of the environment supports a flexible adaptation and extension, enabling users to tailor it to their needs. Additionally, new ways of applying conceptual modelling and their tools in dynamically changing environments are introduced as a basis for the introduced environment.

1.3.3. Literature Review

To allow a better inclusion of this thesis into the established scientific corpus, the literature was searched to identify fitting topics in the context of this thesis. This includes identifying and defining the state-of-the-art, identifying the research gap and gathering the fundamentals for the used and neighbouring approaches. Therefore, ad-hoc literature reviews were conducted to familiarise the readers with different topics, like conceptual modelling, image recognition, and metamodeling. This literature was found in various ways, including searching scientific databases, participating in conferences, and getting suggestions from my supervisor or colleagues. The literature review procedure described below was used to answer specific questions through literature. Such a search was used to support identifying the requirements introduced in chapter 4.

Literature reviews are used to answer questions important to the researchers conducting the research, and they usually have a specific target audience in mind. Common goals of literature reviews are summarising, criticising and integrating published findings and their containing outcomes, methods, theories and applications [55].

Snyder [56] differentiates systematic and semi-systematic literature reviews among

⁴<https://nemo.omilab.org/> (accessed: 09-06-2024)

1. Introduction

others. In this thesis, the more general semi-systematic literature review was used to explore and understand the basic fields of interest for this thesis, like the fundamentals. For specific questions, a quantitative systematic approach was used, such as identifying the requirements from the literature. The systematic approach was applied where specific questions should be answered based on the current state of the literature.

Semi-systematic literature reviews do not follow such rigorous procedures but enable the creation of a broader understanding than a specific review [56]. This is often used to analyse a broader domain, where different conceptualisations were used and must be connected by the researcher. Therefore, this approach was used to better understand the underlying topic areas of the research.

Systematic literature reviews focus on rigour and reproducible procedures to answer a question with literature [56]. It is used to identify the relevant publications to be synthesised to answer the question. Procedures in this context are rigorously defined and, therefore, easier reproducible.

The procedure to conduct the systematic literature review in this thesis was based on the approach introduced in [56, 55]. First, the concrete questions must be identified and formulated based on an initial scanning of the available literature. This includes the decision if a literature review is needed in the first place, meaning evaluating the importance of the question and if literature already exists. The initial scanning is also needed to get the first overview of the topic and the terms used. The terms used in a specific research field can influence the concrete formulation of the search terms, as they should be carefully chosen to improve the quality of the search results.

After having a general overview of the literature, inclusion criteria for the articles must be established. Within the planning phase of the literature review, as well as the sources used (e.g. scientific databases, journals), the search terms, publication year, and languages must be defined. In the search itself, it is preferred to search in databases as in specific journals so that relevant literature is not left out unintentionally [55].

Further, it is essential to identify suitable scientific databases [55]. They should focus on the underlying topic, and one way to find them is to identify those databases which allow access to high-ranked journals and conferences within the topic area. Additionally, papers can be found by looking at references provided by fitting papers. This approach is also known as *backward search*.

In information system (IS) literature, the volume of published papers is increasing very fast, which influences the procedure to select papers for further analysis [55]. Here, defining the inclusion criteria and procedure for selecting the papers is especially important. The large number of publications makes it essential to understand why some papers were analysed further and others were not.

During the execution of the literature review and after analysing the first found articles, the defined procedure will be evaluated and adjusted if necessary [56]. This could include adapting the search terms, the searched publication years or similar. To select articles, first, the title and abstract will be read. Afterwards, if the article seems promising, it will be scanned and evaluated to see if it fits the literature review. After the sample of articles is finalised, the framework for abstracting the needed information will be created,

and an overview of the metadata of the literature review will be created. Metadata here means information like the number of articles found, how many publications are taken into the final set, and so on.

After the final set was created, the papers were read in detail, and the needed information was extracted and synthesised to answer the research question [56]. The way of doing this is dependent on the concrete literature review. Furthermore, the results are represented in a comprehensible way and described within the thesis.

1.4. Addressee of the Thesis

This thesis focuses on a digital environment to support physical design thinking workshops through conceptual modelling. Therefore, it addresses people whose professional interests lie in the context of design thinking and conceptual modelling.

From design thinking, it addresses researchers by introducing a novel way to support physical workshops. For developers of new physical design thinking methods, this thesis introduces a way to support them with a digital tool. The environment designed in this thesis was developed to be adaptable to different ones, even though the prototype was for one. However, the designed environment can be used by other developers as the basis for developing their digital twin for their design thinking method.

For the practitioners of physical design thinking, this thesis introduces a novel way to support their work during and after workshops. Therefore, they can apply this to their workshops and get insights into how digital modelling can support them. Additionally, this thesis discusses the influences this thesis has on the procedure.

Finally, for researchers or practitioners of conceptual modelling, this thesis introduces a new application domain where conceptual modelling can be used. Further, it discusses the need to adapt the metamodel during its usage and introduces an approach to accomplish this.

2. Fundamentals

This chapter introduces the fundamentals of the theoretical background and the context for this thesis. The topics introduced here are used within the thesis to derive the contribution of this thesis and to determine the context where and how it should be used.

It starts with an introduction to design thinking as it is the context in which the contribution of this thesis will be applied. Afterwards, the concept of digital twins is introduced, providing a background on which the digital twin from design thinking is based. This is followed by an introduction to conceptual modelling and metamodeling, used as the core of the digital twin of design thinking. Last but not least, the AI technologies are shortly introduced, which are used to implement the prototype.

2.1. Design Thinking

Designing useful solutions requires a thorough understanding of the needs of users and their environment [9, 57]. Here, design includes aesthetic aspects and how a product or service fundamentally works to satisfy the users. Further, design is not limited to products and services, but a skilled team of interdisciplinary designers can also improve the inner workings of an organisation [57].

Buchanan [11] defined the following four areas belonging to design, which go beyond just visualisation:

- *Symbolic and visual communication*: This area focuses on the visual communication of information using different media. For example, it includes graphic design, book production, and communication through film or pictures.
- *Material objects*: This area includes the visual representation of physical objects and the relationship between the objects and humans, which goes beyond visualisation.
- *Activities and organised services*: This area focuses on how design influences and can potentially improve the overall flow within an organisation, including improving the overall satisfaction of involved humans.
- *Complex systems or environments for living, working, playing and learning*: This area focuses on how design can help to improve a complex system as a whole and not just individual parts and functions within it. This also includes the relations and influences between the separate components.

Another difficulty for design is the need for well-designed solutions for complex problems, which are often classified as *wicked problems* [11, 58]. Wicked problems are no new

2. Fundamentals

phenomenon. According to [58], the term *wicked problem* was already used in the 1960s to describe a set of problems which focus on social systems, where the problem itself is ill-defined, includes multiple stakeholders with different goals, often conflicting with each other. Additionally, the problem's context is uncertain, and proposed solutions cannot be evaluated before their implementation, leading to unsatisfactory solution attempts. Till now, no completely satisfactory solution approaches to wicked problems have been defined, and there are still wicked problems that need to be solved.

In wicked problems, one cannot apply a linear method which is rigorously followed to find a solution [10]. The creation of a solution needs flexibility. An understanding of the domain can only be fully completed after the solution is designed, as the design itself may raise its own questions. Such situations are aggravated by the fact that there are no natural final solutions, only better and worse ones. The parameters for accepting a solution are not found within the problem but defined externally, for example, by how much money and time was spent or how well the involved stakeholders receive it.

One example of a wicked problem is the creation of innovative ideas to implement digital transformation or novel business models in organisations. Complex technical and social environments must be understood and aligned with the set goals.

However, innovative ideas do not spontaneously form in the minds of specially gifted people, as often assumed [9]. This usually involves multiple design cycles with multi-disciplinary teams [9, 57, 15], where not all members are necessarily sophisticated in designing innovative solutions. Therefore, such projects need guidance and framework conditions to facilitate the team members.

People with different backgrounds, for example, from different industries and cultures, may need help efficiently working and communicating together [15]. This includes stakeholders from different departments, disciplines and hierarchy levels [14]. But not only within an organisation but also externally, including possible customers. The knowledge of all these people should be utilised within design thinking. Such multi-disciplinary teams need guidance to improve the results of the cooperation.

One way to support cooperation is to use face-to-face meetings [15]. This can be done in the form of workshops, where a limited amount of people come together and work on solutions for complex problems [17, 18, 59]. The number of participants should be limited to support cooperation and allow everyone to contribute.

To further improve cooperation in workshops, guiding methods can be used. The workshop must have an achievable goal, and the method must support the participants in reaching it. Here, it is essential that the method is not restrictive and facilitates flexibility and creativity among the participants.

During design thinking projects, three significant factors must be considered [60, 14]. These are the human needs, the available resources, and the business constraints. The first includes really understanding human needs and innovative ways how to satisfy users. Resources in this context not only mean in the context of one organisation but also include which solutions are feasible within today's technology. Last but not least, the solution of the design thinking project must be sustainable and beneficial for one or more organisations. Innovative solutions can be found if these three major influence factors are

considered.

Design thinking is considered a well-suited tool for designing useful artefacts [9, 61] and includes a mindset, a process and a toolbox for designers [9, 57, 61]. All three are essential in the context of design thinking, and this thesis improves the toolbox by enhancing the digital twin of design thinking. Even though the process and the mindset are not the immediate focus of this thesis, they will be influenced.

To support people in using design thinking, regardless of whether they are sophisticated designers or not, the following principles can be used to achieve the right mindset and better solutions [9, 61, 62, 14, 60, 63]:

- **User-centric:** One central principle is to apply a user-centric view, meaning that the design of the solution must focus on its users and what they need.
- **Work iterative and interactive:** Often, an iterative and interactive procedure is needed, meaning that the proposed solution should gradually evolve, step by step until it is finished. In this evaluation, prototypes should be used to gather feedback from stakeholders, including the users.
- **Teamwork:** A multi-disciplinary team must work together and communicate with each other.
- **Early Tangible Prototypes:** Tangible prototypes should be created regularly and early to foster understanding. These do not have to be physical but must be comprehensible to the involved stakeholders. They can be virtualisations like diagrams, physical representations created in the workshop, or, in later stages, more sophisticated prototypes. Such prototypes can be used to evaluate the created solution idea.
- **Exploration and Creativity:** The designers should explore multiple alternatives before deciding on a specific solution which will be implemented. To find these alternatives, the designers should not restrict themselves but think freely and outside the box.
- **Critical Thinking:** Assumptions and solutions should be critically scrutinised to improve the overall quality of solutions.
- **Solution Oriented:** Even though understanding the problem is important, the focus should be on finding a solution and losing oneself in the problem.
- **Flexibility:** The designers must be flexible in their thinking and applying their process and tools, which should always be adapted to the current design and not vice versa.

Through focusing on a user-centric way of thinking, empathy can be triggered, leading to an increase in the intrinsic motivation of the designer and better results in the long run [63]. This includes their inner attitude, meaning they must be open to challenging

2. Fundamentals

their assumptions about how the world works. Design thinking must be applied in a context which allows learning of the designers, which includes ways that allow them to understand. Users must also possess the right tools to enable the actual utilisation of this intrinsic motivation.

The process view to design thinking is based on a general procedure guiding users during their design [61]. They usually contain essential tasks that must be accomplished and how they relate to each other. Section 2.1.1 will provide more information about example processes.

If someone talks about applying design thinking, for example, within organisations, they often refer to what is called a designer's toolbox [62]. Under a toolbox in this context, one can understand a set of approaches or methods practitioners know and have at their disposal. Based on the specific case and their preferences, they use different tools to facilitate different phases of their design process [14, 60]. These tools have their own goals and procedures but can be used together to solve complex problems. Examples of such tools are introduced in section 2.1.2.

Tools used in design thinking are visual and support the internal processes of designers and their communication with other stakeholders [60]. Such tools help to make abstract ideas more tangible so that they can be discussed and put in a shape that can later be used as a foundation for the implementation.

Design thinking uses visual representations, created with the tools, to understand the problem and a possible solution [61, 14]. These representations will evolve as the designers increase their understanding and improve the solution. The visual representation can be a diagrammatic depiction [61] or physical artefacts [9]. It is essential that it can be easily comprehended and support communication between the stakeholders involved in the design project.

To counteract possible later confusion, especially when we talk about conceptual modelling, in the remainder of this thesis, such approaches, tools or methods will be called *design thinking techniques* or *techniques*. In section 2.1.2, examples of such techniques are introduced.

When using design thinking, the physical space is also important [14]. Physical meetings are important in design thinking and should be held in places that support creativity. Here, enough space must be available, the layout must be flexible and adaptable and enough resources to create physical visualisations must be prepared. Physical meetings are crucial at the beginning of projects to strengthen teamwork [63].

2.1.1. Design Thinking Processes

There is no one right way to apply design thinking. This section will introduce three examples of established design thinking processes to provide background on how design thinking can be used. Each process has advantages and drawbacks, and the users must choose the best one for their problem [60]. However, pre-defined processes can not support all possibilities and restrict the designers. However, they can help by making the design process more approachable to a wider variety of people. This will improve the design results, especially for novices.

The different processes vary in their concrete shape but have some common characteristics. They contain a set of tasks and a basic order. These are then used iteratively to improve the solution step by step. The order itself should be adapted to concrete circumstances, if necessary.

2.1.1.1. Three Spaces of Design

The first design guide discussed contains the three design spaces, as introduced by Brown [9]. The author uses spaces instead of steps to emphasise the non-linearity in the process. A space consists of a set of activities which belong together. The activities in these spaces do not have to be done in a particular order, but their order should be tailored to the concrete case. Each space can be used multiple times in the same design project, leading to an iterative approach. All three of the spaces are needed for a successful design.

The three phases are called: *Inspiration*, *Ideation* and *Implementation*. The first focuses on the design project's motivation and understanding of the problem and the stakeholders involved. The designers must emphasise with the future users of the solution and understand the core of their problem.

In the *ideation* space, the ideas are generated, prototyped, developed and tested. The activities in this space will be applied iteratively until a feasible solution is designed. New insights during the ideation can lead to going to the *inspiration* space and gathering new insights from potential users. During the ideation, multiple solution candidates will be created and explored.

Last but not least, the *implementation* space focuses on how a created solution from the first two spaces can be brought to the market and, therefore, how the new solution can be offered to potential customers.

2.1.1.2. Six Phases of the Design Thinking Process

The following design thinking process has six phases and was created by the Hasso-Plattner-Institut d-school in Potsdam [64]. These phases should be used non-linear and iteratively to tailor them better to the concrete case.

The six phases are [65]:

- Understand
- Observe
- Define Point of View
- Ideate
- Prototype
- Test

2. Fundamentals

The process starts with a problem or challenge that should be overcome by using design thinking. Then, the *understand* phase focuses on understanding the concrete problem by collecting relevant data and assumptions about the challenge in the context of the design project. If available, this will be enhanced by existing data about the users or similar projects.

In the *observe* phase, the designers talk to and analyse people involved in the challenge and possible solutions to gather a thorough insight into the problem. This can be done through interviews or observing people naturally interacting with the problem. This phase includes the search for existing data, such as available best practices to solve similar problems.

The next phase is called *define point of view*. The designers synthesise the information to gather a concise understanding of the user groups and possible solutions. For example, this phase uses techniques like personas to make the motivation for the solution more tangible and support empathy for the users within the design project.

In the *ideate* phase, multiple ideas for solving the challenge are created using creative methods, like brainstorming. The focus is creating many possible solutions for the problems defined in the previous phases. Afterwards, the ideas are summarised, discussed and aligned with the previously identified problems and user groups.

The next step is to create *prototypes* for promising ideas. First, they are low-level prototypes that focus on making ideas more tangible. They often have a physical form but are quickly created with cheap materials, like cupboards or plastic. Also, non-tangible prototypes, like videos, can be created. In this phase, different prototypes for different ideas are created. The goal is to gather insights from the creation of the prototypes and to use them to communicate the ideas with other stakeholders.

Last but not least, in the *test* phase, the prototypes are evaluated within the design team and with potential users. The tests gather essential feedback and allow a comparison between ideas so that the designers know where to emphasise in the subsequent iterations.

2.1.1.3. The Double Diamond Model

The last example process is the double diamond model of the Design Council [1]. This model's name is based on its visualisation, which can be seen in Figure 2.1.

The model consists of four phases [1]:

- Discover
- Define
- Develop
- Deliver

In the *discover* phase, the problem and its context are explored by talking to involved people and understanding their situation. The second phase consolidates the findings and creates concrete defined problems. Afterwards, the *develop* phase creates different ideas and prototypes which can help the before-defined problems. Last but not least, the

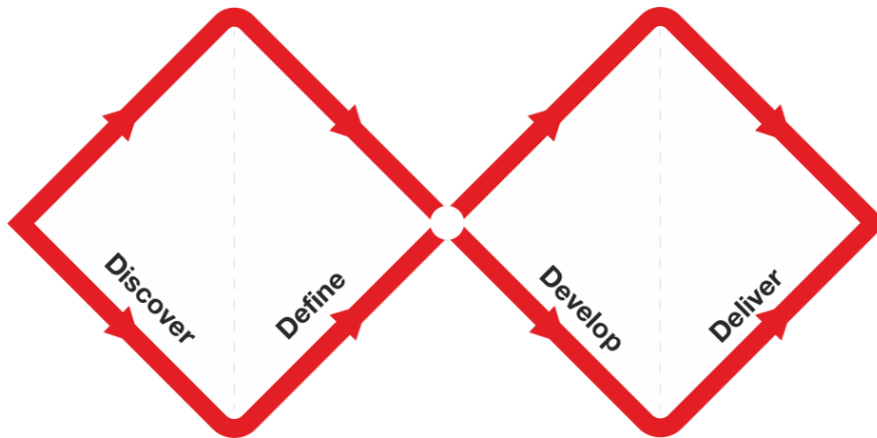


Figure 2.1.: Visualisation of the Double Diamond Model from the Design Council [1]
(published under the *CC BY 4.0 license*)

different ideas and prototypes created are tested and evaluated so that a set of chosen ones remains. These ideas should be used further to design and create the solution.

The double diamond shape comes from the amount of possibilities defined in the phase. Multiple possibilities are analysed and created in the first and third phases to broaden their horizon. In the second and fourth phases, the available possibilities are analysed and consolidated into a few promising ones that can be further used.

2.1.2. Tangible Design Thinking Techniques

To better understand what toolbox means in the context of design thinking, we now introduce three selected design thinking techniques that can be used in physical workshops. Numerous techniques exist with their benefits, drawbacks, and domains where they should be applied. Therefore, the best or most widely used method does not exist. The three techniques introduced below were chosen to introduce how such techniques can look.

The *Business Model Canvas* method was used as it is a well-known and established method in the context of innovation. The *Service Blue Print* method was chosen as a well-established but still used method in service design, which is also described in scientific literature. Additionally, the template used in *Service Blue Print* is not as strict as that for the *Business Model Canvas*. The *SAP ScenesTM* storyboarding method was chosen as it is focused on semantic rich paper figures and not sticky notes. This was also chosen as the technique for the prototype.

2.1.2.1. Business Model Canvas

The *Business Model Canvas (BMC)* was created to support people in designing innovative business models [66]. This is done by providing nine key building blocks that must be considered when designing a business model. The users of the BMC must already have a business idea, and the canvas supports the design by providing a guideline in the form

2. Fundamentals

of the nine building blocks. These must be filled with concepts from the domain of the business idea. The Business Model Canvas approach can be applied by start-ups as well as established companies.

To improve the usability of this method, the creators of the Business Model Canvas created a visual template [66]. This template can be used in workshops in printed form. In this case, the physical template is positioned in front of the workshop participants, for example, by attaching it to a wall. Then, the participants can discuss their ideas in the structure of the BMC, write their identified concepts on sticky notes, and position them to the corresponding building block.

The building blocks for the the BMC as introduced by [66] are the following:

- **Customer Segment:** Customers are a key factor in each business model, as they consume the products and provide revenue [66]. This building block builds customer segments by grouping customers after common characteristics. The building block should contain all the possible segments from which those are chosen, which the business model should serve.
- **Value Propositions:** This building block collects the add-value of why the business model created in the workshop is preferred to other ones [66]. This value is based on the products and services offered to customer segments.
- **Channels:** The channel building block contains how the company interacts with the customer segments and provides the value proposition [66]. Different channels can be utilised, and all should be collected in this building block.
- **Customer Relationships:** This building block contains the different types of relationships cultivated with the customer segments [66]. Such types can include communities, self-service, personal assistance, and similar. The different relations can be further characterised by adding their motivation.
- **Revenue Streams:** The revenue stream building block contains ways a company can create revenue from their customer segments [66]. Revenue streams can be built on various models, such as subscription fees, sales, or renting. An important characteristic of a stream is whether it is a one-time or recurring payment.
- **Key Resources:** This building block contains the essential resources the company must possess to create the value proposition [66]. These can be physical, like machines or immaterial, like patents. Human resources also belong in this building block.
- **Key Activities:** The key activity building block contains the essential activities which must be executed in the company so that the value proposition can be generated [66].
- **Key Partnerships:** This building block contains the important organisations the company is involved with [66]. This partnership can possess different types of competition, buyer-supplier, strategic alliances, and similar.

- **Cost Structure:** The cost structure building block contains the important costs while running the business model [66]. They are often related to key activities, key resources or key relationships.

2.1.2.2. Service Blueprint

Service blueprinting has been around since about 1990 [67, 68]. Its goal is to improve services by understanding their internal and external structure. This knowledge is visualised in a blueprint to be easily understood. Originally, this technique was not used in workshops, but in the meanwhile, people started to use them in workshops to analyse existing or ideate new services [69].

The service blueprint method provides the concepts and categories to consider if a service is analysed or designed. However, as it is common with creative methods, variations exist on how to create and what to visualise in a service blueprint.

For this thesis, the similarities of [70, 71, 72, 73, 74, 75, 68] will be used to abstract a common structure for what should be considered in a service blueprint. This should not be proposed as the best or new version but as a showcase of how knowledge can be structured in a workshop using a prepared method with a template.

The most important parts of a service blueprint are the different categories used for structuring it and aggregating activities and evidence. The template categories are represented as rectangles, where the corresponding concepts are written into them. The following three categories are used: *swimlane*, *evidence* and *time*. The last one is often mentioned as an additional category, which is optional and is not used in every instantiation of the service blueprint. The other two are used and described in every one of the listed sources.

Evidence comprehend all the physical evidence given to or used by the customers. For example, if an e-mail is sent to them or they receive a printed ticket. Keeping track of this evidence is important because it directly contacts the customer and influences their satisfaction.

Swimlane is an umbrella term for a set of similar concepts. Their unifying trait is that they contain the activities which customers or providers execute. The different swimlane types are listed below in their usually applied order:

- *Customer actions:* are activities which the user must perform to utilise the service.
- *Frontstage actions:* contain all activities in which the provider directly interacts with the customer. It does not matter if a human or a digital agent does this.
- *Backstage actions:* are those activities which must be performed to offer the service to customers but are invisible to them. Providers have to ensure that these activities can be performed, but the customers have no insight into what is happening.
- *Processes:* are the support processes which must be performed to allow the backstage actions. They can be executed by the provider directly or by third parties.

2. Fundamentals

- *Individual*: is not a predefined swimlane but represents swimlanes which are added for specific services in specific cases. This is needed to tailor the swimlanes, and it should also be possible to separate them further [72].

The swimlanes are grouped to specify which activities a user can interact with, which are seen by the users, and which are the backstage activities. Visually, the groups are represented by separation lines. *Line of interaction* separates the customer's actions from the provider's activities. *Line of visibility* declares up to which point the user can see what is happening on the provider side. Lastly, *Line of internal interaction* indicates the differentiation between the backstage actions and their support process.

There can be two types of relations between activities. First, the control flow indicates the logical order of the activities. The *agreement* relation specifies that agreement between two objects must be reached.

The *time* category stands orthogonal, usually vertical, to the other categories and provides a second axis, after which the evidences and activities can be grouped. It adds time spans to the activities of the overall service to allow a better analysis.

2.1.2.3. Storyboarding

In design thinking, stories can be used to understand, analyse and communicate insights from users [76, 60]. Stories from (potential) users can help the designers to better understand the users' pain points or communicate solutions. The telling of the story can help to better understand and communicate and find logical inconsistency within the story and, therefore, the proposed solution or the described problem.

Stories can be expressed not only through narration or text but also through storyboards. Storyboards visualise the key moments of a story graphically, which supports a more straightforward interpretation by humans [77]. The key moments can be hand-drawn or digital pictures and are set in a sequence, usually on a time axis. Storyboards in design thinking can be used to understand the situation, capture the knowledge of a design, and evaluate a design early on by showing its storyboard to users and walking them through it.

Storyboards do not always have to be drawn by hand or digitally but can also be created with physical objects. This was done by *SAP AppHaus*, which created SAP ScenesTM as a physical storyboard approach [7]. Here, pre-defined paper figures are used to create the storyboard physically on a table. The key moments are created in individual scenes. If they are put together, they create a storyboard. Using pre-defined paper figures restricts the possibilities but supports the participation of the people, as people without sophisticated drawing abilities can also participate. The figures provided by SAP are grouped into categories based on common features of the figures, e.g., characters or devices.

The pre-defined categories are [7]:

- Arrows
- Office Furniture

- Buildings
- Backgrounds
- Call-Outs
- Devices
- Signs
- Transportation Element
- Characters

For some of the paper figures, additional attributes should be defined [7]. The objects of the categories devices, signs and call-outs are also used to write text on them. The character should be able to hold emotion, and some additional information, like their roles, should be added.

2.2. Digital Twin

This section will provide a short background about the concept of digital twins, as this thesis takes the underlying premise and creates an environment which enables what we call digital twins from design thinking. How the concept is applied will be discussed later in this thesis, and the fundamentals of design thinking will be introduced here.

To improve the utilisation of physical machines and objects, a popular approach in the context of Industry 4.0 and IoT is to use digital twins [78]. Even though there is no consensus about a definition of the digital twin concept [79], a general characteristic is that two parts are linked together: a physical twin and a digital twin. The first is something physical, e.g., machines in a production line, and the other is a digital representation of this physical part. These two parts work together to improve their overall usefulness.

A digital twin consists not only of collected data but also of how they are structured and what algorithms and operations are executed on them [79, 80].

Different classifications and types of digital twins exist within the literature [78, 79, 81]. Even though not exclusively, digital twins are often mentioned in production, which was also the domain where the concept was first mentioned in 2002 in product lifecycle management. Later, it started to be used more frequently around the 2010s, where it is often associated with the production domain [78, 81], and also with the controlling of cyber-physical systems [80].

In the production domain, a digital twin represents a physical system over the time of its life cycle [78, 81], meaning that it is not only a static representation, but the digital twin and the physical twin are connected to each other and change together. Therefore, changes applied to one part will be translated to the other, establishing a bidirectional relation between them, leading to a real-time data exchange.

In this context, digital twins offer advantages for the operation of systems, as they provide a digital way to interact with physical systems and to record data about the

2. Fundamentals

system [81, 80, 78]. This data can then be used not only for real-time adaptation of the system but to simulate and analyse the real system or in the form of a prototype to predict future behaviour and, in this way, support decision-making. However, this presupposes that the physical environment can send and receive information, as it is done by using embedded systems within production environments.

Kritzinger et al. [79] also differentiate between the following three categories of digital representations based on their interaction between the digital and physical twin in the production domain:

- Digital Model: A digital representation of a physical system can be used for planning or executing operations. Here, humans do all the interaction between the digital and the physical parts manually.
- Digital Shadow: Digital shadows are based on digital models, but here, automated information exchange from the physical to the digital twin exists. Therefore, if the physical system changes, so does the digital system.
- Digital Twin: Based on digital shadows, a digital twin then possesses information exchange between the physical and digital in both directions. Therefore, changes in the one lead to changes in the other.

Digital twins can be representations of individual physical twins and establish an instance-based digital twin with a one-to-one relation. This notion can also be applied to create digital twins of systems, which consist of models, contextual data, and services connecting it with the represented system [80]. Therefore, not only digital twins of individual physical objects are possible, but also of systems.

Therefore, they can consist of different components, connected to each other and can even have a hierarchical relation to each other [81]. Having these different components working together supports cooperation between experts, as they can interact with the familiar components, and the links between them can be used to communicate with other experts.

Additionally, digital twins can be used in the design by creating the digital twin as a prototype, which can be used for testing and evaluation [78]. Having a digital twin as a prototype helps to capture the concepts in people's minds and make them storable and processable [78]. With digital presentation, people can work better than with models only in their minds. The digital representation also reduces the need for physical components, which usually leads to more overhead and costs.

Digital twins can not only be applied to represent technical objects. For example, the digital twin concept can also be applied to the healthcare domain [82] to create a digital representation of patients and improve their internal processes. Another domain outside production, where digital twins are used, is farming [83]. In this context, digital twins are used to gather real-time data and provide information about the current status or deviations to farmers so that they can act accordingly.

The topic of a digital twin of organisation has also gained more momentum recently [84]. Such digital twins enhance the idea of representing only physical objects towards

representing whole organisations [85]. Here, data from sensors can be used, and information from different sources can be combined to gain insight into the organisation and provide suggestions for future actions. Digital twins, in such cases, must not necessarily directly influence the physical twin, as in some cases, it is important to include humans in the decision-making [84]. Therefore, a human must trigger or confirm the actions that should be executed.

Independently of which kind of digital twin is used, an environment is needed in which the digital twin can be created [78]. Such an environment must interact with the physical twin, hold the gathered data in a structured manner and execute the needed operation upon the collected data.

Further, to support the representation and the offering of the additionally created information, the digital twin must evolve alongside the system it represents [81]. Therefore, the digital twin must adapt, and this is not only the representation of the system's state, but the structure of the digital twin itself must be adaptable to allow further usage, even if the system itself changes.

Even though digital twins started and are often mentioned in the context of production, the basic concept is also often used in other domains to generate benefits for them. For this thesis, we will apply the concept of digital twins to design thinking, to support workshops and the created results.

2.2.1. Repurposing Digital Twins for Design Thinking

This section will discuss how the basic premise of digital twins can be repurposed to be used in the context of design thinking and its workshops. Therefore, digital twins in production and for systems are compared to using digital twins in the context of design thinking.

In Table 2.1 an overview of two definitions of digital twins is provided based on the information above. The right column then contains how the notion of digital twin should be repurposed for design thinking. It is not the idea that digital twins with all their characteristics are applied to design thinking but that the basic notion is mapped to generate benefits. This thesis focuses on how this digital twin from design thinking can be established, what it must offer, and how it can be used.

A digital twin must be a virtual representation of something that already exists or should be created in the future. The second column (which is called digital twin in production (DTiP) and is based on [81]) are, for example, robots that are used in factories. The third column (which is called the digital twin of systems (DToS) and is based on

⁻³<https://azure.microsoft.com/en-us/products/digital-twins> (accessed: 13-7-2023)

⁻²<https://cyberbotics.com/> (accessed: 13-7-2023)

⁻¹<https://azure.microsoft.com/en-us/products/digital-twins> (accessed: 13-7-2023)

⁰<https://cyberbotics.com/> (accessed: 13-7-2023)

¹<https://azure.microsoft.com/en-us/products/digital-twins> (accessed: 13-7-2023)

²<https://cyberbotics.com/> (accessed: 13-7-2023)

2. Fundamentals

Features \ Type	Digital Twin in Production[81]	Digital Twin of Systems[80]	Digital Twin from Design Thinking
Physical	physical objects (existing or future)	cyber-physical systems	designs of innovative scenarios using haptic objects
Digital	Data, software	models of the system, data from a system, services for using data and models	conceptual models, software
Relation between digital and physical	state of physical and digital twins are mirrored	data integration from the system and invoking services	enrichment to improve conceptual design
Operations	control planning and execution of physical systems	control and analyse systems	support design flexibility, fast prototyping, aggregate and enrich information for automation,
Examples for Software	e.g., Azure Digital Twins ¹ , Webots ²	e.g., MontiArc[80], ROS2[80]	e.g., Scene2Model, CuTiDe[25]

Table 2.1.: Overview of different kinds of digital twins and their features

[80]) is the cyber-physical systems which are connected, analysed and controlled by the digital twin.

For the digital twin from design thinking (*DTfDT*), the part that the digital twin should represent is the innovative idea created in the workshops (see section 2.1). In workshops, such ideas are created using tangible techniques that physically represent them. Of course, the physical representation does not capture all the information as it is captured in the participants' mental models. Additionally, the mental models can not be easily processed and shared. Here, the notion of digital twin should be applied to overcome these drawbacks. This is similar to what is achieved with digital twins as prototypes, as introduced above.

To establish a digital twin, the digital components are necessary. This includes software containing the logic of the digital twin, structured and computer-processable data, and ways of interacting with it. The *DTfDT* will be based on diagrammatic, conceptual models, as they can help to structure and represent knowledge and are used to describe mental models explicitly (see section 2.3 for more information). The modelling tool is the software that allows the interaction with and processing of the digital twin.

The physical and digital twins are connected in the *DTfDT* by creating a digital model out of the physical representation created in the workshop. This is then a replica of the visual representation, which can be enriched with information, and mechanisms can be applied to ease the work for the users.

A digital twin is used to represent information, process it, and execute operations on it. In *DTiP* and *DToS*, this is usually some interaction with the physical component to control it or to use the data to analyse to make implicit knowledge more explicit. In *DTfDT*, the processing of the models should support fast prototyping in capturing the mental model of the participants, which can be captured and shared with other stakeholders for evaluation. However, the *DTfDT* captures the knowledge and makes it processable, supporting workshop participants by providing additional information even during their work.

The digital twin also increases the flexibility as it can be continuously changed without physical components. Here, it means that the models can change, and the underlying modelling method can be adapted to meet the user's needs. Last but not least, a workshop's novel idea may need different models to capture all the information, which can be aggregated in the digital twin to get a more complete representation.

2.2.2. Tools for Creating and Using Digital Representations in the Context of Design Thinking

One important aspect of digital twins is that digital environments are needed where they can be established. This section will introduce different tools already used to create and manage digital representations in the context of design thinking or workshops. This should provide input to existing solutions and explain their benefits or drawbacks. Even though this thesis is focused on using conceptual digital models to structure digital twins in design thinking, the tool discussed in this section must not be based on conceptual models.

2. Fundamentals

This section introduces tools that can digitally represent the knowledge created in design thinking workshops. They can be used with tangible techniques in a workshop or be created purely digitally. The important aspect is that they are created with creative techniques to support the design of novel ideas. That the digital representation is created automatically from physical representation is optional here, as no such tool was found.

One tool which supports design thinking is *MELCA* [42]. It is a modelling tool which implements selected design thinking methods and allows them to relate their concepts. For example, methods like stakeholder maps, personas and journey mapping are implemented. Therefore, the modelling method was defined, and the needed concepts were offered in the modelling tool to create digital models with the tool. The different supported techniques can capture different perspectives of the idea.

Another tool to support design thinking through conceptual modelling is *CuTiDe* [25], which is an extension to the before mentioned MELCA tool (cf. [42]). The CuTiDe tool thereby supports more design thinking techniques and allows mechanisms to support the linking and transformation between different implemented techniques.

Both tools can help by offering basic concepts to visualise their ideas. The digital models can be manually created through the user interface, which enables their processing and eases the sharing but hinders the cooperation of multiple stakeholders. This is because the modelling tool offers an interface which allows only a single user to create a model. Further, the usable concepts are predefined by the creators of the modelling tool.

Another tool based on modelling tools is *AOAME4Scenes* [26], which is a tool that offers a design thinking technique implemented into the modelling tool. The difference from the tools before is that the tool supports an easy adaptation of the available concepts to fit the user's needs better. Through this, an on-the-fly adaptation is possible. Additionally, the models can define and evaluate semantic constraints based on the underlying ontology structure. However, the tool also has an interface over which specific models can only be adapted individually, hindering workshop cooperation.

FlexiSketch [19] is a tool which combines creating sketches on tablets or other computer devices and then allows to define types and cardinalities between the drawn elements. The authors did this to add lightweight metamodeling capabilities. The created node-and-edge diagram can then be exported as a semi-formal model. Sketches can be created here, but no design thinking techniques are supported.

There are not only model-based tools for supporting the creative creation of novel ideas but also tools like *Mural*³ or *Sketchboard*⁴. These are collaborative whiteboard applications, meaning that the users can draw together over the internet on digital whiteboards to create a common visual representation of their idea by drawing it in the browser.

This possesses the benefit of a location-independent collaboration and a digital representation. However, the digital results are less machine-processable than conceptual models and do not support physical working.

With the *Tele-Board* [44] tool, the authors created software that allows the use of digital

³<https://www.mural.co/>, visited: 24.08.2023

⁴<https://sketchboard.io/>, visited 24.08.2023

whiteboards collaboratively, locally, and remotely. Therefore, the participants in different locations are part of a video stream shared with all locations. Here, digital whiteboards or other digital devices capture and share the information with the participants in different locations. In this way, they can collaborate by manipulating the same artefacts and seeing each other expressions through the video stream.

To summarise, this was an excerpt of available tools to support design thinking and creating novel ideas through creative techniques, focusing on those using conceptual modelling. There exist tools which focus on creating digital, conceptual models out of the result of design thinking workshops. Others focus on the collaborative creation of digital diagrams of remote teams, and some tools want to recreate the flexibility of haptic techniques with digital measures by using digital whiteboards or touch screens (e.g., tablets). However, no tool was found that tried efficiently to use both a haptic and a digital representation, with support for combining them. This gap is one of the main motivations of this doctoral thesis.

2.3. Conceptual Modelling

This section introduces a background to conceptual modelling, which is used as the core for the design thinking digital twin. Therefore, the digital representation of the innovative idea created in a workshop will be captured as a diagrammatic conceptual model, making it human and machine-interpretable. As there is no globally accepted definition of conceptual modelling, this section will introduce the topics of modelling, conceptual modelling and metamodeling to understand the background of this thesis.

This section will start with an introduction to modelling, followed by diagrams as a representation and modelling methods to improve the comprehensibility of the models. Finally, metamodeling will be introduced as a method to define modelling methods.

According to Stachowiak [86], model is an umbrella term for many, often very different, kinds of representation. If one wants to use it as an all-encompassing term, it must be on a high abstraction level, limiting its meaning. However, [86] also defines some main features of a general model term. The first is that models always represent something natural, artificial, or even a model itself. It can be something that exists or some to-be system. The next feature is that models do not contain all characteristics or attributes of the original, but only those which are useful for the creator of a model. Models are not only created as a representation of something but as a representation of someone, be it a human or a machine. The last property determines that a model has to have some usefulness, at least for a specific period. This use guides the creation of the model and which characteristics of the original system are included.

In their book, Thalheim and Nissen [87] define models as a reduced image of an item or process, which can be used as a proxy to interact with it. The models themselves serve a purpose and fulfil a specific task. Thereby, they use a dedicated language to represent the model. The purpose of the model thereby directly influences the chosen abstraction level of the models. This hinders or prohibits the use of models outside their defined purpose. According to [87], some typical purposes of models are gaining insight,

2. Fundamentals

supporting the understanding of the original, planning for creating a to-be original or verifying hypotheses.

Henderson-Sellers et al. [88] describe models as a simplified representation of (parts of) the real world, called a system under study. The simplification is reached by applying abstraction to the identified portion of the real world, which is needed to focus only on the important aspects of the problem domain, which are then solvable. Solutions found in the abstraction can then be transmitted to the real problem domain if applied correctly.

Mayr and Thalheim [21] stress the fact that for many models, it is not enough to have just a mental model (which exists in the head of the creator), but that a created model must be shared with stakeholders. Hence, defining an understandable representation, like a diagram, natural language, or an XML file, is important. It is only important that the needed characteristics can be represented and the model can be interpreted by corresponding agents (e.g., other humans or machines) [21].

One purpose of a model is that it can be used instead of the real world, and thereby, it is important that one can drive the same information or can come to the same conclusion, regardless of whether the model or the real system is analysed [33]. This is only true for the purpose of the model as it cannot deliver the same results for every possible analysis because abstraction at its core means that information of the real system is reduced.

To summarise the above introduction to the term model, one key aspect of models is abstraction. In this way, the complexity of the system under study can be reduced, and the focus is set on the needed characteristics, which are integrated into the model. The model's purpose defines this subset of characteristics, and for different purposes also, the subset of characteristics can change. Models are created as an abstraction of something, physical or artificial. Further, it does not matter if it is created of something already existing or planned. Therefore, models cannot only be used to analyse something existing but also to design something.

One kind of model often used in computer science is called conceptual model, where there also does not exist a generally used definition of what precisely a conceptual model is [21]. According to Mayr and Thalheim [21], conceptual models are often created to design a system for later realisation in the real world. In computer science, this realisation usually involves software, databases or workflows. Conceptual models are used to represent concrete objects, their relations and their characteristics. With this model, we do not talk about a concrete instance but the concept it represents, hence the name conceptual modelling. An artificial language is usually used to represent the created model, offering terminology and some rules. The meaning of the terms in the models comes from mapping the represented concept to a concept space, which is defined through community members. However, models do not only support communication through the representation of their language but by defining a namespace and terminology for the analysed domain [89]. These defined terms also support communication between different stakeholders.

Guarino et al. [90] argue that conceptual models are models of how agents perceive a domain. The conceptual models are, therefore, a possible representation of the mental image of a domain, which is created within the mind of a stakeholder/modeller. Therefore, they do not capture a domain directly but what an agent perceives of the domain. In

this context, conceptual modelling creates theories, methods and tools to build and process models. These models should be created to help design and create information systems, enterprises, business processes, databases and similar. Conceptual models thereby represent concepts and their relations. However, the goal is to represent a domain and make this representation meaningful and intuitive to understand.

For [90], conceptual models convey conceptual semantics based on concepts humans can interpret. Therefore, models without such conceptual semantics should not be considered conceptual models, according to [90]. Represented data alone is not considered a conceptual model; only if the attached concepts hold the data can it be considered one. In addition, when creating a conceptual model, one defines a set of concepts and relations that exist in the domain [90]. The model can then be created with these defined concepts, which create an ontological view of the domain.

Based on this view, a language for creating conceptual models can be defined. This language holds the conceptualisation of the domain from the creator and can be used with the same conceptualisation in mind. A language user can have another conceptualisation, leading to different interpretations of the same model. Therefore, constraints on the conceptualisation should be introduced so that misinterpretations are minimised [90]. Such a definition is even more critical if conceptual models should be shared between different stakeholders.

A *concept* is an abstract idea that generalises defining characteristics of observed instances in one's mind [91, 21]. Concepts are mental representations containing the parts relevant for the modeller and help to perceive the real world [90]. Therefore, the models are conceived in one's head, represent real objects, and help identify the essential aspects and what can be abstracted. Objects can be tangible things, systems, humans or anything. If conceptual models are used in a community, the different participants must understand the concepts similarly.

[27] defines the activity of conceptual modelling as creating a formal description of some aspects of a system under study, and a conceptual model is then the formal representation of the description. Those support the understanding and communication of the system under study. The formal representation also helps to capture the semantics of the domain [27]

Simulation is another field where conceptual modelling is used. Robinson et al. [92] introduce and discuss different interpretations of conceptual modelling within the field of simulation, as in this field also, a clear consensus has yet to emerge. One difference to conceptual modelling in computer science is that the models' goal in the simulation context is restricted to facilitating simulation [92].

The conceptual model is often created before the actual simulation model to improve the quality of the simulation and the results [92]. The conceptual model is used for documenting and formalising the system, which should be simulated. Therefore, a special language can be used to support communication between different stakeholders, which is one of the main benefits and improves the overall quality [92]. Additionally, the conceptual models can be used for an earlier evaluation, improving the trust in the simulation.

[92] argues further that it is impossible to create a simulation without conceptual

2. Fundamentals

modelling because this model is always created in the users' mind [92, 90]. Therefore, the only question that must be answered is if the model is documented, which is needed to share the mental model with other stakeholders. Using a formal way to describe the model enables effective sharing and improves the overall results, as everyone has a similar understanding.

However, [92] does not highlight the representation of objects and their concepts but uses conceptual modelling more as a documentation of the domain, which can include the concepts of objects. This is different from [21], where the represented concept has a more central position in the definition of conceptual modelling.

Another application area of conceptual modelling is enterprise modelling, which uses models to design, improve and analyse enterprises.

In [93, 23], enterprise modelling is introduced as a (computational) representation of an enterprise. Represented are, for example, the organisation (structure), behaviours, processes, resources, human actors or constraints. Such models are used to analyse, design, control or reason about various aspects of enterprises. Therefore, it is not clear whether humans or machines use the models. Enterprise modelling thereby externalises the knowledge of an enterprise within so-called enterprise models.

To convey such information, it is essential that the representation is based on understandable concepts (semantics) and that the same terms reference the same concepts in all parts of an enterprise [93]. This is especially important for automated processing because machines do not easily possess the capability to understand different terms with the same meaning. The increased need for agility in enterprises leads to a need for better cooperation between different parts of the enterprise, which is a complex task that can be supported through enterprise models.

Enterprise Modelling (EM) was formed in the eighties based on the three main streams (cf. [23]):

- Business analysts who had to describe and analyse the current version of an enterprise and future solutions.
- To create complex enterprise software and information systems, structured approaches for capturing and analysing business processes were needed.
- The need for integrated product design and manufacturing systems and to combine them with other systems within an organisation called for approaches and methods to support such complex undertakings.

An enterprise model does not contain all the information in one representation but separates them in different *modelling views* [23]. Therefore, individual models are created and integrated. Together, they present a more holistic view of the enterprise.

To maintain their value, models must evolve alongside the current developments of the system under study, which is not only the model itself but also the used concepts. EM is on the domain where it is important to enable the modelling languages to evolve with what they should represent [23]. One example of such an evolving field is Industry 4.0.

To fully utilise models beyond representing a mental model, digital tools are necessary [23, 93, 90, 20, 94]. These tools allow us to easily share and adapt the models and analyse and process them, creating additional information and value. For example, query the right information for big enterprise models, apply reasoning, and make implicit knowledge explicit.

Till now, we have discussed different aspects of conceptual modelling and two examples, being Enterprise Modelling and conceptual modelling for simulation. Models are abstractions of a system to fulfil a specific goal. However, to make the models useable outside one modeller's head and to make them shareable with other humans and machines, they must be encoded understandably [21, 23, 90]. Therefore, the available concepts and how they can represent knowledge must be defined. We call such definitions modelling languages, which will be discussed in more detail later (see section 2.3.2). The created models then have a representation, for example, through diagrams, which are visualisations. Diagrams will be introduced below (see section 2.3.1).

In computer science, we often use semi-formal diagrammatic modelling languages for representing conceptual models [21, 92, 23]. Diagrammatic means they have a graphical representation, shapes, labels, lines and similar. However, it is essential to remember that the diagram is only the representation, and the model itself is more than just the representation. Semi-formal in this context means that the semantics of some aspects of the language are defined, and others are encoded informally, e.g., through labels [95].

The following section provides a background to diagrams and introduces modelling languages in more detail.

2.3.1. Diagrams

Diagrams can help to better represent information than textual structures [96, 97]. This is because they allow for a particular encoding of information, which can show implicit knowledge explicitly. Diagrams are often applied during problem-solving tasks, where an internal representation (in the brain) is translated to an external representation (information represented on some surface, whiteboard, paper or computer screen). The diagrammatic representation can explicitly store topological and geometric relations within the representation of essential parts of the problem space.

More generally, different possibilities exist to represent a problem during the solution-finding process. For example, the problem can be described textual or diagrammatic, whereby the same problem can be represented simultaneously in different ways. According to Larking and Simon [96], such different representations are informationally equivalent if the same information can be derived from both. They are computationally equivalent if they are informationally equivalent and if the same information can be derived with (nearly) the same resources. [96]

Even if two representations, e.g., one in a diagram and one in natural language, are both informationally equivalent, how good they are understandable can differ. For example, geographical problems can be represented graphically, which is easily understandable by humans [96]. If, then, for the same problem, a natural language description is created,

2. Fundamentals

the same geographical information must be encoded in the text, which can be hard to do. Here, a diagram can be easier for users to understand.

Spatial information holds value because humans can infer additional information if it is diagrammatically represented in a two-dimensional space [96]. An example would be the search for local maxima on a graphical curve instead of a presentation of the different values in a table. Therefore, a specific representation in general is only useful if the processing unit (e.g., humans) can understand and process the underlying data structure. If nothing can be derived from the representation, it will be seen as useless and, therefore, will not be used. Depending on the capabilities of the processing unit more or less information may be found within the presentation. Information saved in a picture of a diagram (so no machine internal representation like a graph is available) is not as accessible to machines as to humans [96]. Computers need more complex algorithms to interpret such a representation (see section 2.4.2).

However, diagrams do not automatically increase the understanding of the represented knowledge [96]. The diagrams must take advantage of their spatial information encoding and put information together, which belongs together. So, these useful features must be applied thoughtfully and correctly to gain additional value.

2.3.2. Modelling Language and Methods

Based on the introduction to modelling and diagrams above, this section introduces the background of modelling language and methods. These define what can be represented in conceptual models to make them understandable to humans and machines. As with this thesis, we will focus on diagrammatic models for this section.

Mayr and Talheim [21] state that a language, independent of whether it is natural or artificial, is only useful for proper communication if certain prerequisites are met. The communication partners must be familiar with the grammar of the language. The concepts described in the language must have a similar meaning to all stakeholders, and the receiver must correctly understand the information sent.

A language (natural or artificial) contains terms that describe concepts/things and their properties and are used to create sentences for describing statements [31].

Harel and Rumpe [97] separate modelling language in two aspects: *Syntax* and *Semantic*. Whereby the visualisation of the language is contained within its syntax. In [97], syntax is the form in which information is stored. Other stakeholders then must interpret the created form to extract the encoded information and its meaning. This meaning is then called semantics, which can differ in the context in which the syntactical data is interpreted. Therefore, the same syntactical data can be interpreted differently based on the situational context or domain. Therefore, it does not matter if we talk about a natural or artificial language. If two agents (human or machine) want to communicate, they must first agree upon the language to transport and adequately understand the exchanged information.

Such a language with defined syntax and semantics is needed so that humans can interpret it and that digital modelling tools can understand and process the models [97]. Therefore, the definition of language is important, as computer-aided tools cannot

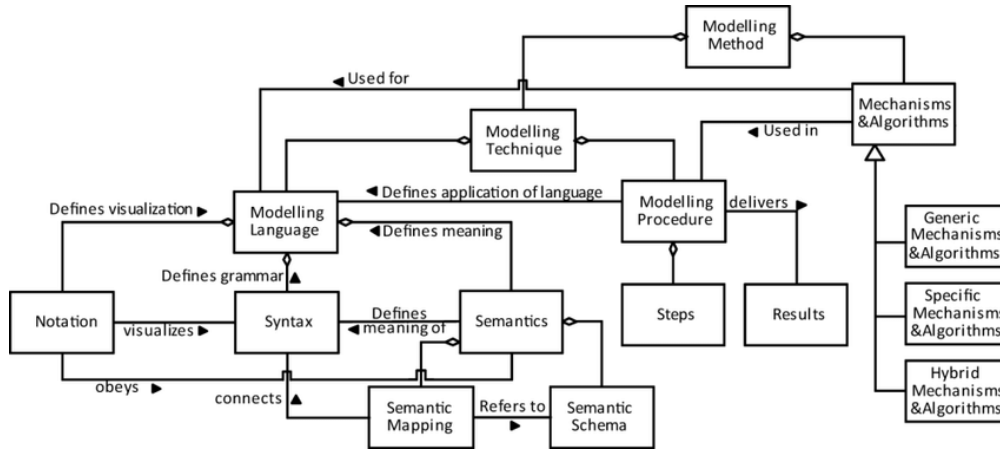


Figure 2.2.: Modelling method components form the metamodelling framework of [2, p.3])

interpret models if they deviate from syntax. Additionally, they cannot offer concepts to represent a model if their meaning is not defined.

Meaning can be defined through the specification of a semantic domain and a mapping from the syntactical terms to this domain [97]. The domain contains the set of all possible concepts that can be used, and the mapping relates such a domain to a term of the language. In this case, the meaning of a term in the language is given by relating an abstract concept from the semantic domain to it, and the term then invokes the linked concept when used.

Karagiannis and Kühn [2] separate a modelling language into three aspects: *Notation*, *Syntax* and *Semantics*. Syntax contains the concepts used to describe models and grammatical rules defining a model's appearance. Semantics then maps the used concepts, parts of the model or the model as a whole to its meaning, encompassing the domain to which it is applied. Last but not least, the notation contains the definition of the representation of the models. These will be graphics (e.g., pixel-based or vector-based) for diagrammatic models. However, not only static representations are meant here, but also dynamic aspects if the visualisation can highlight certain states within a model. The notation can either be static or change based on the information encoded in the model [2].

Further, Karagiannis and Kühn [2] discuss modelling languages and define modelling methods. The components of modelling methods according to [2] are visualised in Figure 2.2. A modelling method contains a modelling language, a *modelling procedure* and *mechanisms & algorithms*. The modelling procedure contains the steps which must be fulfilled to create a useful model with the defined language. Moreover, the mechanisms and algorithms are used to process or create models. For example, the processing can include evaluation, querying, analysing, simulation or transformation. Mechanisms & algorithms can also support the modelling procedure by creating parts of models out of data, user input or other sources. Another support could be to show the user where there are inconsistencies or mistakes in the created model (e.g. if the syntax rules are violated).

2. Fundamentals

In the research field of conceptual modelling, there exists a classification into a domain-specific and general-purpose modelling language, whereby the concrete separation between these two is not clearly defined [32]. Domain-specific modelling languages are those that integrate concepts/terms of a specific domain into the modelling language itself to allow for a more natural depiction of the domain [32, 29]. General-purpose languages can be used for various domains without adaptations. As examples for general-purpose languages, often standardised languages like Unified Modeling Language (UML) or Business Process Model and Notation (BPMN) are taken. As stated by Karagiannis et al. [32], it is often not clear where to position the separation line between general-purpose and domain-specific modelling languages because the definition of the domain can vary itself. For example, is software engineering a domain, or is the domain the context for which the software is created?

A need for directly including domain-specific concepts into the used language is known from modelling languages and software engineering, where many languages include domain concepts as first-class citizens [98]. In such domain languages, the concepts are available immediately and do not have to be created with more abstract concepts from the used language. Regardless, often general-purpose programming languages (e.g., Java and C#) are preferred because the well-designed languages are of little use if adequate supporting tools (e.g., compilers, debuggers and editors) are not or insufficiently available [98]. This means that such tools must not only exist but must also work well and support the user to generate additional value.

Another way to classify modelling languages is if they are standardised or not. With standardised here languages are meant that are well established and known to a large audience or which are defined in a published and standardised specification. An example, for the first, would be Entity-Relationship (ER) diagrams (cf. [99]), which are well known in the conceptual modelling community, and nearly everyone has an understanding of them. Even though there exist extensions and different variations to the ER diagrams (see [100] for an overview of different variations), many people in the domain of modelling or databases know this language.

For a group of published and standardised specifications, two well-known examples are UML (cf. [101]) and BPMN (cf. [102]). Both were standardised and, by the Object Management Group[®] (OMG), have detailed specification documents online and are well-known in the respective goal domains. On the other hand, the modelling languages and methods published in [103, 104] are not standardised but were created for a smaller and specific target group. They are used in their communities but are not necessarily well-established in the conceptual modelling community.

Standardised modelling languages benefit communication by providing a common language that can be learned and looked up by different stakeholders. Such modelling languages are often valid for a long time because adapting and republishing a standard is time-consuming. Having this longevity supports communication between a wide array of stakeholders. This widespread use often comes at the price of a higher abstraction level ([34]), which does not directly support the usage of concepts from the application domain [28, 105].

According to Karagiannis [28], even though standardisation has its benefits, it is not suitable for all domains and application scenarios. The modelling languages must often adapt to changing domains or requirements towards the modelling language, especially if domain concepts are used to increase the expressibility of the language [28, 105]. Such adaptable modelling methods are then usable by a smaller group of stakeholders but can improve their model value [28]. In such cases, the developed or chosen modelling method must adapt and evolve alongside the stakeholders to better meet their needs. One framework to allow such fast adaptation and support continuous development of methods is Agile Modeling Method Engineering (AMME), which was motivated and introduced in [28].

One example of adaptability in a modelling language is ArchiMate [106], a standardised language for enterprise architecture management. Even though it is standardised, it encompasses some adaptability as the notation can and should be adapted to specific stakeholders to help them better represent their knowledge. This does not mean there is no notation in the first place but that the visualisation is defined as changeable in the standard itself. In the language itself, the concepts and their representations are separated. However, all the concepts have a default visualisation, usually a rectangle with a symbol in the top-right corner [106] and a colour coding representing their belonging to the different aspects. The adaptation can be made by changing the symbol in the top-right corner of the symbol as a whole.

Another drawback of standardised modelling methods is that they often do not focus on operationalising the knowledge captured in the models [34]. Suppose processing of the modelled knowledge is available in modelling tools. In that case, they are often based on the modelling language in the specification or extensions to it but not defined in the specification itself. However, fully utilising the captured knowledge can increase the model value for the user. The framework introduced [2] (see also Figure 2.2) states that mechanisms and algorithms are essential enhancements to modelling languages. [2] further states that it is essential to support the creation of new modelling methods (including mechanisms and algorithms) to allow more tailored modelling methods for the user's needs.

Independent if a modelling method is general-purpose, domain-specific or standardised, it must be defined to be used and understood by other humans and machines [107]. This description must be unambiguous to clarify the meaning of the models created with the method. Computers, in particular, lack the ability to interpret the model and its context to make conclusions. Therefore, an exact definition is necessary.

There are two approaches which are widely used to define modelling methods [2, 21, 97]: *Metamodels* and *(Graph) Grammars*. It would also be possible to define such a method purely through natural language. However, this approach is omitted in this work because it is regarded as imprecise and depends too much on the interpretation of humans. Even if humans can often interpret models with such imprecise definitions, it can easily lead to misunderstandings.

According to Kleppe [107], a metamodel can only be used to define a language's abstract syntax, aka the internal structure. However, the concrete syntax, how the models are

2. Fundamentals

represented, and the semantics must be defined additionally. Whereby the metamodel often influences the concrete syntax. A concrete syntax is important, as this is the part used to communicate the models. For example, the notation must be defined and known by the different stakeholders for diagrammatic models. Otherwise, the modelled information cannot be exchanged.

One language can also have multiple concrete syntaxes. For example, BPMN can be represented as a diagram or in an XML format. These different possible concrete syntaxes call for the need to describe languages in an abstract syntax as a common representation [107]. Metamodelling is a potent way to describe an abstract syntax.

Especially if the modelling method should be able to evolve alongside its use, as proposed with AMME (cf. [28]), it is important to have a definition which is understandable and comprehensible both by the users and machines. This is utilised when using metamodelling platforms to create modelling tools [2]. Here, the defined metamodel is interpreted, and a tool is generated based on it. Therefore, if the modelling tool should change, the metamodel must be adapted, and the tool changes automatically.

The following subsection will go into more detail about metamodelling in the context of defining modelling methods.

2.3.3. Metamodelling

This section provides background to metamodelling and metamodels as a foundation for implementing modelling tools. Conceptual models are used as the core for the digital town from design thinking, where the models and the used modelling method must adapt to domain changes during its usage. Metamodelling will be used as a foundation for these changes.

The difference between the terms metamodelling and metamodel is that metamodelling describes the activity of creating modelling methods, and a metamodel is an artefact created in the course of it. A metamodel represents the core structure and the concepts available in a modelling method and its models. It is the core structure and description of a modelling method. Metamodelling contains the metamodel and additional constructs like how the language is used or ways to process the models [2, 31, 107]. Even though metamodel is one of the core artefacts created during metamodelling, the activity of metamodelling also includes the definition of additional constructs like consistency rules, algorithms, model creation procedures, and similar.

In [31], a metamodel in the context of informatics and business informatics is defined as a description model of a language which describes a model on a lower modelling layer. Because this language is used to create the models and the metamodel is an abstraction of the language, the metamodel is a model of the model and can be seen as a model on a higher abstraction level. However, it is not only a model of a model but also a description of the language that is used to describe a model. This means the purpose for applying the abstraction differs from creating the model on the lower level. This whole construct is not only restricted to two layers (model and metamodel layer) but can, in theory, be applied infinite, whereby the model on the higher abstraction is the metamodel of the model and the language one layer beneath [31].

2.3. Conceptual Modelling

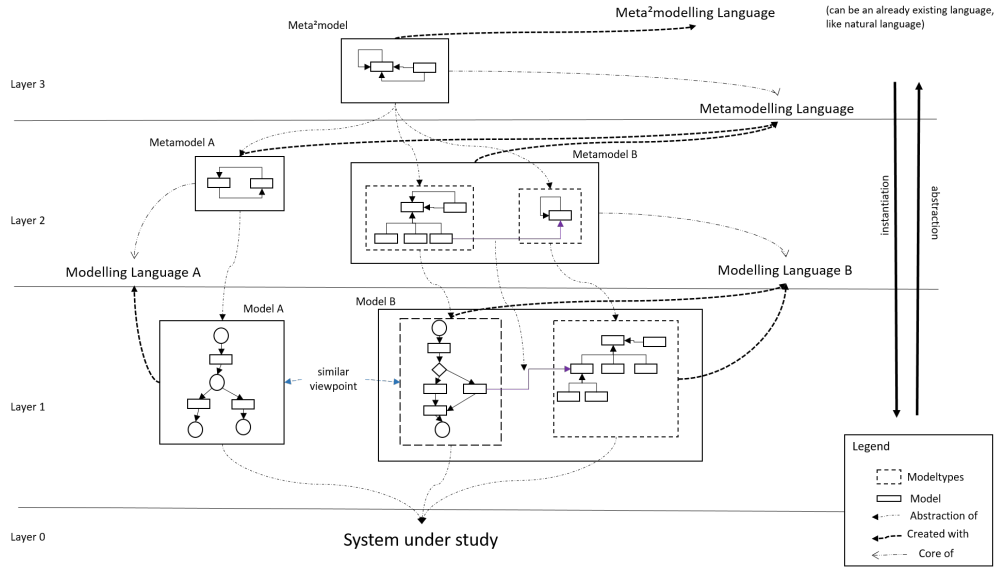


Figure 2.3.: Visualisation of a three-layered model hierarchies

Different modelling layers are visualised in Figure 2.3. Three modelling layers and the system under study (layer 0) are shown. In layer 1, the models of the system under study can be seen, and they are created within the defined language. Then, on layer 2, one can see the metamodel layer, which is a further abstraction of the models while considering the core aspects needed in the language. The models in the different layers are not real models but only placeholders. In Figure 2.3, more than three modelling layers could be used by creating a meta³-layer, which would be created in the same principle as the other model layers and, therefore, would not generate an added value for the presentation.

Another language is needed to describe the metamodel, often called the metamodelling language. Therefore, different languages are also aligned alongside the layers of the metamodel hierarchy, which also leads to a hierarchy of the modelling languages [31]. This can be seen in Figure 2.3, where three languages are shown on the right side of the figure: (i) modelling language (ii) metamodelling language (iii) meta²-modelling language.

However, defining a metamodel as just a model of a model is not sufficient [31]. The key aspect is that the metamodel is an abstraction of the model and not a different abstraction of the system under study. It is not enough that the system is described differently, like another focus, but on a higher abstraction [31]. Therefore, the modelling language should be considered in the abstraction, not just the domain.

A starting point must be defined to build a model hierarchy, called *anchoring* in [31]. The Strahinger [31] starts with layer 0, which is already a model on its own. Underneath layer 0 is the system under study, represented in the model. The system under study limits the model hierarchy in the direction of instantiation (downwards) and provides a starting point for the hierarchy. However, a model hierarchy should not only be restricted downwards, but it is also helpful to restrict it in the direction of abstraction (upwards)

2. Fundamentals

[31]. If a modelling method should be defined, there needs to be an end of abstraction so that one can use the model hierarchy to create models of the system under study. The language on top is often described using natural language, with which the basic constructs in the meta^Xmodel can be defined. These constructs then lay the foundation to describe all the other model layers. With the meta^Xmodel, the model on the topmost layer is meant. Which layer the topmost layer is, must be defined for each model hierarchy individually.

An indication in the form of a specific relation type must be available to differentiate between different layers. Therefore, it must be possible to distinguish relation types used within a layer and those used between different layers [31].

To build a model hierarchy, models in one layer must have metamodels to them. However, it is not limited to that each model in a layer has only one metamodel created in one language [31]. One model can be abstracted in multiple metamodels, abstracting different properties of the model. [31] therefore differentiate *one-model-approaches* and *multi-model-approaches* (the terms were translated from the German language).

In Figure 2.3, one can see that the two models on the right side (*Model B*) belong together. They represent two views of the system under study and build a model together. They both have their own part of a metamodel describing their core concepts. However, both of the metamodels in layer 2 (*Metamodel B*) belong together and form the core for one modelling language (*Modelling Language B* in the example of Figure 2.3). In the figure, the two views were grouped to visualise that with one meta²-model metamodels for different languages can be described.

If different modelling methods are described in the same metalanguage, they can also be compared, and commonalities or differences can be identified [32, 31]. In Figure 2.3, this is visualised on layer one with the *similar viewpoint* relation. Both have a flow in the model from the top to the bottom with one splitting point. Differences could be recognised if their metamodels are described in the same language. For example, both of them could be used to describe processes.

Geisler et al. [108] propose that often, systems under study cannot be described with only one kind of model, but different views are needed. For example, static and dynamic aspects of the system should be considered. Metamodelling and its metalanguage can be used to define such different views and also connect them. In Figure 2.3, the linkage between model views is shown through the purple arrow between the model types of *Metamodel B*. This link can also be used in the model layer below to relate concepts of different models to each other.

One example of a linkage of different model views, defined on the metamodel layer, can be found in [109]. Among other things, Junginger et al. introduce the Adonis tool for process management, the different applied modelling views for business processes, organisations, and products, and how these models of these different views can be linked. This linkage is defined in the metamodel of the different views. Another example can be found in [110] where multi-perspective enterprise modelling (MEMO) is introduced as a modelling approach for enterprises containing different views (e.g., business processes and information systems), which are integrated and linked on a meta layer.

From a model-driven engineering perspective, [33] defines a metamodel as a model of a model whereby the modelling operation is applied to the base model. The term *meta* means the same operation is applied twice. So, a metamodel is a model of a model, where the metamodel is created by applying the same modelling operation to the model itself. According to [33], it is not enough to call a model a metamodel just because it is a model of the model, but only if an abstraction was applied. Therefore, the metamodel is on a higher abstraction layer than a model.

Two further separations were introduced to further fortify the definition of [33] on what a metamodel is, once between *token models* and *type models*, and between *ontological instantiation* and *linguistic instantiation*. Next, these different categories from [33] will be shortly introduced.

First, the difference between *token models* and *type models* will be briefly summarised. Token models describe a system under study in a one-to-one relation. The goal is to have an understandable description [33]. The token model contains the values of the real system, which can change over time to form a snapshot of the system at a specific moment. The type model then abstracts different tokens of a token model and abstracts them to find similarities between different instances. In this way, classes or concepts of the tokens are created, which are captured in a type model. Therefore, the token model can be seen as an instance of the type model. However, being a type or token model is not an intrinsic property of models. However, it can only be determined in relation to the system under study and what should be described.

Ontological instantiation is used if the instantiation relation between objects in the model or model language is based on its meaning provided through the domain [33]. Linguistic instantiation is based on whether a model object can be seen as a part of the language used to describe the model. Types from the ontological view are more abstract versions of the meaning of the used element (e.g., Socrates as an ontological instance of human). Types from the linguistic view are language concepts that can describe the ontological model (e.g., human is a linguistic instance of a class).

With this, a metamodel is created if a linguistic type model of a model is created [33]. This definition connects the metamodel to a modelling language, which it helps define. According to [33], this language possesses a set of sentences which can be built, and the models, instantiated from the metamodel, are then elements of these sentences.

Regarding the components of a modelling language (notation, syntax and semantics, cf. section 2.3.2) according to [33, 97], a metamodel is used to specify the syntax of the modelling language. Even though syntax alone cannot define *languages* as a whole, it is a central element for describing and connecting the notation and semantics of a language [107].

One way to specify the concrete syntax of a modelling language in the context of using metamodels is to define the notation as attributes of the concepts represented in the metamodel [107]. In this case, the abstract and the concrete syntax are closely related. Kleppe [107] defines semantics as a way to describe the concepts of a language so other humans understand it. However, this is not only on a level where the meaning of different instances of the language (=models) but also on a level where every model created with a

2. Fundamentals

language can be correctly understood. Therefore, the semantic description will also be based on the abstract syntax, as it defines the basic concepts used in the language.

A further aspect often defined in or based on the metamodel is constraints [111], which reduce the number of valid sentences created with the metamodeling language. Based on the concepts in the metamodel, different aspects and how they can be connected in models are restricted. For example, certain relations are only allowed between specific concepts, or models need a minimum number of instances of a specific concept.

Henderson-Sellers et al. [88] define metamodels as abstractions of models, which are themselves models. The metamodel thereby defines what can be expressed in a valid model of a specific modelling language. Metamodels can include multiple hierarchy layers, where a model on one layer is an instance of the type model of the above layer.

Modelling languages, or parts of them, can be defined through the metamodel by creating the concepts available in the language [111]. However, the metamodel must be enhanced with additional information to create a complete model definition. However, these definitions must also be defined in the metamodel layer. Therefore, the definition and creation of a modelling language happens on the metamodel layer.

The metamodel then directly influences what can be described on the model layer [111], and conversely, all created models must comply with the metamodel created one layer above. Therefore, metamodeling is an important aspect of method engineering (see section 2.3.3.3) and supports the design, construction and adaptations of methods [112]. In the context of this work, method engineering is viewed from the perspective of modelling methods, their creation and their application, even though method engineering is not limited to modelling methods.

In the context of method engineering, a further benefit of metamodels is that they can be used to analyse and compare methods [112]. This is especially useful if methods are created from existing fragments that need to be compared and connected to create a new method for a problem.

To function as a shareable specification of the models, which can be created with a modelling language and what information/knowledge should be conveyed with the models [112]. Having this description as a metamodel also supports the creation of modelling tools because if the modelling tool conforms to the metamodel and the language definition, created models are valid “sentences” of the defined modelling language automatically. The metamodel improves the creation of modelling tools by supporting a consistent and integrated implementation of modelling languages [112].

However, having and sharing the metamodel alone is often not sufficient because to facilitate the communication and definition aspect of the metamodel, the metamodel itself has to be interpreted in the right way [112]. Therefore, the metamodel can be seen as a documentation of the available modelling constructs and relations. However, understanding them relies on interpretation.

This problem, or rather a solution, was already introduced in this chapter. It is the creation of a metametamodel, which is the foundation for a language to describe metamodeling languages. Having such a metametamodel (=M²-model) can be used as a basis for the implementation of modelling tools within a platform instead of reimplementing

a tool from scratch for each modelling language [2]. If the M^2 -model is understood by a platform, it can be used to create a corresponding tool automatically. Such platforms are essential because only through modelling tools is it possible to fully harvest the potential of modelling methods and increase the value for the user. Therefore, the resulting modelling tool and its processing capabilities must also be tailored to the domain and its users [29, 41, 28].

In this way, multiple modelling methods can be described using the same metamodelling language, further allowing a linkage between different languages. For example, in [32], the *Bee-Up* tool is introduced as a single tool containing five fundamental and standardised modelling methods. These are BPMN, ER, event-process chains, UML and Petri Nets. The authors use the M^2 -layer to define and create the joint metamodel, containing all the needed sublanguages to implement them in a metamodelling platform. Such integration of different modelling methods needs a proper definition of the metamodels, where metametamodels can help [113].

Having the modelling methods defined in metamodels with a corresponding M^2 model does not only facilitate understanding or allows a linking between different modelling tools but also allows the integration of modelling method [114]. This can also be used to integrate parts of modelling methods and use them together [115].

Metamodelling is a valuable tool that provides many benefits. However, it also faces some challenges. One challenge in metamodelling is the evolution of the metamodels and the metamodelling languages in correspondence to their domains [28, 111]. The challenge here is that if the domain or the usage of a modelling method is changing, the metamodel and the language must also evolve to cover these changes. This adaptation must be consistent with other aspects of the metamodel and influence if models created with older versions are still usable in the new language.

Even though metamodelling is helpful in the context of creating and tailoring modelling methods for business, its use is limited [112]. Leist et al. [112] gather that it is often a long and cumbersome undertaking. On the contrary, businesses are getting more agile and thinking in shorter terms, contrary to the more extended and strategic orientation of method engineering. The undertaking is complicated, and a vast range of expertise is needed to create or adapt modelling methods in a useful way. [112] concludes that metamodelling is less valuable for small businesses but has much potential in bigger enterprises, where management and reaction to the market are increasingly complex.

2.3.3.1. Flexible Modelling Approaches

One challenge for creating modelling methods, especially tools, is adapting alongside dynamically changing domains [28]. This section briefly introduces approaches identified in this context of dynamically adapting modelling methods and tools during their usage.

Atkinson et al. [30] introduce a multi-level modelling tool. The authors argue that in metamodelling platforms, two different types of classification take place: linguistic and ontological. Linguistic means the *type/instance* relation between elements of models and its abstract syntax. Ontological means the (hierarchical) structure between the domain-specific concepts in a metamodel. Both must be recognised if a more useful

2. Fundamentals

meta-modelling tool should be built. The constraints of the domain must be based on the ontological classification. The approach is called multi-level, as multiple levels of abstraction can be modelled. The modeller has access to all the levels and can change them. Therefore, the meta-levels can also be adapted during usage.

If more than two modelling layers are used, concepts that are both classes and objects exist, which must be considered when handling them [30]. For example, attributes from a higher level need a value, and other attributes for the lower level should be specified. In other words, the difference between attribute instance and attribute type [30].

[30] introduced thoughts on how a multi-level modelling tool can be designed, and the authors also discuss requirements, based on shortcomings of other approaches, for this design and architecture. One of the most prominent features is the conceptual separation of the linguistic and ontological type structure. It also discussed how the instantiation of classes/objects and relation/links can be handled and that allowing a user to add domain-specific constraints based on the ontological metamodel is important. They also describe how they want to base the graphical representation of the modelled objects on the ontological metamodel as the priority and only as a fallback on the linguistic one.

In [105], another approach and architecture are introduced to manage multi-level modelling. The introduced tool allows for the adaptation of the meta-model layers at the modelling tool's runtime. It underlines the importance of linguistic representations of a system (not the description in machine code) for its design and interaction with it. These linguistic representations are even more useful if they represent direct concepts of the domain under study. So that the modeller does not have to spend time constructing known concepts with a general-purpose language but can better communicate with people from the same domain or area. However, there is always a trade-off between how easily different people understand a language and how well it can express specific concepts from the domain.

The approach in [105] uses different editors to describe models of different abstraction layers, aka the metamodeling levels. Further, multiple requirements for a multi-level modelling environment are introduced and discussed. Most point towards more flexibility in such a tool and the language itself. For example, the number of modelling layers should not be fixed, and the instantiation should support the easier reuse of already defined concepts.

Laurenzi et al. [43] the authors argue that it is crucial to create and adapt a domain-specific modelling language (DSML) in a timely way. This is often a problem because the model method engineer needs expertise in creating modelling languages and domains. The authors also argue that most approaches for creating modelling languages are sequential, which leads to more time needed to integrate new requirements. Therefore, they introduce an ontology-aided modelling environment, where changes to the modelling language can be made while using the modelling tool.

They also argue that it is crucial for a modelling environment to allow domain-specific adaptations [43]. By this, they mean that the constructs of a modelling language can be newly introduced, removed, modified, or borrowed from other application domains. When defining the concepts for a domain-specific modelling language, the designer always

has to find a trade-off between expressiveness and the reusability of the concepts [43].

In [43], not only is the importance of adding or adapting modelling constructs on the fly highlighted, but these changes in the meta-model must be consistent with previous definitions.

Wüsst et al. [19] introduce a tool which uses touchscreens to enable the sketching of node and edge diagrams. This can be used to design diagrams freely, providing freedom in creative environments. Afterwards, the drawn symbols can be enhanced with lightweight metamodelling capabilities, e.g., by adding types and cardinalities.

2.3.3.2. Metamodelling Platforms and Modelling Language Workbenches

This section will introduce applications which are used to create modelling tools. One of the goals of this thesis is the creation of the *Adaptive Modelling Environment*, which allows the adaptation of the used modelling method during its usage and create models. This will be done based on metamodelling, as introduced above. This section now introduces examples of how metamodelling platforms can be used to create modelling tools.

Metamodelling platforms offer an environment where modelling tools can be established by defining the metamodel [2]. This enables the user to focus on the modelling method and the metamodel, as standard functionality, e.g., the user interface, managing models, exporting them and so on, are implemented in the metamodelling platform and can be easily reused in the modelling tools.

This also enables the users to quickly create prototypes and evaluate the tool with them [28, 116, 43]. However, the modelling tool and the underlying modelling method can be evaluated this way, improving the quality of both.

What functionality is available out of the box or can be added to the modelling tools depends on the used platform. For example, the *Eclipse Modelling Framework (EMF)* and *MetaEdit+* allow the creation of modelling editors by defining a metamodel. However, those environments are focused on code generation [117, 118, 119]. They both allow the definition of different generators, which transform the models into runnable code if enough information is given.

Different metamodelling platforms will be introduced below.

ADOxx is a meta-modelling platform to support the definition and implementation of modelling methods for enterprise information systems[120]. It can be used to create modelling tools for diagrammatic modelling methods.

Using the modelling method notion introduced by [2] ADOxx not only focuses on the definition of the (diagrammatic) modelling language but also on supporting the creation of models (modelling procedure) and tailored or newly implemented mechanisms and algorithms for processing models.

The platform offers a tool for defining and implementing the modelling method and a second for creating and processing models. To support the sharing and usage of the modelling tools, ADOxx offers a way to generate an installable modelling tool over its homepage (www.adoxx.org). To further support the domain-specific processing of models, the platform offers its scripting language[121]. With this, the models can be processed, and external systems can be integrated, for example, by making HTTP calls.

2. Fundamentals

In [43], the authors introduce *AOAME*, a modelling environment based on ontologies that allow the user to adapt the metamodel during the creation of models. Therefore, a basic ontology was created, defining the constructs and the available modelling concepts. They aim to allow users to adapt domain-specific modelling languages to meet their specific needs better.

Not only is the adaptation essential, but also the consistency of the languages with their previous definitions [43]. Therefore, the authors use rules based on the defined ontology to allow a consistent adaptation of modelling methods.

The *Eclipse Modelling Framework (EMF)* is a framework for transforming (ecore) models into other representations [117]. The main focus is on creating Java applications. For example, UML diagrams can generate Java interfaces or XML schema. It is meant to bridge modelling and programming by supporting both.

It can help define an application's data model in a way that allows different transformations [117]. In this way, the user does not need to write the same structure in different languages but can transform them into each other.

Melanee is a workbench for multi-level modelling, which allows users to define a modelling language and its metamodel in one tool [46, 30]. Its focus is the differentiation between the linguistic and the ontological meta-model. The first separates the model from its abstract syntax and uses a *type/instance* relation. The second aspect describes the ontological hierarchy between the domain concepts used in the modelling language. Also, the notion of a *clabject* highlights that a concept in a multi-level environment can have both roles of a class and an object.

However, not only the definition of the modelling language is available, like consistency checks within the ontological hierarchy, visualisation (textual and diagrammatic) adaptations of the modelling language on different layers and transformation support, including the available meta-information [46].

MetaEdit+ offers developers an additional level of abstraction from code or UML diagrams during the creation of applications [118]. This is done by offering users a domain-specific modelling language to describe the goal system. The platform is then able to generate runnable code out of these models. The code generation can be done because the domain is limited (often to one company), and therefore, more expressive modelling methods can be created and used.

The tool offers an interface for creating the domain-specific modelling language and the code generators and another one for creating models [118]. A separate metamodeling language will be used to create the languages. With it, the concepts, properties, and rules can be defined.

The tool was developed to allow automated code creation in an environment where multiple users can work together and create multiple methods [119]. In addition, it should hide the data handling and low-level logic implementation to offer the user a conceptual interaction.

However, *MetaEdit+* is not simply meant to be used alone but also offers integration possibilities through XML and web service [118, 119]. These interfaces allow the user to integrate it into their development pipeline. Also, the tool allows the reuse of already

defined method parts to create new methods.

The *XModelerML* is a multi-level modelling environment focused on creating modelling languages and models in the context of code generation [105]. It uses the FMML^X (cf. [105]) execution language, which is itself based on *xcore*. The goal of the environment is to allow a connected representation of the modelling language, the models and the corresponding code in one environment [47].

The tool implements the architecture and thoughts introduced in [105], which underlines the importance of using domain-specific languages and allows for an unlimited number of modelling layers. The abstraction layers are separated into linguistic and ontological parts. The instantiation capabilities should also support the reuse of already-defined concepts.

2.3.3.3. Modelling Method Engineering

As mentioned in the previous subsection, creating modelling languages/methods is challenging. Therefore, scientific communities and literature have created guidelines, tips and requirements for creating such or similar methods. Even though some guidelines exist for creating modelling methods, compared to the numbers for creating programming language, only a few examples exist [98].

This section will discuss different approaches to how modelling methods can be created to provide context for essential tasks that must be considered and what information must be examined. This section will not focus on the requirements towards a modelling method, as they are discussed in chapter 4.

Modelling method engineering is the umbrella term for methods, guidelines, best practices or similar, which focus on supporting users in creating a modelling method. In the context of this thesis, input from modelling method engineering will be used as input for the conceptualisation of the *Adaptive Modelling Environment* of this thesis and as input for requirements on how such an environment can support the creation and adaptation of modelling methods in the context of digital twins from design thinking.

The focus here is not only set of conceptualising a modelling method but also how this influences the modelling tool, as nowadays the usability of models and their value can only be fully utilised by using digital tools [34, 105]. Further, modelling tools can support the creation of modelling methods, as they enable fast prototyping and an early evaluation [116].

Not only the procedure and the included tasks can differ from method to method to method, but also the starting point itself. Selic [98] therefore defines three general categories:

- *Refinement:* The more general concepts of an existing modelling language are specialised into the more domain-specific constructs.
- *Extension:* An existing language is expanded with new domain-specific constructs.
- *Definition:* A new modelling language is created from scratch.

2. Fundamentals

Each of these categories comes with its benefits and drawbacks [98]. Defining a novel modelling language grants the most freedom and allows for a perfect fitting of the language with the problem domain and its usage. On the other hand, this is the most complex and time-consuming undertaking. To create a beneficial language, expert knowledge and experience in modelling method engineering are needed, usually not provided by domain experts themselves, but modelling methods engineers must be consulted. On the other hand, refining existing methods is the most practical and cost-effective approach for defining domain-specific languages. Such an approach is subject to the most restrictions from the original language [98].

Below, a short overview of selected modelling method engineering approaches is provided. They show different approaches to how modelling methods can be and are created.

2.3.3.3.1. Using UML Profiles for Creating a Modelling Language Selic [98] introduces a procedure on how Unified Modeling Language (UML) and its profiles can be used to create domain models and corresponding languages. This is an example of using an extension to create a domain-specific modelling language. It uses two artefacts in the procedure: the domain model and the UML profiles.

The procedure starts with defining the domain model, which will later be used as the metamodel of the modelling language [98]. It should be created without considering the UML metamodel, to be not influenced or restricted by it, even if it cannot be directly mapped to the UML metamodel later. The domain model should contain fundamental language constructs, relationships, constraints, concrete syntax and semantics.

After the domain model is created, it should be mapped to the UML metamodel by identifying the most fitting UML meta concepts for the created domain model [98]. This includes thoroughly analysing the chosen meta-concepts that fit semantically into the domain concepts. Therefore, the constraints on the metaconcept, its attributes, and its relations to other meta-concepts must also be checked. If this is done for each concept in the domain model, the domain-specific modelling language is established.

One benefit of using profiles and stereotypes over extending UML directly is that they can be added and removed without changing the underlying model itself [98]. If a class is directly added to or removed from the metamodel, this will influence existing or future models on a fundamental level, which must be considered during the creation.

Concluding, for the context of this thesis, the above-introduced approach would restrict the overall idea of this thesis, as it is based on the UML method and how it can be extended. Therefore, such an approach will not be the focus of this thesis.

2.3.3.3.2. Agile Modelling Method Engineering Karagiannis [28] introduces the Agile Modeling Method Engineering (AMME) framework for adapting modelling methods to evolving requirements from the domain. The focus is set but is not limited to adapting existing modelling methods to make them fit new usage scenarios. It highlights the importance of domain-specific modelling methods and that they must be tailored to their domain and evolve alongside it.

In this context, it is important to have tools that support a fast adaptation of the

modelling method and its tools, for which [28] proposes using a metamodeling platform. This allows not only the adaptation of the modelling tool but also the evaluation of the conceptual changes to the modelling method through a fast prototype approach [116].

Within this framework, the AMME life-cycle is introduced as a procedure which guides the adaption of modelling methods [28]. However, the procedure itself is agnostic if it is used for adaptations or the creation of new modelling methods. Therefore, five steps are introduced in an iterative life cycle. Iterations cannot only be made on the five steps as a whole, but feedback cycles also occur between two neighbouring steps.

The five steps, as introduced in [28], are:

- *Create* focuses on understanding the domain and its problems and subsequently defining the requirements, goals, important concepts and other fundamentals of the modelling method.
- *Design*, which takes the insights from the previous step design the modelling method itself. Therefore, the metamodel, the notation, functionality and other concepts are conceptualised and specified so that, in the end, a coherent definition of the modelling method is established.
- *Formalize*, which takes all aspects designed in the previous step and describes it in a non-ambiguous way so that it can be shared and as preparation for the next step.
- *Develop* focuses on implementing a modelling tool for the formalised modelling method. The result is a usable tool, including the modelling language and its functionality, which can be used in a testing or real environment. Based on the implementation environment, adjustments to the design may be necessary.
- *Deploy/Validate*, which focuses on providing the developed tool to stakeholders so they can use and evaluate the results of the current iteration. This includes preparing the tool to be easily installed at the customer's site.

In conclusion, some thoughts on how the introduced modelling method engineering approach can be seen in the context of this thesis are provided. If the procedure introduced above is made usable during the use of a modelling tool, some of the steps will be influenced and may need support. The creation depends on the users and cannot be directly supported by a modelling environment during usage.

Design can be supported by allowing the definition of new or adapting existing concepts, relations, attributes and similar during usage. Here, it must be differentiated after the user. If the users are no experts in modelling method engineering, then the functionality for the creation or adaptation should be restricted to support the definition of useful modelling methods.

The formalisation phase will not be applicable during a usage-based adaptation, as it would take too much time during a workshop. However, a weak formalisation will be provided as an implemented modelling tool.

The development and deployment steps are native to the approach, as the changes will be directly implemented in the tool and then made available to the users. Then, the

2. Fundamentals

users can be evaluated during the usage of the tool by trying to model the problem they recognised and which triggered the adaptation. Here, another aspect is how the changes can be mitigated for other users of the modelling tool.

2.3.3.3.3. Modelling Language Development - Macro Process The *Modelling Language Development - Macro Process* was introduced in [29] and focus on providing steps on how a modelling language can be defined. Even though creating a modelling tool is mentioned, it is not the focus of the introduced procedure, as it mainly focuses on how the modelling language can be conceptualised.

Each of the introduced steps contains a procedure and tasks that should be accomplished. A short overview of the macro process steps is provided below [29]:

- *Clarification of Scope and Purpose:* Here, the design objectives and the budget will be defined.
- *Analysis of Generic Requirements:* Definition of the generic requirements and creation of the corresponding documentation. Generic requirements apply to modelling languages in general, such as the ability of all objects of the real world to be represented or the clear distinction between the different concepts provided in the modelling language.
- *Analysis of Specific Requirements:* Definition of the specific requirements for the modelling language. The specific requirements are depending on the domain of the domain-specific modelling language.
- *Language Specification:* This step defines the metamodel and the corresponding information, like the constraints, to capture the structure and semantics of the modelling language. Here, input from experts, like modelling methods engineers or domain experts, is needed.
- *Design of Graphical Notation:* This step builds upon the previous and designs appealing graphical notations for identified concepts. The notation must correspond to the semantics of the language and its concepts.
- *Development of Modelling Tool:* In this step, the modelling tool, corresponding to the definition of the previous step, is made. In the procedure of [29], no guidelines for the development are provided.
- *Evaluation and Refinement:* This step defines and executes a systematic and comprehensible evaluation procedure for the modelling language and tool. Its goal is to improve the quality.

In conclusion, some thoughts on how the introduced modelling method engineering approach can be seen in the context of this thesis are provided. This thesis can only indirectly support these three steps, which usually belong to a more extended planning phase. However, the approach of this thesis focuses on allowing a fast adaptation of the

domain to capture knowledge from design thinking workshops. However, the users still consider the requirements, even if they are implicitly available.

The fourth and fifth steps can be supported by allowing a direct adaptation and creation of the modelling method, directly supporting the design and graphical notation in the tool. The development and evaluation can be supported as the tool automatically adapts the tool to the needs.

2.3.3.3.4. Situational Method Engineering Situational Method Engineering (SME) is an approach for supporting software development, where the focus is set on tailoring the used methods to the specific needs of a certain group, e.g., an enterprise [88]. In general, this method uses pre-defined sections, which are called method parts, to build a method tailored to the specific needs of a group. These parts are combined to create the needed method. It does not matter if the users start from scratch or choose an existing method as a base. The result is a method that can cover all identified cases.

To support the reusability of method parts, they must be described in a standardised way, which can be supported using a metamodel [88]. Putting such parts together results in a tailored method that can be used in a certain situation. However, to enable users to create their methods based on method parts, a storage facility of methods parts is needed, from which they can be selected [49].

Once a method is created for a specific situation, it will be applied for its intended purpose, leading to new requirements tackled by utilising other method parts [88]. Therefore, a tailored method will be created once and then used and adapted to newly identified or evolving requirements. Here, it will also be differentiated how sophisticated a method must be, as this differs in the size and importance of the application context. For small projects, the tailored method will be on a lower sophistication level so that the effort does not outweigh the benefit.

The tailoring of the methods to concrete projects increases the method's applicability in the context and reduces the overhead imposed on the users [88]. More sophisticated methods are often more expressive and can cover more different cases, but learning such methods is difficult and slows down project application. Smaller tailored methods are often easier to learn and apply.

A procedure is discussed in [49] to guide users in creating their own methods based on existing method parts. The first step is to decide if an existing method can be taken as a starting point or if one must be created from scratch. Therefore, the goal of the method and its engineering process must also be defined.

Afterwards, the method can be created and evaluated [49]. The method itself can be created by applying different approaches. For example, an assembly-based approach can be used, in which different parts of the method are analysed, compared, and then selected. This is done by analysing them in the context of the defined method goal. Then, the different parts must be assembled to create a coherent method, which can be done by integrating or associating the parts. The method parts can be chosen from existing ones or explicitly created for the new method.

However, the process of how exactly the method is created is not the focus of SME,

2. Fundamentals

but here to adapt the method to concrete projects or for a concrete group of people by reusing parts of methods and designing the methods that parts of them can be reused [49].

In conclusion, some thoughts on how the introduced modelling method engineering approach can be seen in the context of this thesis are provided. In general, it fits this thesis, which is to adapt modelling methods to concrete domains or problems.

The idea of reusing existing method parts can benefit the idea of this thesis but also imposes the need for knowledge on the users, which may not be all feasible. Therefore, if such an approach should be applicable, it should be based on a non-mandatory basis. So that experienced users can reuse existing parts and the adaptation is usable for inexperienced users. Allowing the re-usage further imposes requirements on the tool support, as a way to save and combine such method parts must be supported.

Another aspect that should be considered is that this thesis aims to use conceptual modelling in design thinking workshops, where adaptations must be made quickly and fit the domain. Therefore, needing to consult a repository with the methods part can be overwhelming for the users. Additionally, SME uses well-defined methods parts for combining them into a tailored method. Defining these parts in the context of digital twins from design thinking can be difficult, as the parts must be created sophisticatedly, which is also a non-trivial task.

2.3.3.3.5. Example-Driven Meta-Model Development López-Fernández et al. [45] introduce a metamodeling development method in the context of model-driven engineering, which aims to allow domain experts to contribute to the creation of a metamodel in a direct manner.

The basic idea is to apply a bottom-up metamodeling approach by allowing both experts to create example models from which the metamodel is derived [45]. These example models do not need to be full models but can also be individual fragments of the models. However, they document the requirements, and a method engineer can then evaluate the derived metamodel. After the metamodel version and the corresponding language, it should be tested again with the domain and technical experts.

Through the evaluation, problems or new requirements can be identified, allowing the creation of the metamodel in an iterative and exploratory way [45]. Through this, the metamodel can evolve naturally until it is useful to the experts.

After the metamodel is finalised, it should be translated to fit a specific platform and then become usable [45]. By creating the metamodel detached from a specific platform, the created metamodel is not restricted and fits the problem space better. Therefore, the created example models which [45] mentions are not conceptual models but only diagrams which should later become full models.

In the proposed procedure (cf. [45]), first, the domain experts create the example models that should be available in the resulting language. Afterwards, a method engineer can adapt or annotate the example models. The metamodel is automatically induced from the annotated example models, and a digital assistant makes improvement suggestions. If problems are detected, the metamodel and the example models are reworked by the

domain experts or the method engineer. If no problems are found, the metamodel is evaluated and implemented in the target platform.

In conclusion, some thoughts on how the introduced modelling method engineering approach can be seen in the context of this thesis are provided. Deriving the metamodel from examples is a promising approach that can support this thesis's goal. However, keeping the metamodel target platform independent till the last step is not feasible here, as the digital model is used within the workshop. Therefore, the machine-interpretable version is needed on the fly. However, the basic idea is similar to instance-based metamodeling as it is introduced later in section 5.1.2.

2.4. Artificial Intelligence Technologies

This section will shortly introduce Artificial Intelligence (AI) approaches used to support the design and implementation of the digital twin from design thinking. They are essential to let the *Adaptive Modelling Environment* understand the semantics of parts of the digital twin, enabling the environment to support the users better. For example, without a semantic understanding of the tangible models created with design thinking techniques, the digital model cannot be automatically created and used directly during the workshop. The information provided below introduces the approaches and provides the aspects that are interesting in the context of this thesis.

2.4.1. Ontology

Within this thesis context, computer science ontologies are used to save information in a machine-understandable way and to share information between different components of the *Adaptive Modelling Environment*. For example, to derive additional information which can be added and shown to the conceptual models in the modelling tool.

The term ontology is widely known for its philosophical roots, aiming to structure and describe reality correctly and universally validly [122]. In computer science, we use the term ontology for a formal and explicit representation of knowledge of a shared understanding containing a shared set of terms, with shared semantics, relations between the terms and rules to describe meaningful structures. Through this, the data context can be described in a machine-processable way, enabling machines to understand the data better.

In this context, the *formal* feature is important because this allows for a computer-understandable representation and supports the sharing between applications and automation of tasks upon the data structure [122]. Here, the terms used in the ontology must be linked to a shared understanding of the referenced concept.

One such formalism, which comes from the context of the semantic web movement, is the *Resource Description Framework (RDF)* (cf. [123]) or, to be, more specifically, the *Web Ontology Language (OWL)* (cf. [124]) language. At which the knowledge is described in statements, consisting of *Subject*, *Predicate* and *Object*. This fundamental structure is called triples. With these triples, a graph can be created, which functions as a formal

2. Fundamentals

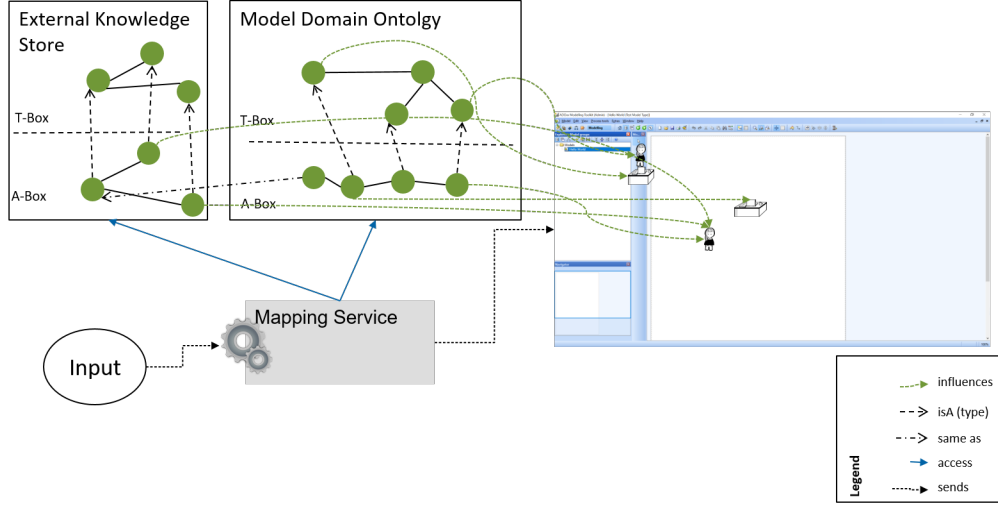


Figure 2.4.: Conceptual representation of the usage of the ontology

description and can be processed by the machine. The goal is to have a standardised way to describe and link information, which is understandable by the machine [122].

One additional benefit of RDF is that data entries and the meta-information are encoded in the same structure and that instances (data entries) can be created without a description of the class structure [122]. Therefore, only the data can be saved, but if a class structure is available, it can be used to derive additional data. This is accomplished by applying reasoning to the data set to support the extraction of implicit knowledge.

The semantic web approach possesses a technology stack for supporting the usage of semantic web technologies [122, 125]. The stack has had some revisions, but the core specifies vocabularies and technologies based on RDF, which should enable the semantic web. For example, ontologies/languages exist to describe the syntactic class structure (RDF Schema) or more semantic properties (OWL). Further, it contains ways of querying information (SPARQL), defining rules and more.

In Figure 2.4, the basic conceptual approach of using ontologies in the context of the modelling environment can be seen. Input from an external system is provided; for example, this could be user input or come from a recognition based on convolutional neural network (CNN) (cf. section 2.4.2). Afterwards, the concept is looked up in a particular created ontology to extract knowledge about the concept. This knowledge contains the name and other characteristics of the concepts, which can then be represented in the modelling environment. With the basic language of ontologies, distributed knowledge can be referenced, which is here used to map a concept to a *External Knowledge Store*, which is also based on RDF. An example, would be *DBpedia* (<https://www.dbpedia.org/>). With this, the available characteristics for a class can be defined, and the knowledge of already created instances can be reused.

By defining a class structure, RDF enables automatically deriving additional information, which the users do not have to define explicitly [122]. Therefore, the class structure must

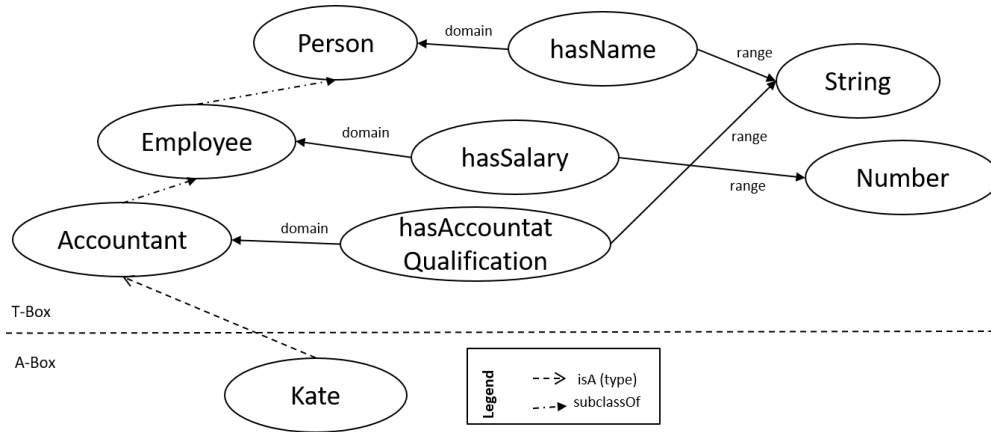


Figure 2.5.: Example hierarchy of classes and an instance

be semantically described so the machine can understand it. In the context of RDF, standardised vocabularies can be used. *RDF-Schema (RDF-S)*⁵ to describe the class structure or *OWL*⁶ can be used.

For example, RDF-S contains the *subClassOf* relation, which has the pre-defined semantic to define class hierarchies [122]. This is also defined as a *transitive* property, enabling a further utilisation of a defined class structure.

OWL offers various relations with pre-defined semantics, which can be used [122]. For example, relations can be defined as symmetric, asymmetric or functional.

This supports retrieving additional information, as visualised in Figure 2.4 by minimising the data that must be provided as input to get a more sophisticated representation.

Figure 2.5 shows an example of a small hierarchy of three classes and one instance. With the defined relations, it is possible to quickly find all characteristics used to describe the classes of *Kate*. We have to find all classes, of which *Kate* is an instance, which is: *Accountant*, *Employee* and *Person*. Afterwards, we must find predicates with these classes as *domain*.

2.4.2. Object Detection

The next AI approach that will be introduced supports the *Adaptive Modelling Environment* in identifying the tangible objects used in physical workshops and, therefore, enables the transformation in the digital model. This is one fundamental ability the environment must possess. There exist different ways to solve this problem. However, this section will focus on object detection using convolutional neural network (CNN) based machine learning, as it is a widespread approach in computer vision.

Therefore, we will introduce a way that allows *Adaptive Modelling Environment* to interpret pictures to generate models. In the context of this thesis, this capability is

⁵<https://www.w3.org/TR/rdf-schema/>

⁶<https://www.w3.org/TR/owl-features/>

2. Fundamentals

essential to allow users novel interaction capabilities with the modelling environment during physical workshops. The automated way allows the use of digital models and, with it, the digital twin from design thinking, as the digital representation is instantly available and does not have to be created manually.

In computer science, the field that analyses how machines can interpret and analyse pictures of the real world is called Computer Vision (CV), a difficult task to accomplish [126]. Humans can easily make sense of their visual input from the surrounding world, but for computers, this is not easy, and applied algorithms are often failure-prone. Computer vision is therefore considered a part of AI. It aims to reproduce the cognitive capabilities of humans to understand pictures, like, for example, separating the fore- and background or identifying objects within a picture. [126].

Within CV, visual recognition of objects is a crucial problem [127, 128]. Its core focuses on learning categories and then visually identifying new instances of these categories in pictures. Thereby, recognition approaches are often separated into specific objects and generic categories. The first tries to recognise specific instances in pictures, such as a concrete person or building. The second one aims at recognising objects and assigning them to classes like persons, buildings, cars and so on [127, 128].

The recognition output can vary depending on the application context [127, 128, 129]. It could be a list of labels from objects identified within the picture, called image classification. The next step would be to draw a rectangle around the object, often called bounding-box detection. Alternatively, contouring the object in more detail is known as figure-ground segmentation. In other words, segmentation is the separation of the objects from the background. The identification of an object in a picture and determining its position is also called object detection [127, 128, 129].

The instances or categories are identified by extracting their features in pictures and mapping them to the class or instance [127, 128, 129]. This feature extraction is done while learning the detection model. Based on this model and the identified features, the objects can be found in a picture and labelled. Features can be the colour, shape, texture or something similar. Thereby, different objects can be identified using different features or a combination of them [127, 128, 129].

To achieve any of the outputs mentioned above, a model for visual recognition must be created, which is usually done through training with a set of prepared training images [127]. From these, an algorithm can adapt the model to identify instances of the defined classes (independent of instances or categories that should be recognised). The training must consider different influences on the objects, which should be recognised. These can be considered by adding pictures that influence the training images. Pictures of objects can often contain different environmental influences like lighting conditions or the background, additionally to the different shapes the object can adopt [127].

When using object recognition or detection, usually a trade-off must be considered if *accuracy* or *efficiency* is favoured [128, 129]. The first one regards how accurately an object is identified, including whether it is identified and given the correct class/instance. It is also essential that the object is correctly localised in the picture itself. The second one engages with questions about time, memory, and storage consumption. These aspects are

influenced by the concrete chosen recognition method but the variation and characteristics of the objects in the training images [128, 129].

In Bansal et al. [129], different classification techniques for object recognition are introduced and discussed. They can be classified into supervised (see also [130]) or unsupervised classifiers. For the first class, a dataset of training data with labels is available to determine the model to identify objects in pictures. There are no labels for the dataset by unsupervised classifiers, but the algorithm clusters the given dataset into groups based on a similarity measure [129].

[129] lists the following examples as supervised clustering techniques: BayesNet, Decision Tree, Convolutional Neural Networks, k-Nearest Neighbour, Multilayer Perceptron, Naïve Bayes, Random Forrest and Support Vector Machines. Examples of unsupervised classifiers are K-Means, Fuzzy Clustering and Hierarchical Clustering [129].

According to [128, 130], machine learning approaches based on (deep) neural networks lead to potent object recognition approaches. They are excellent for finding structures within high-dimensional data. Therefore, they are often applied to complex problems like image or speech recognition. In [131], a deep convolutional neural network is introduced, improving object recognition performance in vast and complex datasets.

2.4.2.1. Deep Learning

Deep learning uses a multi-level neural network, which can be adapted through learning algorithms [130]. The levels consist of neurons that use diverse mathematical functions to activate each other. Therefore, each level takes its input and transforms it into the corresponding output, which is input for the next level. The output from the last layer is then the output of the deep learning model. The learning process adapts the parameters of the mathematical functions and neurons and, in this way, shapes the final output in the wanted direction.

Having these multiple levels allows for learning more complex models, bringing better results in recognising objects in pictures [130]. If they are used for classification problems, earlier layers can highlight important features and suppress unimportant ones. In deep learning, the layers must not be designed by hand but through machine learning [130].

In deep learning, input, output and hidden layers are used. The input layer takes the prepared input and feeds it into the net [130]. For example, this can be a picture encoded in matrices. The output layer defines with which probability an input belongs to a class known by the network. For example, a label is provided, shown to a user or further processed. The hidden layers contain multiple layers of neurons, which can have different characteristics to process the input and lead it to the wanted output [130].

convolutional neural network (CNN) were found to be easier to train, and compared to networks with fully connected adjacent layers, they generalise better [130]. This increases use in the CV community, as a generalisation is needed to find the trained objects in new pictures. CNN create better results in object detection, as they consider singular pixels and their surroundings.

2. Fundamentals

2.4.2.2. Convolutional Neural Networks

In the rest of this section, the focus will be set to convolutional neural network (CNN) as they are a well-established approach to object recognition in pictures and are widely used in this domain [130]. Primarily, the constant improvement of hardware and algorithms led to its widespread application of image recognition and other scenarios [130].

CNNs are often used for cases where the input can be encoded as multiple arrays like pictures [130]. They can be represented as three two-dimensional arrays, each with pixel density in three colour channels. One benefit if CNN is that it can consider areas of information, so not only one data point but also its neighbours are considered. This is often useful in pictures because the meaning of a single pixel can be enhanced by looking at its neighbours. For example, a line can be identified by two different colours on each side.

The analysis of such areas is done through what is called a filter [130]. Each filter recognises other shapes or constructs in pictures. Such filters can also be combined to recognise complex structures.

As CNN are based on typical neural networks, they consist of neurons which can be enhanced through learning algorithms [132]. Neurons are separated nodes which can process data and are connected to create a neural network. Nodes in a CNN behave similarly to those for a typical neural network. They receive input and calculate an output. How the output is calculated must be determined during the net design. The difference to typical neural networks is that CNNs are designed to recognise patterns within pictures easily, hence their widespread use in object recognition [132].

Traditional neural networks can also be used for object recognition. However, this leads to massive networks that focus on particular features [132]. The first problem can cause problems because computational resources are not unlimited, and the second one can lead to overfitting, where the net is not abstracted enough to recognise objects within unknown pictures. CNNs are preferred because they are good at recognising objects in pictures by remaining smaller than traditional neural networks. In CNN, the first problem is tackled by connecting the neurons only to regions of the input layer and not to every aspect, as it is done with fully connected neural networks [132].

A CNN usually consists of different types of layers, which together are used for recognising objects in pictures [132]. The first type of layer is called the input layer and contains the pixels of the input picture. The second type is called the convolutional layer and gives the name to convolutional neural network. Here, filters are applied to search for patterns in smaller portions of the picture. The next type is called the pooling layer, which is used to reduce the dimensionality of the input. Last but not least, fully connected layers are used to map the generated score to the output classes. Layers can exist more than once, and convolutional and pooling layers can take turns. In addition to the convolutional layers, rectified linear units (ReLUs) layers often exist, which transform the output feature map of the convolutional layer to remove negative values [132].

As the name suggests, the convolutional layers are essential for CNN [132]. They take their input and apply filters or kernels to recognise patterns within the input. However, not all of a picture is analysed at once, but only regions of it, which is called the receptive field.

The picture that should be processed is usually encoded into a matrix where the numbers represent the pixels. The columns and rows represent a picture two-dimensionally (width and height). Different layers (often called channels) of this two-dimensional representation can be used to represent one picture. For example, if an RGB colour scheme is used, each channel (or layer) can be used for one colour [132].

The kernel has a width and height defined [132]. During the recognition, the kernel glides over the input and convolves the input towards the output. This is done by calculating the dot product for each square on which the kernel is applied. The window of the kernel is applied to squares of the same size in the picture. The stride parameter defines how much the kernel is moved between each application, defining the number of pixels the kernel is moved. Each value calculated in this way is saved in a new matrix called the feature map. The position of the value in the feature map depends on the window on which the kernel was applied [132].

In Figure 2.6, the application of a kernel is visualised. The visualisation is based on an online lecture held by Jeremy Howard, which was uploaded to YouTube in 2016 (cf. [3]). Amongst others, the lecture explains the application of convolutional layers on a number taken from the MNIST database (<http://yann.lecun.com/exdb/mnist/>). In the figure, one can see the input on the left side, the kernel in the middle and the feature map on the right side. The green markings highlight the kernel window and where it is applied. The nine numbers within the kernel are transformed into one number in the feature map. Even though the use case is simplified, one can see that the input represents the number "4". The picture is then decoded in zeros and ones, whereby the zeros represent white, and the ones are black.

The "4" should be recognised by the convolutional neural network, and therefore, the kernel is applied. The kernel is used to detect features within the picture. In Figure 2.6, the kernel tries to find horizontal lines within a picture. This is depicted by the 1s in the middle row. One can see that in the feature map, the highest number is at the position of the horizontal line of the input, which means that the horizontal line is found here.

Regarding the kernel application, different parameters must be defined and can be used to optimise a CNN. The stride parameter was mentioned and defined how many pixels the kernel window is moved before each application [132]. For example, if we take Figure 2.6 and define a stride of one, the whole window will be moved one column to the right. The feature map in the figure was created with a stride of one.

The second parameter is the padding, which defines if a border is added to the input [132]. Usually, if such a border is added, it is filled with the value zero, called zero-padding. This allows a feature extraction at the edges of the input. In Figure 2.6, the resulting feature map is smaller than the input. This is because a padding of zero (therefore, no border) was applied.

In a real scenario, the values in the picture would not be zeros and ones, but the colour intensity in an RGB model and instead of one two-dimensional matrix, three two-dimensional matrices will be given, one for each colour of RGB [132]. In a real application, the kernels in the first layers of the CNN detect simple geometric shapes like lines, rectangles or circles. Moreover, these different shapes are combined in the later

2. Fundamentals

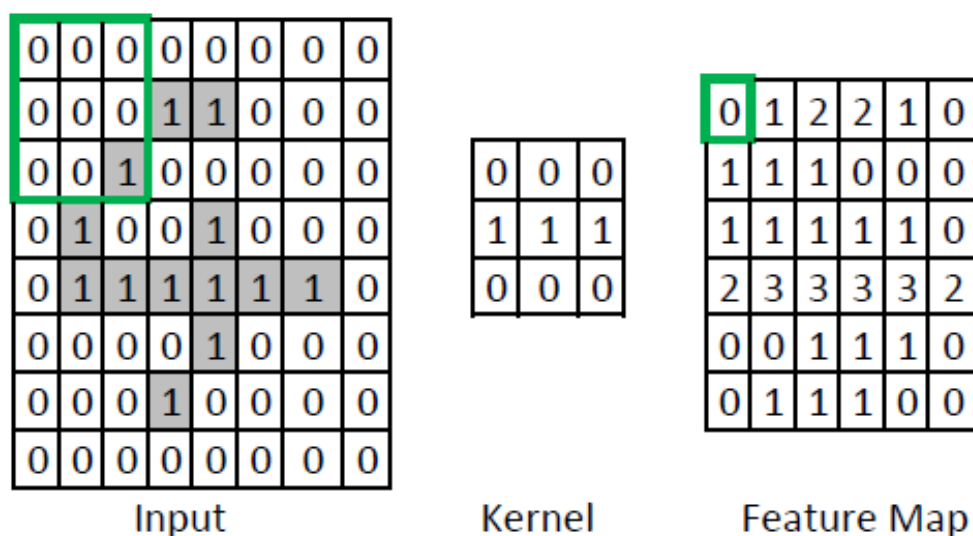


Figure 2.6.: Visual representation of the application of a kernel. The representation is based on a lecture by Jeremy Howard titled Lesson 4: Practical Deep Learning for Coders, which is available on YouTube. [3]

layers to recognise more complex patterns.

An example is provided in [133], where features of faces are recognised in different stages of a CNN. First, simple shapes like curves or lines are recognised. Next, they are combined into more complex features of a face, like a nose, eyes or a mouth. Deeper in the net, these features are combined to recognise complete faces. In general, the first convolutional layers can recognise more simple patterns, which are later combined with more complex ones [134].

The next type of layer is called *Pooling Layer* [132, 134]. Pooling layers are used to decrease the computational complexity by reducing the dimensionality. It processes its input, usually obtained from a convolutional layer and reduces it by applying the pooling function. Often, a max-pooling of a two times two matrix with a stride of two is used [132, 134].

Pooling is applied to enable a CNN to become more robust against the position or lighting condition within pictures [135]. Therefore, it helps to focus on the essential features and lower the influence of unimportant features. Pooling is applied by defining another window, which glides over the input. If max pooling is used, it takes the max value of the window and saves it again in a new matrix. Then, the window is moved, and the max value is retaken. This reduces the overall size of the input and highlights the important features. Max pooling is not the only pooling method which can be used. For example, average pooling can be used to calculate the mean of the numbers in the pooling window and save it to the new matrix [135].

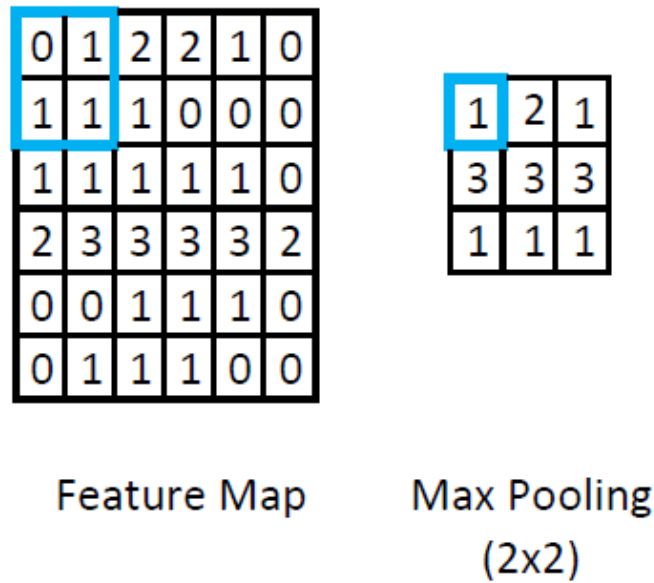


Figure 2.7.: Visual representation of the application of the pooling operation. The representation is based on a lecture by Jeremy Howard titled Lesson 4: Practical Deep Learning for Coders, which is available on YouTube. [3]

In Figure 2.7, an extension to the example from Figure 2.6 can be seen. Here, the feature map from Figure 2.6 was taken, and two times two max pooling with a stride of two was applied. The blue rectangles show which numbers were considered from the input and where they were saved after the max pooling. Max pooling takes the maximum number, which is the digit "1" in the blue rectangle of the left side of Figure 2.7. With a stride of two, the whole window is moved two numbers to the right. This leads to the following number for the result matrix, which is "2".

The next layer type is called *Fully-connected Layer* and consists of neurones which are connected to all the neurones from the layer before and the layer after it [132, 134]. A CNN can have multiple fully-connected layers. They are used to map the output from the feature extraction part of the CNN and map it to classes of recognised objects, which can be used as the net's output. A fully connected layer is organised similarly to traditional ANN [132, 134].

However, the introduced components of CNN are not the only aspects one must consider if such networks should be used [132]. One crucial factor is how big the pictures fed to the network are because their size directly influences the computation complexity. Further, how much and which picture variants are fed for the training is also essential for the success of the CNN [132].

This introduction to convolutional neural network does not claim to be complete and covers all possibilities on how CNN can be used and applied in practice. This introduction

2. Fundamentals

should provide a short overview of the capabilities and the general procedure of CNN.

There are many different designs and implementations of neural networks for object detection [128]. Even though most of them are based on the CNN architecture, there are still many variations within the general structure of CNN, which have their advantages and drawbacks [128].

According to [128], the RFCN (Region based Fully Convolutional Network [136]) needs more time to train but is accurate and can use pictures of arbitrary size as input. On the other hand, YOLOv2 (You Only Look Once [137]) and SSD (Single Shot Detector [138]) are faster in training and still very accurate but need a fixed input size and have problems by detecting small objects. [134] contains a comparison and analysis of different kinds of approaches to object recognition. In that paper, different kinds of the *SSD* and the *YOLOv2* architecture are evaluated as the most precise or fast, depending on their specific configuration. An overview of the comparison which is discussed in [134] originally published in [137].

However, to decide which one to use based on such comparisons and benchmarks introduced above is tricky [128]. This is because the different data sets and implementation influence the results. The implementations are constantly improving and adapting. However, there is often a trade-off between performance and accuracy. However, the architectures often found to be the better ones are the ones listed above. Types which are often mentioned are, for example, Faster R-CNN or YOLO. This can also be seen in [134, 128]. The topic of which CNN approach to use for a problem is also a topic of many different websites. Two examples are: [139, 140]

There also exists approaches which try to make object recognition accessible for devices with limited resources, like mobile devices [141]. Therefore, the nets must have a high performance, and the saved network is small. Here, the file size, which occupies storage space on the device, is meant. Fang et al. [141] introduce a net called *Tinier-YOLO*, which is one net created to be used by devices with less performance.

2.4.2.2.1. CNN configuration: How well a CNN or other artificial neural networks can be trained to fulfil its goal depends on the overall architecture of the network, the training data and the configuration of the network [130, 142, 143]. In the following, some of the needed hyperparameters are introduced. The hyperparameters are not limited to CNN, but the focus here will be set to this kind of net because it was used within this thesis. Additionally, different parameters could be necessary for different implementations of CNN.

The configuration is done via variables called hyperparameters, which are adapted before the training is started [142, 143]. Different configurations are often tested before the final network is finally trained and used.

One important hyperparameter is the **number of hidden layers and unit** [144, 143], which defines how many layers are between the input and the output layers of a CNN. Adding more layers improves the network results but needs more computing power. Having too few layers can lead to underfitting.

The **learning rate** defines how much the weights are updated during the training. A

high rate is faster but can have problems by converging, and a lower takes longer time but converges better [144, 143]. The learning rate will start higher and then be reduced during the training.

Momentum defines how much the history of training influences the next step [143, 145]. The history is considered so that one round of training does not influence the adaptation of the weights too heavily over the current trend.

The *batch size* defines how many pictures/samples are given to the CNN at once for one iteration of the training [145, 144, 143]. The training data set is separated into smaller batches, which are given to the CNN and used one after another to update the weights in the network.

There are also some dependencies and influences between different hyperparameters. For example, Kandel et al. [142] tested the correlation of batch size and learning rate for image recognition in the medical field and concluded that larger batch sizes should use higher learning rates and vice versa.

Another aspect, besides the architecture and the hyperparameters, which influences the performance is the training data [146, 147]. This is a set of labelled images, which is cumbersome to obtain. Labelled images mean that for the images in the training data, the objects which should be identified are marked with a bounding box and the corresponding label. Here, data augmentation can artificially inflate the training set [146, 147]. Data augmentation itself is a challenging task, and different strategies exist to do it. For CNN, it usually contains the creation of additional pictures from the ones in the dataset. For example, one popular method (cf. [147]) is flipping the pictures over their horizontal axis. The augmentation thereby not only helps to increase the dataset for the training but also helps to prevent over-fitting by introducing more variance in the training set [146].

Augmentation strategies should be applied to the already labelled pictures to reuse the labels for the augmented pictures. Therefore, if the picture's position is adapted, the bounding box must also be newly calculated [147, 146]. For example, the colours of the pictures can be adapted (e.g., brightness), the layout of the picture can be adapted (e.g., rotation), or only the content within the bounding box can be adapted (e.g., rotation) [147]. Different strategies possess their own parameters with which can be adapted.

Taylor et al. [146] compared different augmentation strategies to each other. They grouped them roughly into two categories: geometric methods and photometric methods. For the first category, the objects within a picture are not altered but rearranged within the picture. Examples are flipping, rotation and cropping. The second category manipulates the RGB channels to change the picture and adapt its colours, lightning and similar. Examples are colour jittering, edge enhancement and fancy PCA Principal Components Analysis.

Zoph et al. [147] concluded that effective data augmentation policies can improve the performance of CNN, especially if the data set is small. Taylor and Nitschke [146] concluded that data augmentation could improve the performance of CNN, whereby geometric methods out-performed photometric ones on their coarse-grained data.

2. Fundamentals

2.4.3. Text Recognition

Text recognition in the context of this thesis is the task of finding the position of written text in a picture and transforming it into a machine-readable string. Therefore, the first task also includes concepts from object detection (as introduced in section 2.4.2) to find where the text is positioned in the picture. Afterwards, the machine must understand the handwritten text in the picture and present it in a computer-processable way, e.g., by saving it in a string.

Often, the recognition of handwritten text is split into two categories, which are called online and offline [148]. Online means that trajectory information is available, as it is during writing, e.g., using smart pens. Offline, however, means that the handwritten text is statically recognised within taken pictures. For the context of this thesis, offline handwritten text recognition is meant if text recognition or handwritten text recognition is mentioned. This is because it better fits the overall goal of the thesis, which includes transforming tangible design objects into digital models.

Recognising handwritten text from a picture itself is an old but still actual problem in Computer Vision (CV) [149]. The topic is still important as there are many cases where it is helpful if a computer recognises handwritten text and can make it computer-processable. It is also difficult to solve based on the many different handwriting styles. However, approaches based on neural networks, for example, using Recurrent Neural Networks or convolutional neural network, and with massive datasets, can help to solve the problems [149, 150].

Most approaches can be separated into three steps: pre-processing, feature extraction, and classification [149] where the first step is focused on supporting the other two by segmenting and detecting the text and its part in the picture. Afterwards, the features are extracted from the identified texts, and last but not least, the features are used as input for the classification of the handwritten text to a computer-processable format.

There exist different ways to implement these steps, influencing the overall performance. However, similar to object detection (see section 2.4.2), this thesis aims not to create or find the best approach but to apply the general concept to utilise it for the thesis's objective. This is because this is still a domain where research and constant improvements are made. Additionally, the approach of this thesis is designed without relying on a specific one so that new achievements in the field can be integrated more easily.

2.4.4. Sentiment Analysis

This section provides a short introduction to the aim and possibilities of sentiment analysis as a basis for how it is used in the design of the result of this thesis. It does not aim to provide a sophisticated introduction, as sentiment analysis is not specifically adapted for this thesis, but the existing ideas and approaches are used as they are.

Sentiment analysis is a part of Natural Language Processing (NLP), focusing on analysing statements and assigning them a sentiment between positive or negative, or how the concrete sentiment classes are defined [151]. The statement on which the sentiment analysis is applied can vary from whole documents to paragraphs, sentences or phrases.

Sentiment analysis can be used to get an overview of a set of statements made about a topic from different stakeholders [152, 153]. For example, it is used to gather the opinions of people on social media on a specific topic. In such cases, the statements posted on social media platforms are collected and then analysed to assign each post a sentiment. Subsequently, one can derive the general opinion of many people on a specific topic.

Sentiment analysis thereby assigns a sentiment value, like negative, neutral or positive, to a provided statement [152, 153]. When applied to a more extensive set of statements on a specific topic, one can derive a tendency of the statements' creators towards the underlying topic. In this way, a quick overview can be generated without manually reading all the statements.

In this thesis, sentiment analysis is used as an approach to analyse feedback to models to support a quick evaluation of novel ideas encapsulated in digital conceptual models. How this can be used is discussed in more detail in section 3.4.2.

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

This chapter introduces the *Adaptive Modelling Environment*, which supports the ideation and design of innovative ideas through computer-aided conceptual modelling and enables the combination of tangible modelling techniques used in workshops with a digital and processable representation, creating digital twins of such tangible modelling techniques. The focus is set on what this environment must be capable of and how it can support such workshops. This core idea is visualised in Figure 3.1.

The tangible modelling methods will be based on design thinking (as introduced in section 2.1), already used in physical workshops to create innovative solutions. There exist multiple tangible techniques that can be used. The one that will be taken as an example for this section, the prototype implementation and the thesis as a whole, will be introduced in section 3.3. This technique is based on SAP ScenesTM (as introduced in section 2.1.2.3). In short, the technique uses cut-out paper figures placed on a table by the workshop participants to visualise their ideas. The paper figures use notations that are meaningful to the participants so that they can interpret them naturally.

The digital twin from design thinking will be created to represent the tangible objects, e.g., the paper figures, in a digital way, enrich them, and process them to support the workshop. Therefore, not only is a picture taken, but the individual objects must be identified and represented digitally to make them understandable to the machine. We will use conceptual modelling for the digital representation, as introduced in section 2.3. The physical objects will be represented through modelling objects. The needed context information for processing is encoded in the metamodel and adapted to the current domain and tangible method used in the workshop. The digital twin will be internally structured as a conceptual model, leading to the need for a modelling environment. This environment belongs to what we call *Adaptive Modelling Environment*. It includes the tangible and the digital components and the procedure for using them. This concept is visually represented in Figure 3.1. Below, we will discuss the different aspects needed for this idea.

One aspect is the tasks that must be accomplished within the *Adaptive Modelling Environment* and how they relate to the workshop. The most important tasks can be grouped in the time frame called *Runtime*. When the term runtime of the *Adaptive Modelling Environment* is used within the thesis, it corresponds to usage during a workshop and means both tangible and digital modelling. Therefore, with runtime, not the active time of the application is meant, but the overall workshop. *Preparation* contains all tasks

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

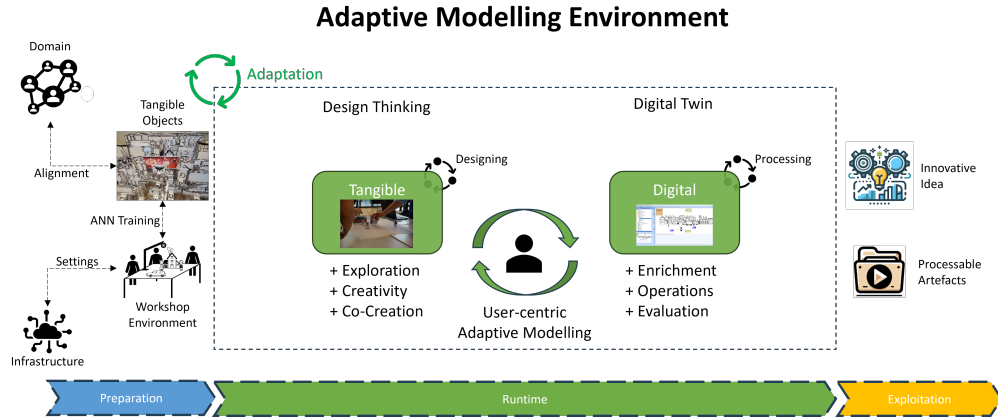


Figure 3.1.: The *Adaptive Modelling Environment* (the figure contains pictures from SAP Scenes™ [7])

done before a workshop, and *Exploitation* contains the workshop results, which can be further used for analysing or implementing the innovative idea.

The essential time frames are the runtime during the workshop and the exploitation afterwards. One of the environment’s central aspects is the adaptations applied during usage. Tangible and digital tools are used to represent innovative ideas. During the workshop, the understanding of the domain of the created idea increases and, therefore, is not fully known at the beginning of the workshop. This means that the tangible and digital parts must be adaptable to the dynamically changing domain. This is the foundation for user-centric adaptive modelling, which means that the methods used must always fit the users’ understanding of the domain and help them express and capture their thoughts.

Design thinking and tangible techniques thereby support a straightforward exploration of the domain, foster creativity and facilitate co-creation by allowing the users to visualise their ideas with semantic-rich representations. This enables an easier understanding of all participants who are engaged to provide their viewpoints. This is easily doable as the tangible objects can be quickly interacted with and adapted even more efficiently in parallel.

The digital twin of design thinking is used as a processable representation of the tangible model, supporting the workshop participants through enrichment and operations based on the digital twin. Based on the enriched information within the digital twin, the ideas created for the workshop can be evaluated based on the semantics and the results promoted to the workshop participants. The enriched information can include instance-level information, such as the person being 23 years old, and meta information, e.g., that a person possesses an age that influences their capabilities.

However, the tangible techniques must first be digitalised to use such enrichments and processing. Therefore, an automated way to transform from the tangible to the digital is needed. If this is not possible automatically, the usefulness is restricted as it puts

additional effort on the workshop participants. Within the workshop, it is crucial not to interrupt the flow of the discussion.

The environment and workshop must be prepared to use the adaptive modelling environment in a physical workshop. This includes the tangible method with the used physical objects, which must be aligned with the workshop's domain to allow the users easy interpretation and usage. In addition, the infrastructure of the workshop environment must be prepared, which means that it must be provided and configured. For example, a camera must be installed to capture the tangible objects and enable the environment to recognise them. One way to allow such a recognition would be to train an artificial neural network (ANN), see section 2.4.2.

Last but not least, after a workshop is finished, its results must be used in the next steps of the innovation project, like evaluation or starting its implementation. A workshop within the adaptive modelling environment has two results. One is the innovative idea available in the participants' minds, like in other design thinking workshops. Additionally, workshops with digital twins also offer processable artefacts as results, which are the conceptual models tailored and enriched for the specific workshop. These can then be further used to share and enhance the created design along site with the project's development.

During the runtime of the workshop, an important aspect is user-centric adaptive modelling, meaning that, on the one hand, it is essential to adapt the workshop and the used technique flexibly to the needs of the participants and possible changes in the perception of the domain [63]. For example, the physical means to represent the ideas must reflect the concepts from the domain to ease the understanding through experts [28, 37, 29]. On the other hand, the digital environment itself and its functionality must be adaptable to the specific needs of workshop participants, meaning that the operations executed upon the digital twin must support the need of the users to increase the value of the models [2, 34].

Based on the usage in the workshop, flexibility for the modelling environment means that the adaptations must be applicable during its usage and not to create an entirely new modelling tool, as it is often done with metamodeling platforms [2]. As creating modelling methods is a complex task, the participants in a workshop may not have the needed knowledge and, therefore, must be supported in applying the adaptations to allow for user-centric modelling. Here, a balance between supporting users with flexibility and not overwhelming them with complex functionality is needed, especially considering that the time in workshops is limited and the focus is set on creating an innovative idea rather than a highly sophisticated domain model.

Runtime adaptations to modelling methods, which we use as the fundamental structure for the design thinking digital twin, mean that the metamodel changes during usage. Therefore, the explicit metamodel in the digital environment must change according to the often implicit metamodel of the tangible technique used. The adaptations must consistently be applied to all the parts of the *Adaptive Modelling Environment* to keep all its parts usable.

Some general assumptions and prerequisites will be introduced before we go into more

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

detail about the *Adaptive Modelling Environment*. They provide influencing factors and insight into the context as basic assumptions for conceptualising the adaptive modelling environment.

The first assumption is that it is used for complex problems with no obvious solution and where novel ones must be designed. Additionally, the context of the problem has yet to be fully known, and therefore, it is uncertain which concepts will influence the solution. Such problems are also known as *wicked problems* [11], which need special handling methods, like design thinking.

This leads us to the subsequent assumption that one person alone cannot solve the complex problem alone. However, multiple stakeholders with different expertise must work together to find suitable solutions. They need common ground to communicate sufficiently with each other and explore novel and unknown solutions.

The following assumption is that organising physical workshops with the essential stakeholders is possible. Therefore, they can meet in a physical location for a longer period and communicate sufficiently to discuss ideas. Additionally, if too many stakeholders must be involved in the discussion (e.g., more than ten [17]), it should be possible to create smaller subgroups to keep the workshops efficient and engaging.

For adapting the functionality of the modelling environment, it is assumed that the needed functionality is implemented so that it can be loaded into the modelling environment in a plug-and-play fashion. Therefore, the workshop organiser can tailor the modelling environment to the needs of the workshop without the need for sophisticated knowledge in methods or software engineering.

For adapting the domain concepts needed in the workshop, it is assumed that the required figures are easily found so that the ideas of the workshop participants can be represented feasibly. Therefore, we do not detail how the fitting representation of the concept can be found, as many pictures can be found online.

To make the workshops useful, they must be sufficiently prepared. This means that the workshop participants can contribute to the solution, and the workshop organiser has prepared the method so the participants can work properly. All the technology is installed and ready, enough domain concepts are designed, and the organiser is experienced enough to support the participants during the workshop.

3.1. Digital Twin in the Context of Design Thinking

In this section, we will focus on the added value of the design thinking digital twin, which has a digital conceptual model at its core. Therefore, it contains not only the picture of the tangible model but also a digital representation of the used individual physical objects and related information, such as abstraction in the form of classes, characteristics, or relations. The digital representation contains information explicitly, which is implicitly available in the tangible model through human interpretation. In this way, the related information and the tangible model are processable by applications.

However, to improve the value of the digital twin of design thinking, we need a digital representation of a tangible model and a processing of the recorded information during and

3.1. Digital Twin in the Context of Design Thinking

after workshops. One fundamental operation is the automated transformation from the tangible to the digital model, which enables us to process the tangible model immediately without burdening the participants. If users had to manually create the digital model in addition to the tangible model, it would hinder the creative flow, and workshop participants would be more reluctant to use it. Having the processable digital representation of a tangible model instantly available can improve the workshop experience [4].

To enable sophisticated participation of workshop participants, only a few people should be included so that everyone can contribute and fruitful discussions can be held [17, 18]. However, limitations on the number of participants also limit the ideas and input that can be created. Here, a design thinking digital twin can support the participants by gathering input from other stakeholders, not in the workshop, or generated input, for example, based on AI.

One way workshop participants can be supported is by saving related information to the digital model. The more helpful information available, the better the *Adaptive Modelling Environment* can support the participants. Here, a digital model can be used to automatically suggest information that should be saved. For example, attributes of the represented objects can be searched in knowledge bases or the internet based on labels and a short domain description. These can be attributes proposed by the system and inset by the participants, or the system can also propose the attributes with values. Here, the participants should have the last say, but suggestions enable them to consider aspects outside their thinking patterns. Through such an approach, additional expertise can be integrated into the workshop without increasing the number of participants.

Similarly, relations between the objects can be suggested by a system, as having information about how the objects relate to each other is essential to know [21]. In the archetypical example, the paper figures positioned on the table are related, which is clear for the participant based on their discussion. For example, a smartphone next to a human suggests that the human interacts with the smartphone. However, this context information must be made available to the system to make it processable. Further, a system can suggest further relations to once again provide new input for the participants and, in this way, improve the overall innovative idea.

However, not only can generated information support workshop participants, but a design thinking digital twin enables real-time sharing with external experts, who can provide additional insights. The enriched context information in the digital models can be used to suggest questions or provide context text for the experts, improving their input for better understanding. The input provided by external experts can be processed and integrated into the model digital model to make it easier and more understandable for the participants.

Here, the goal is not to establish hybrid workshops, as the hybrid nature often hinders the flow of the workshop, based on problems with technology of feasible communication differ from physical or digital [63]. Therefore, the design thinking digital twin can support this by processing the workshop progress, preparing a summary for the external experts, sending them the information and providing the processed answers to participants.

The enriched information in the design thinking digital twin cannot only be used to

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

suggest additional information like relations or attributes to the participants. However, it can also support model consistency by finding constraint violations and prompting them to the participants, like marking a relation between two objects or an attribute value which does not fit the domain.

The design thinking digital twin provides added value during the workshop and improves the results, as it offers processable artefacts, which can also be used after the workshop. The enriched models contain the visual representation and its context information, which can be shared with further stakeholders and exploited in the further evaluation or implementation of the idea. As they are digital, conceptual models can be adapted to newly gained insights. Such adaptations can be made on the model and metamodel levels within the *Adaptive Modelling Environment*.

Suppose multiple workshops were held in the context of a common project. In that case, the digital models can be integrated based on the enriched information, automatically identifying similar concepts used multiple times or even parts that contradict each other. Based on the digital models, such integration can be automated to support the users.

Operations on the created models can also be used after the workshop, like transforming the high-level models created in the workshop to other conceptual models used to implement the idea. Here, the modelled information can be used to create at least a starting point for the other models to ease the users further.

3.2. Added Value of Tangible Modelling

This section will highlight the role of using tangible modelling within the *Adaptive Modelling Environment* and discuss why this thesis aims at combining tangible modelling with design thinking digital twins and does not replace the tangible means through a purely digital approach.

With tangible modelling in the context of this thesis, creating a tangible model in a workshop is meant like it is done in design thinking workshops. With a tangible model, we mean a representation of the idea through physical means, such as paper figures, sticky markers, or similar. The term tangible modelling is used instead of design thinking to emphasise the creation of the tangible artefact and not on other aspects of design thinking like the mindset or procedures. Such tangible visualisations are then used as core artefacts in the ideation, design and later implementation of innovative ideas.

However, innovative breakthroughs are not some spontaneous and lucky ideas, but need hard work and a creative human-centred way of working [9, 14], which cannot be done by one individual alone, but need multiple experts with various backgrounds working together. Even though technology has made progress in the last year, physical meetings can only partially be replaced, especially where participants in a project need to get to know each other and build a team [63]. Therefore, physical meetings and workshops improve cooperation between the participants. This is also because digital tools often only insufficiently show the whole spectrum of information about other participants like facial expressions and gestures [44]. This hinders communication and cooperation and calls for physical meetings.

3.3. A Tangible Design Thinking Technique Example

To enable the cooperation between the experts with their diverse backgrounds, not only the mindset of the participants is decisive, but also the physical environment providing enough space, opportunities for flexible working and needed tools and materials to visualise the ideas [14]. Visualising the idea enables the participants to share their thoughts clearly and further establish a common understanding among all the workshop participants. This visualisation's collective creation helps coordinate the thoughts and communicate on a level everyone understands.

Representing one's ideas quickly through rough visualisation helps to externalise the idea and make it easier to communicate with other stakeholders [60]. Especially if visual means are used, which improves understanding as visual perception is dominant among our senses. Therefore, using visual representations and encoding knowledge in spatial information eases comprehension through humans [96]. Such visualisation can also be achieved through sketching, but using physical objects allows one to quickly adapt their position and explore different variations of an idea easily. Using multiple physical objects allows different workshop participants to interact with them in parallel. Having the physical representation instantly supports a quick evaluation of the idea through early prototyping [14]. Based on the quick adaptation of the physical means, such prototypes can easily be adapted and re-evaluated.

For the visualisation in workshops, people prefer to use pen and paper approaches, like sticky notes, paper figures or flip charts, and consider digital means only if they feel like the physical counterpart and offer additional benefits [154]. For example, Smart Boards have the feeling of using flip charts. However, digital means of creating notes are often less smooth than sticky notes and pens. Additionally, such technical means often require complex and expensive hardware and software, hindering applicability.

Within a project, not each step must have or benefit from a physical workshop; for some cases, virtual cooperation is sufficient [63]. However, physical workshops are essential for crucial moments in innovation projects, especially if a new team should get to know each other if the underlying problem is not yet fully understood. In such cases, coming together physically and using creative techniques can facilitate this understanding. As physical and digital meetings are beneficial in different situations, this thesis aims to lay the foundation for using both together. However, this thesis does not focus on how modelling can be utilised in digital workshops, as this would go beyond the scope.

3.3. A Tangible Design Thinking Technique Example

This section introduces an archetypical example of a tangible design thinking technique as an example of a tangible modelling technique used in physical workshops and supported by computer-aided conceptual modelling. The goal of this section is not to provide a detailed description of how such an approach works but to provide an example referenced throughout the thesis to support the discussions on the introduced concepts. This example contains input to the physical and digital components used in and after a workshop.

For this example, a physical storyboarding technique based on SAP Scenes [7] will be taken and enhanced by a digital modelling tool. More information on storyboarding can

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

be found in section 2.1.2.3. In this technique, cut-out paper figures are positioned on a desk and are used to visually represent the story of users interacting with a product or a service. The figures show semantically rich pictures fitting to the domain of the workshop, which are easily interpretable by humans. Such icons are used to facilitate co-creation within the workshop participants. However, to enable this, the paper figures representing needed concepts from the domain must be prepared.

The figures on the table are used to visualise a story containing the key moments of a user interacting with the product or service. Each key moment is represented through a set of figures, representing the essential concepts and their relation to each other. The relation and interpretation are based on the visualisation of the individual paper figures and their relative position to each other. These key moments are then chronologically ordered to create the storyboard. However, the visualisation itself is essential to create knowledge and facilitate discussions within a workshop.

The physically positioned figures are then used to create digital models as the basis for the design thinking digital twin. The models are not mere pictures of the physical figures but enrich them with computer-processable understanding. Therefore, the digital twin captures the provided information and allows processing it in real-time to provide input for the workshop participants. For example, questions based on the domain and figures can be created through generative AI, or constraints can be checked based on the pre-defined semantics of the figures.

Therefore, each paper figure must be represented as its own object in the model and share the notation of the paper figure. Context is provided by assigning the used figure to classes and making implicit knowledge, available through human interpretation and discussion, explicitly available. For example, attributes and relations can be added to the digital modelling objects. Relations and attributes can be derived from the domain semantics of the concepts. In addition, relations can be derived from the position of the figures, as objects near each other may interact.

3.4. Adaptation at Runtime

This section will discuss what is meant by the adaptability of the modelling environment during runtime. The focus will be on the design thinking digital twin, the underlying modelling method and what should be adopted. This will include technical and conceptual aspects, meaning how the application must change or what conceptual aspects must be covered, like operations on the metamodel and the usage to make it useable. Lastly, aspects of the physical modelling, e.g., regarding the paper figures, will also be discussed.

Adaptation in the design thinking digital twin context can be separated into two aspects. One is the changing of the underlying meta-model and the modelling language. Therefore, the concepts that can be used to represent the domain are adapted. The second is to change the functionality, which can be used to process the model-based digital twin and offer additional value to users.

3.4.1. Adapt Domain Concepts

First, we focus on adapting the modelling method and the corresponding metamodel to enable an expressive representation of the idea created in the workshop. Therefore, the domain concepts can be adapted and tailored to the workshop, meaning that the available concepts, their hierarchy, attributes and relations can be newly added, or existing ones can be changed during the usage of the modelling tool in a workshop. This differs from most modelling methods, as all the concepts are usually pre-defined, and the users use them to describe their system. However, as in the design thinking workshop, the domain is not fully known, and a pre-defined approach can not be used. Hence, the possibility of changing the domain concepts is offered.

Therefore, workshop organisers do not have to fixate on the concepts in advance, so they do not need a sophisticated understanding of the domain before the workshop. This reduces the preparation time but adds additional tasks which must be handled during the workshop, including searching for fitting figures and preparing them to be used physically and digitally. Therefore, this not only influences the digital component but also requires additional equipment like a printer or pen and paper for the physical component.

The workshop can start with an empty set of concepts (physically and digitally) and simultaneously create the set and the models. This allows for the most freedom and puts the most workload on the workshop participants. Therefore, organisers can and should still prepare physical and digital figures and provide them to the participants. However, they must also consider that available figures will be changed and new figures will be added. Therefore, additional time is needed during the workshop, which must be anticipated in the planning. The figures must be based on and adapted to the workshop's topic, not that the available figures define the workshop's content. This calls for a balance between preparing enough figures to work properly and leaving enough room to add needed figures.

Adapting the domain concepts of a modelling method for a specific case can be similar to creating a new modelling method. However, creating useful modelling methods is a non-trivial task and requires experience and expert knowledge. As it cannot be assumed that the workshop participants are modelling methods engineers, the adaptation capabilities of the *Adaptive Modelling Environment* must be tailored to the tangible design thinking method and the workshop's participants. Therefore, the way the adaptations are made and how the phrasing for the functionality must be understandable by users in the workshop. Therefore, the adaptive modelling environment aims not to enable everyone to create sophisticated modelling methods but to enable the users to adapt the domain concepts based on their expertise. This can then be seen not as creating a class with attributes that can be instantiated. It can be seen as adding a graphical figure and providing a descriptive label and some characteristics.

Therefore, the modelling method should provide a basic structure that workshop users cannot change. Only the set of concepts representing the used figures should be adaptable. For example, the structure of the models, which have multiple scenes that form a storyboard and some basic characteristics to describe the figures, should be provided fixed, and only the concrete figures should be adaptable.

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

These adaptations should be possible during the runtime of the *Adaptive Modelling Environment*, therefore, during the usage in the workshop, without reimplementing the metamodel and redeploying the application.

The modelling environment should not only represent the models and save them but also provide processing capabilities to users to increase the model value [34, 2, 105]. These processing capabilities must then be considered during the adaptation, and the modelling environment must ensure that the needed processing capabilities are possible, even after adaptations are made or at least warn the users.

The adaptation of the modelling method and the concepts provided can be done not only during a workshop but also afterwards. The result of such workshops is processable artefacts which can be used and further adapted as the idea from the workshop is evaluated. Therefore, a need arises for the adapted modelling methods to be shared with other stakeholders.

Allowing the tailoring of domain concepts increases the expressiveness of models for humans. However, it restricts the domain-specificity of the mechanisms that can be provided as these can not be implemented in advance if the concepts are not known and cannot be implemented ad-hoc, as the workshop participants may not be expert method engineers. The mechanisms can only rely on the unadapted parts of the modelling method.

Knowledge is not purely assigned to the used figures but also to metadata assigned to them, the classes in modelling terms. Therefore, the creation of the classes should not merely add new classes but also allow the adoption of the class hierarchy. Classes should be able to inherit from each other to reuse already defined knowledge. Supportive capabilities should be offered to enable users to efficiently define the class hierarchy, like combining the objects of two classes or splitting classes into multiple ones.

Therefore, a method engineer is still needed to implement the basic structure of the modelling methods, which are not adaptable and prepare the bases for the adaptable domain concepts. This includes the adaptation mechanisms that may be adapted to the methods, further easing the usage in workshops. Here, the method engineer must also ensure mechanisms for consistency regarding the provided processing capabilities.

However, the adaptations must be applied within the digital modelling environment and reflected in the physical models, as the physical and digital parts must be coherent. Therefore, changes to either side must be made sufficiently to its counterparts. This needs an easy adaptation of the physical parts, usually provided by crafting materials and printers. The automated transformation from the tangible to the digital models must also adapt to changed concepts. Therefore, not only the concepts themselves must be adapted, but also the functionality of the modelling environment and the underlying modelling methods must be extended to fit the users' needs.

3.4.2. Extend the Modelling Method Capabilities

Above, we focused on the adaptation of domain concepts, and now we will discuss how the design thinking digital twin can be further enhanced to fit the needs of the workshop

participants. This is the next step in adapting domain concepts and focuses on extending the modelling method through new languages or processing capabilities.

Here, structural parts of the metamodel can also be changed, like integration of other modelling views (e.g., processes), references between different models, supportive concepts that can be utilised, or elements needed for the automated processing of models.

To put this in the context of the archetypical example (see section 3.3), different scenarios are possible, like linking the high-level representations of design thinking based models with existing or more specialised modelling languages. For example, the BPMN modelling language could be added to describe processes linked to the design thinking models. In this context, the workshop participants must know the added modelling language to make it worthwhile. To fully utilise such an extension during the runtime, the additional modelling languages must be added dynamically.

Additionally, a new concept can be added to the domain model, representing the figures, for linking them to the BPMN models. This way, the users can capture additional knowledge from the workshop and link it to their domain models. Establishing such links changes the structure of the modelling method and not only the direct domain concepts, as the linking concept does not necessarily represent a concept from the domain.

Another aspect of this scenario is that the structure of the modelling method can be adapted to increase its processing capabilities by allowing additional mechanisms to be added to the modelling method. A structural extension is necessary if the processing requires specific information to work properly. This could be certain concepts that must be available or that the domain concepts must possess specific attributes. Through this, the modelling method engineer does not have to foresee all possible usage scenarios and provide the functionality, as it can be dynamically loaded to the modelling method during its usage.

The goal is not only to adapt the domain modelling capabilities of a modelling method but also to extend it with additional functionality or modelling capabilities needed for a specific case. However, not by implementing each possibility in one enormous modelling tool and providing it as-it-is, but by allowing a straightforward extension of the modelling method for a specific usage scenario. This is done to avoid overwhelming the users by letting them search through all functionalities and modelling capabilities, as this can overwhelm them. Also, such extensions should not need the user to implement, integrate and deploy them. However, the modelling environment itself should support a dynamic on-the-fly extension. This goal is based on the idea of the on-the-fly computing approach (cf. [155, 156]), where applications are tailored to the user using automated and configurable deployment.

Users of the *Adaptive Modelling Environment* should not be elevated to modelling method engineers and forced to learn how to engineer complete modelling methods, as this task is complex and time-consuming even for experts. This scenario focuses on a modelling environment which allows novice users to integrate existing components of the modelling method to enable easy usage. Therefore, the presumption is made that a definition exists of the component or element users want to use and that it can be imported into the modelling environment. Then, the modelling environment must make

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

the necessary adaptations to the metamodel so that functionality can be used.

The on-the-fly extensions of the modelling environment also support the usage in the workshops, as the additional functionality or modelling extensions can easily be added and used. Then, the digital models can be automatically created from the physical figures and used by dynamically added extensions.

In the context of our prototypical example (see section 3.3), this means that the participants of a workshop cannot only adapt the domain concepts, which are used to visualise the storyboards but also add additional concepts that can be used to explain the scenes further. This could be relations between the concepts or supportive concepts not part of the modelling method. Examples of supportive concepts could be notes or aggregations, which allow the addition of textual context to models or visually group concepts.

Notes and aggregation do not build a complete list of supportive concepts. Depending on the specific workshop domain, additional ones can be useful. Additionally, there may be a need for concepts that correspond to a particular functionality that should be used in the workshop.

This could be functionality,

1. which needs additional supportive concepts,
2. which is agnostic to a specific metamodel and
3. those which need an adaptation of the metamodel to work.

The first case needs a specific concept, which can be added to the modelling method's core. Therefore, the core modelling method does not change by adding the functionality, but the users gain an additional feature.

For example, the workshop participants want to evaluate their ideas early by asking a broader audience. Therefore, the digital model should be posted on social media to gather feedback. To ensure that the broader audience better understands their idea, they import a noted concept and add context to the different scenes. The note concept is dynamically added for this purpose, which allows the addition of textual notes, which are visually shown within the model.

After they asked some people about feedback, they want to get an overview of the feedback, as within a workshop may not be enough time to analyse it all thoroughly. Here, the modelling environment can help by allowing the posting of the model and an automated analysis. One way to achieve an automated analysis of multiple texts regarding their general attitude towards a topic is *Sentiment Analysis*, which is often used to analyse posts in social media (cf. [153]). The results of the analysis should be saved and visualised in the model.

The idea of this functionality is visualised in Figure 3.2. In the figure, the functionality is separated into multiple conceptual parts, like the modelling tool, the publishing service and the sentiment analysis. In an implementation, these do not have to be separate entities. A more detailed description of this idea can be found in section 5.4.2

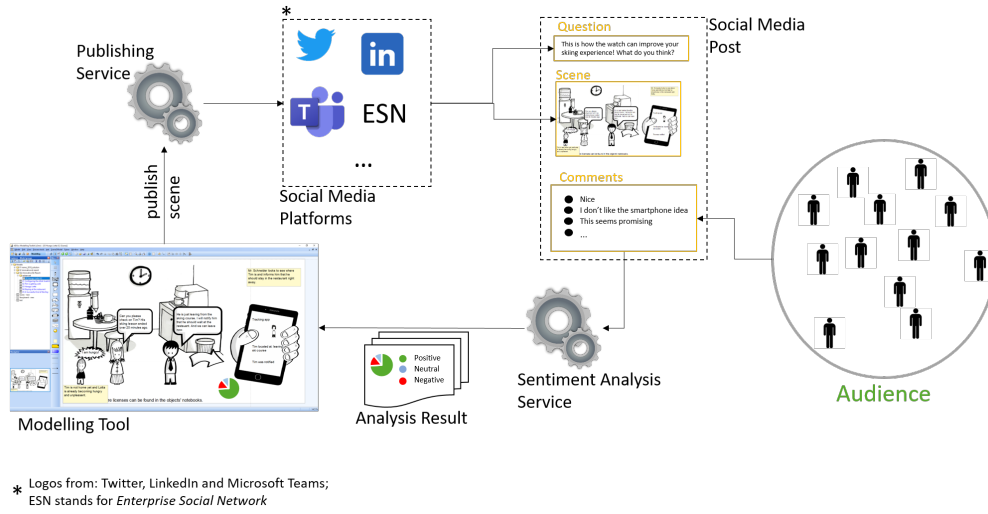


Figure 3.2.: Concept of directly posting models to social media platforms and retrieving analysed feedback (the figure contains pictures from SAP ScenesTM [7])

However, this example should not be understood as the only one for this scenario but only as an example to better understand its goal: functionality can be added during the usage of the modelling environment by simultaneously extending the modelling method. The example can be achieved by providing a script handling the sending of the model, gathering the feedback and a concept to save and visualise this. These two components (script and concept) can be dynamically loaded to the modelling environment and used without the need to implement or redeploy the modelling environment.

The second mentioned type of importable functionality is agnostic to a specific metamodel. This is similar to the *generic mechanisms & algorithms* introduced by Karagiannis and Kühn [2]. As generic mechanisms are independent of concrete metamodels, they can be integrated into all modelling methods implemented with a specific modelling environment and the provided meta² model, on which the generic mechanism is implemented.

An example of such a metamodel agnostic functionality would be the transformation of models into RDF graphs, as done by the RDFizer introduced by Buchmann et al. [157]. Such graphs can then be used as input for other applications, analysing the models. However, in this context, metamodel agnostic or generic means that the metamodel must not be adapted. It could be that the environment needs additional internal or external components, so a configuration is needed.

However, this does not only mean that an external definition of functionality must be added to the modelling tool but also that existing functionality can be adapted to better fit the user's needs. Therefore, generic functionality already implemented within the modelling tool should be adaptable to new requirements. This usually means that no

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling

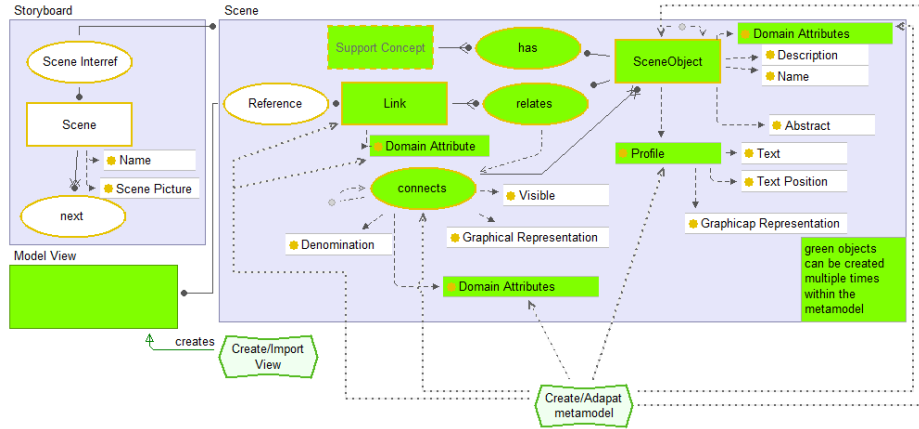


Figure 3.3.: Example metamodel for extending the modelling environment, which is an extension to the metamodel introduced in [4, 5] (visualisation done with CoChaCo, described in appendix A.4.13 or [6])

new concepts must be added, but existing ones must be adapted, which is then used by existing functionality.

An example could be cardinality constraints. The algorithms are usually implemented on the meta²-layer and must be configured to be applied to a specific modelling method. If the domain that should be represented is not yet known, providing constraints for the domain in advance is impossible. Therefore, the domain concepts must be adapted for use with constraints.

If the basic structure of the modelling method is adaptable, the user must be offered a set of elements that they can use to define their concepts. The basic set of available elements is provided in the meta²-model, and it can be used in the adaptation and extension mechanisms. For the archetypical example (see section 3.3), the meta²-model would need the elements of class, relation, attribute, view, link (or reference) and inheritance. Additionally, a way to define mechanisms is needed.

As the extension and adaptation mechanism allow for greater changes in the modelling method, these mechanisms must also ensure that the modelling method stays usable. This can be achieved by allying consistency checks on the meta-model to find and remove inconsistencies before they become a problem.

For extensions to the modelling method, the metamodel changes and the modelling environment itself. For example, the interface (e.g., a GUI) of the modelling environment can be influenced by extensions, which also must be considered. To allow the usage of dynamically added functionality, the modelling environment must allow the definition of triggers. Such triggers could be buttons in the GUI or after another functionality is executed.

Figure 3.3 shows an example metamodel which can be used for extending the modelling environment in the context of the archetypical example (see section 3.3). The green

3.5. Handling of an Adaptive Modelling Environment

objects can be adapted during runtime or provide the basis to derive additional elements. The ones in white are structural elements needed for the archetypical example.

The important aspect here is that generic versions are available for the parts of the modelling method that capture the domain concepts, relations, and attributes. These generic versions, like *SceneObject*, *connects* or *Domain Attributes*, cannot be directly used, but the usable domain concepts can be derived from them using inheritance. Creating and changing the domain concepts, relations, and attributes makes this approach adaptable. However, adaptability does not come from the metamodel alone; the adaptation mechanisms are defined based on the metamodel.

SceneObject is the generic element from which the domain concepts can be derived. It can possess multiple domain attributes or be connected to other domain concepts using relation. A domain concept can possess multiple visualisations, representing different concept variations using profiles. Each profile stands for one visualisation, which can have a graphical representation and text. The relations, shown through *connects*, can also possess domain attributes.

The *Link* concept represents objects in a design thinking model, here of one scene, which contains a link to other models providing *Model Views*. For example, a model view for BPMN is added, and to link it to concrete figures in the design thinking model, a link concept for processes is created.

A link can be added in the core model view (*scene* in the example metamodel) for each of the added model views. Here, multiple links to different model views can also be defined. The link concepts can also be enhanced with domain attributes to integrate the links more into the domain context.

The supportive concepts are not predefined for this scenario. Here, the users can extend the modelling method with the supportive concepts they need and not the ones a method engineer assumes they would need.

3.5. Handling of an Adaptive Modelling Environment

After discussing the concept of adaptability in this thesis in more detail, we will now focus on handling the *Adaptive Modelling Environment*. This means human users can interact with the modelling environment during a workshop by supporting their way of working and not having to adapt themselves to a new procedure.

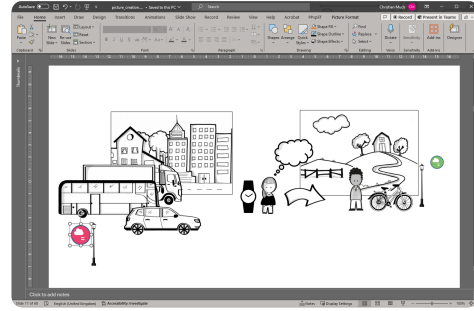
The handling includes the creation of models and the execution of mechanisms, especially for adapting and extending the modelling method. They should be intuitively usable for the user, which calls for user guidance using the modelling environment's functionality.

Supporting the users' way of working in a workshop means that they can still use tangible objects to create their physical models. This part should not be replaced by the modelling environment but facilitated through it. An example of using tangible models can be seen in Figure 3.4 "a". The automated transformation from physical objects to digital twins achieves this facilitation. In the *Adaptive Modelling Environment*, the tangible figures are then one way to interact with the modelling environment to create models.

3. Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling



a) Tangible modelling using paper figures from



b) Graphical user interface to create a digital representation of scenarios with Power Point

Figure 3.4.: Example possibilities for interacting with the adaptive modelling environment (the figure contains pictures from SAP ScenesTM [7] and pixabay.com)

However, this does not mean the only interaction should be over tangible objects. In addition, a GUI will be used to interact with the digital twin. This interface can trigger functionality and create and change models, similar to modelling tools. The tangible objects should not replace this interaction but enhance it.

An example for a GUI from Microsoft PowerPoint, which can be used to make digital pictures of tangible models, can be seen in Figure 3.4 “b”. However, this interface and tool is not a modelling tool and does not support an automated transformation or processing. It can only graphically represent digital figures. Therefore, a modelling tool will be designed in this thesis, not an application to draw pictures.

An example of using paper figures to create models can be seen within Figure 3.1.

Using only one GUI in a workshop, where one person interacts with the system, and the other participants watch and provide input, limit the interaction possibilities and participation, as only one person can interact with the GUI at a time. Therefore, using tangible objects enables participants to interact simultaneously and better work together. Additionally, the amount of figures which can be represented at once to a user is limited on a GUI, which would hinder the possibility of the users finding fitting figures. If the physical figures (e.g., made of paper) are presented on a table, it is easier for the workshop participants to get an overview of the available figures. Therefore, the interaction in finding fitting figures for the specific case is also adapted.

Another vital aspect for interacting with the *Adaptive Modelling Environment* is based on how the automated transformation is triggered. This can be done periodically or triggered by the users using a button in the user interface. If triggered periodically, the system must differentiate between the different scenes. This can be achieved through user input or by enabling the system to differentiate between scenes. However, periodically triggering transformation also saves the idea’s historical development.

However, interaction with the environment is necessary during the runtime but also in the preparation and exploitation phases. After a workshop, the processable artefacts in

3.5. Handling of an Adaptive Modelling Environment

the form of digital models can also be changed and processed. As this interaction is usually not based on a simultaneous interaction, only the GUI can be used here. For example, to enrich the models by adding additional information or triggering other mechanisms. The models can also be sent to other stakeholders and adapted by them using the modelling environment and its GUI.

Users must also interact with the *Adaptive Modelling Environment* to prepare a workshop. However, these users are usually not the participants of the workshop but the organisers. The preparation includes the modelling method and metamodel. A GUI is sufficient as not many people must interact with the system simultaneously. This preparation includes the modelling method and other components needed for the workshop, such as handling the transformation or taking a picture of the physical model as input for the transformation. However, all of them are done through interacting with physical components, like cameras or applications with an GUI.

With other components, we mean software which is needed in addition to the modelling tool to be usable in a workshop scenario. An example is the component which recognises the paper figures from the table and transforms them into a digital model. Such a component must use the same labels for the objects as the modelling tool. Section 5.5 describes such components in more detail.

During a workshop, the interface of the modelling environment must also offer ways to adapt the modelling method. On the one hand, this is adding or changing classes or relations and their attributes or adding needed functionality. The GUI must provide ways of interacting with such operations, e.g. by triggering through a menu entry. However, as the workshop participants are no method engineers, the interface must guide and support the users.

Such adaptation operation can either use saved information from the preparation phase or be added by the workshop participants. For the latter, the interface must also enable the users to provide the needed information. This can be done by providing the information to a GUI, like a form or a wizard, or the information is provided through a generic modelling object, which holds the needed information and can be used as input for the creation of a class [45].

A repository containing them must be available to reuse existing concepts or additional functionality. Domain concepts can be created during a workshop, but to create a whole new and integrated modelling method or implement functionality is too time and resource-consuming during a workshop. Therefore, to use them dynamically, they must be created during the preparation or loaded from a repository. Loading from a repository or activating new functionality must also be supported by the user interface, which will be influenced by it.

For example, if we think about the evaluation of models by posting it on social media and retrieving the feedback (introduce in section 3.4.2), the supportive concept which visualises the results of the evaluation must be dynamically added to the metamodel. This should not be done by the workshop participants manually, but the class should be imported together with the mechanism.

However, such automated adaptation to use mechanisms can also be applied to the

3. *Adaptive Modelling Environment: Creating Digital Twins for Tangible Modelling*

domain concepts. If a specific mechanism needs a particular attribute available to work, these attributes can be added automatically. Here, an additional step for the users may be needed to decide which of the available domain classes should be adapted. In this way, a semi-automated adaptation is used to prepare for the metamodel extension.

3.6. Summary

This chapter introduced the *Adaptive Modelling Environment*, what it encompasses and why tangible and digital modelling is supported. The main focus is set on the design thinking digital twin, which offers added value to physical workshops by mirroring tangible model artefacts, understanding them, enabling processing and usage within the runtime of a workshop, and supporting the exploitation of the results afterwards.

The underlying assumption is that physical workshops with tangible techniques and digital representation have value and that their benefit can be increased if they are combined to support workshop participants and do not put an additional workload on them.

The tangible modelling supports the co-creation and communication between workshop participants in a shared physical space, and the digital twin allows the inclusion of external experts or enables the provision of functionality that prompts help by utilising computation capabilities.

Ideas designed in a design thinking workshop are loosely defined during the preparation and the beginning of the workshop and only gather form during the workshop and its exploitation afterwards. This naturally leads to adaptation of the idea, as the understanding of the underlying idea and the goal changes during the workshop. Therefore, the *Adaptive Modelling Environment* must support the adaptation of its tangible and digital part in a way the workshop participants can understand and use. This adaptation in the *Adaptive Modelling Environment* focuses mainly on the domain concepts used as the core for describing and communicating the innovative idea. Therefore, the notation, concepts, and characteristics must all fit into the workshop domain, which means they must adapt to the user's needs.

Last but not least, digital artefacts can also be used after a workshop as they are processable and adaptable, even during the further evaluation or implementation of the workshop's idea. The digital model can be further enriched and processed to support this exploitation.

4. Requirements for the Adaptive Modelling Environment

This chapter discusses the requirements identified for this thesis's research project. The requirements represent capabilities that the adaptive modelling environment should be capable of to generate added value for a user. The focus is on requirements to allow a modelling environment to adapt to changing or unknown domains. This includes changes on the metamodel layer to adapt the modelling method. Here, the focus is set on adapting the notation of the modelling method and its usage within physical workshops, which is the focus of this thesis. Therefore, even though they are important, some general requirements for the software system are not explicitly mentioned, as this would go beyond the scope of this thesis.

An additional focus is on the notation of model objects and their characteristic within the modelling environment to facilitate the understanding of the human users. However, the requirements are not limited to the notation alone, as additional information is needed to represent the concepts and their classes and process information. The requirements will also be discussed in the context of how they can be applied if the modelling method can change.

In the following subsection, different requirements for a modelling environment, allowing the adaptation of its metamodel at runtime, will be introduced. Therefore, different aspects of where the requirements can be derived are discussed, and finally, the identified requirements will be distilled into a uniform set. The chapter starts with discussing areas that will influence the proposed approach to this thesis. Subsequently, requirements from the literature are introduced and last but not least, the requirements stemming from the introduced settings (cf. chapter 3) will be discussed. However, before the requirements for this thesis are introduced in more detail, background on requirements and how they are used in the context of this thesis will be provided.

The goal of requirements engineering and management is often to get a comprehensive understanding of what an information system should be capable of from the viewpoint of future users and other stakeholders [158]. However, this thesis does not aim to design a business-ready information system but will define requirements for a modelling environment that supports physical and digital modelling by also allowing an adaptation of its modelling method to better represent the system under study. Therefore, the defined requirements do not cover every aspect of a ready-to-use application, as this would be the goal in a business environment. This means the requirements will focus on the dynamic adaptation framework and contain thoughts and requirements for the implementation of it. However, as the requirements will not focus on the application alone, general requirements, which focus on application design, like the performance, will

4. Requirements for the Adaptive Modelling Environment

be omitted.

An example would be the requirement that models can be exported in a file under one second. Such a requirement could be necessary if a use-ready modelling environment should be implemented. However, it is not a key requirement of the introduced adaptive modelling approach in a scientific context. Nevertheless, parts of requirements engineering and management should be used as a guide to find and define the requirements.

First, there is not one right way to find requirements for all possible situations, but there are many different possibilities one can choose from [158]. In [158], multiple methods to find requirements are listed and rated regarding different scenarios where they can be applied and where not. From this list, the method of *use cases* seems fitting for this thesis, as it was rated helpful for systems with abstract requirements within an extensive system scope.

Many methods in [158] are based on gathering information from the stakeholders who want to use the new system. This thesis is a scientific work and not a software engineering project for a given set of stakeholders, so these methods do not fit well. This supports using use cases, which are analysed to refine the requirements. The contextual use cases are described in the introduced settings (cf. 3).

Further, as this thesis is a scientific work, the scientific literature is searched for requirements fitting this context. The literature will primarily be based on requirements for creating modelling methods and tools. The finding will be introduced in section 4.2.

4.1. Origin Areas

This section discusses viewpoints from which requirements can be derived. The focus here is not to directly derive the requirements but to introduce different viewpoints to find requirements for the introduced approach. The different foci chosen to analyse the requirements for the approach of this thesis will be discussed. So, the origin areas represent a viewpoint from which a requirement can originate and for what part of the system the fulfilled requirement should help to improve the value of the proposed approach of this thesis. The following three viewpoints were chosen as they are essential for defining useful modelling methods in uncertain environments.

4.1.1. Application Domain

As the proposed approach uses domain-specific conceptual modelling as a way to capture and represent knowledge, the domain which should be represented will influence the modelling method and, thereby, the modelling environment [28, 41]. The domain defines the system under study, which should be represented in the models, including the concepts that should be available to describe the system under study and the purpose for which the models are created. The purpose concerns the usage of the model or why the models should be created in the first place. For example, models are used to design software systems or execute organisational processes.

The domain specifies what should be represented within the models. The scope of the

chosen domain and the purpose of the domain can change from one modelling method to another [28, 29]. Even within the same domain, the scope can influence the available concepts. For example, there will be a difference in available concepts between the domain transportation in general and public transportation within a city. Moreover, even if the same concepts are used in this example domain, there will be a difference in the exact expression of the characteristics. For example, in the general transportation domain, it would be interesting to know how many containers a train can transport, which may not be as valuable within the public transportation domain.

The origin area of *application domain* focuses on the domain which should be represented in the models and what concepts and characteristics are needed. This is especially true if the modelled domain is prone to changes or if the use of models changes so that different aspects must be captured within a model. In both scenarios, the concepts in the models change as a whole, or their characteristics must adapt so that they can always represent what they should.

In the context of this work, this means that the changes within the application domain must also be reflected within the models. These changes influence the instance or meta-level. Instance level means that the changes to the model can be made through the existing modelling method. Therefore, the available concepts and characteristics are enough to adapt the model to represent the changed system under study again. An example is that a characteristic value of a modelled instance is changing, and therefore, the corresponding value in the model should also change. This would be comparable with changing the value of an object in object-oriented programming (OOP), but the class stays the same. Meta level means here that the change in the domain is fundamental and that the existing modelling method is insufficient. In this case, the model and the modelling method must evolve so that the changes in the system under study can be represented. This adaptation of the modelling method means that the metamodel must change, too.

For example, new information about an existing instance must be captured, and no fitting characteristic is available. In this case, the characteristic must be added to the concept (in the metamodel). Only then can the needed value be set for the instance.

If the concept in the background cannot change, the users are limited in what they can represent within the model. Having the ability to change metamodel is helpful in two circumstances. One is if the domain is changing fast and the domain-specific modelling method must adapt in parallel to stay valuable and expressive [28]. The other would be if the domain concepts which should be represented are not fully defined and still need some exploration. Adapting the metamodel in such a scenario can help to better understand the domain and the purpose of the model itself.

Changing the concepts to fit the application domain can also be dangerous, as some aspects may not be represented as initially designed. Especially if multiple users adapt the metamodel to their needs without discussing it with the other users. In this context, changing the metamodel can lead to conflicts and inconsistency, which a modelling environment should at least recognise.

The application domain influences not only the adaptation of existing modelling methods

4. Requirements for the Adaptive Modelling Environment

but also the creation of new ones, as it is done in the creation of modelling methods in general. Therefore, during the creation of the modelling method, the domain must be understood, and the useful aspects must be abstracted and integrated into the modelling method. However, as a domain-specific modelling method is created for usage through domain experts, the knowledge and procedures of the domain experts must be integrated. Therefore, the modelling method must support knowledge representation naturally for the domain experts, which leads to further requirements of the modelling environment and how the modelling methods must be capable of representing knowledge.

4.1.2. (User) Interaction

This origin area covers different aspects of working with models within a modelling environment. It encompasses two main areas. First, it contains the interaction of the user with the modelling tool to work with the models, and second, the interaction between the modelling tool and external applications. External applications in this context are information systems that do not directly belong to the modelling tool but interact with it to accomplish specific goals.

The term *working* with models was deliberately used, as the interaction can not only be used for creating models but also to trigger other functionality of the modelling environment. Therefore, the term *working* was chosen to encapsulate both meanings. Both introduced areas (interaction with users or interaction with external applications) can be used to work with the models. However, the modelling environment must provide all the possible interaction capabilities. The (user) interaction origin area aims at this interaction to derive requirements for the modelling environment. This can be technical and conceptual. Technical means the implementation must be capable of facilitating the interaction. However, the conceptual parts on what kind of interactions should be possible can also lead to requirements. So to say, what are the goals of the interaction is.

The first mentioned area is interaction with the user. Humans use modelling environments to create and process models to fulfil a specific need. The need depends on the domain and the user, as different people can have different reasons to interact with models. Therefore, a modelling environment must offer different interaction capabilities to support different goals. A modelling environment that allows adapting the modelling method must also offer different possibilities for the user to interact with the models so that users can always choose the one that best fits them.

This includes the graphical user interface of the modelling environment, which can be used to create models or trigger functionality. As there are usually different triggers for functionality, the modelling environment must offer some triggers and the mapping of wanted functionality. Examples of triggers include clicking buttons, clicking menu entries or triggering specific events. Such interfaces should not only support the manual creation of models but also allow for trigger functionality, facilitating the creation of valuable models. This support can be achieved by automated creation of parts of the models or by checking the syntactic correctness of models during interaction with users.

It is essential to consider user interaction, as creating models with drag-and-drop functionality is insufficient in all cases. Drag-and-drop means an overview of the available

concepts, which can be clicked and instantiated in the modelling area. Afterwards, all the needed information is added to the instance by filling out its characteristics. Creating models by hand through a GUI is an expensive and time-consuming task (cf. [36]) and not the best solution for every situation. Aditya Ghose, a professor researching modelling, stated it the following way in his lecture at the Nemo Summer School (nemo.omilab.org): “...conceptual modelling will die, but conceptual models will continue to deliver value...”. This statement is drastic, and no scientific study was found to reinforce this statement. However, it captures the essence (even if it is excessive) of a trend that can be understood. Creating models is time-consuming; therefore, new and easy ways to create them are needed, especially if people who are not modelling experts should use them.

This leads to new approaches to creating the models, as having the models themselves possess value [34, 36]. These approaches usually create the models automatically or at least starting points, which later only have to be refined. For example, [36] introduces an approach and tool that allows the creation of the first draft of process models (which the authors call *proto-models* out of existing models and documents to reduce the time-consuming task of modelling, for which many companies do not want to provide the resources.

The coupling of modelling environments with external applications can be used to automatically create (parts of) models or process the modelled information. The processing makes implicit modelled knowledge explicitly available or influences other systems based on the modelled knowledge. One example would be simulation, where information within the model is taken and used as input to calculate output variables, which provide further insight into the modelled system. This is an example of implicit knowledge, as the output variables are not provided through the modeller but are derived from the explicitly provided information. Of course, implementing the simulation algorithm must not be included in the modelling environment directly but can be used from an external application. In the latter case, the interaction between the modelling environment and the external simulation component must be possible.

Here, whether the interaction between the two components is provided automatically or manually does not matter. Automatically means that the modelling tool can directly send information to the external application and process its answer, and manually means that the model is exported (e.g., into a file) and manually imported into the external application. Both cases can provide requirements for a modelling environment, as the interaction format between the two applications must be defined, as well as the interaction procedure.

4.1.3. Modelling Method

Lastly, requirements can be derived from modelling methods, which the modelling environment should support. As modelling methods are created to generate value for their users, they are created with a goal in mind, which also provides requirements for the modelling tools or environments. The goals of and reasons for modelling methods can be manifold. Therefore, it is not feasible to aim to identify requirements for every possible modelling method.

4. Requirements for the Adaptive Modelling Environment

In this thesis, the focus is set on a model-based innovation space, as discussed in [54]. Modelling in such an innovation space will also influence the requirements discussed in this thesis. Here are insights and experience from the OMiLAB@UNIVIE, which is the OMiLAB node located in the research group *Knowledge Engineering*, of the *Faculty of Computer Science* within the *University of Vienna*.

Within the OMiLAB@UNIVIE, we followed the three-layered approach of *OMiLAB Digital Innovation Environment* (cf. [159, 54]) and analysed how modelling can support such an innovation approach. The goal is for conceptual modelling to capture knowledge in ideating, designing, and evaluating innovative ideas. However, not in the sense that a generic modelling method is created, which can be used for all possible cases, but that conceptual modelling is applied. It is tailored to the domain to increase its understandability. The advantage of domain-specific conceptual modelling should be utilised, and tailored languages should be created to allow the users a more intuitive understanding. From the three-layered OMiLAB approach, this thesis focuses on the top layer.

The top layer focuses on high-level abstraction of the scenario and creates novel business models or innovative ideas [54, 159]. Therefore, creativity and co-creation are needed so the problem can be understood and a useful solution can be ideated. This can be supported using physical design thinking workshops, as introduced in section 2.1. Therefore, the modelling method and its modelling environment must support the working in the physical workshops.

Many creative methods use tangible means to foster participation in such workshops. Therefore, these methods provide additional requirements for the modelling environment, as their basic structure must be supported. Based on the chosen method, the modelling environment should support users and ease the workload.

The bottom and middle layers of the OMiLAB three-layered approach are not the focus of this thesis, but they provide input to the modelling method used in the top layer. This is because it provides input for the domain concepts that should be represented, and a way is needed, be it technical or organisational, so that the information from the top layer can be used in the models from the other two layers. Exchanging information between the layers is an important requirement for the modelling tool.

4.2. Requirements for Adaptive Modelling through a Literature Review

In this section, requirements from the literature will be described. The goal is to find requirements for modelling environments that allow an adaptation of modelling methods during usage, focusing on allowing novel ways to interact with this modelling environment. This was done to guide the adaptive modelling part of the thesis. The literature should provide insights on how modelling methods can be created. However, the difference between using and adapting it must be considered.

Therefore, scientific literature was searched, which focused on creating the conceptual modelling method and what requirements a modelling method must fulfil. These could be

4.2. Requirements for Adaptive Modelling through a Literature Review

methodologies for the creation, guidelines, best practices, or something similar. Thereby, most come from approaches where modelling methods were defined before use. If found, approaches that adapt it during the usage were also considered. Last but not least, the search was focused on requirements influencing modelling environments. The identified requirements will then be used as input for the design of the modelling environment.

For the literature review, a systematic approach was chosen [56] to get an overview of the state of the art and how modelling methods and their tools are created. The conducted literature review is based on the general procedure discussed in section 1.3.3.

4.2.1. Literature Review Procedure

This subsection contains a description of how the literature review was conducted. The databases and platforms will be introduced, along with the search terms and inclusion criteria. Last but not least, an overview of the found literature will be provided before the findings from the literature review are presented in the following subsection.

To conduct a useful literature review, a question must be found that is worth answering. For this literature review, the guiding question is:

How can diagrammatic modelling methods be adapted/created, and which requirements must they fulfil to increase the model value?

The scientific literature is expected to find guidelines, rules, or requirements that must be fulfilled to create useful modelling methods and, consequently, allow for the creation of valuable models. So, even though the provided question is generic for some creation of modelling methods, the literature review will focus on guidelines for creating modelling methods, focusing on diagrammatic ones.

The focus is set on how the metamodel and its notation can be tailored to the dynamic needs of users during usage. These requirements should not only target functionality triggered by users directly but also how the modelling environment can update the metamodel, which may be included in other functionality. For example, suppose the adaptive modelling method is used in a design thinking workshop. In that case, the automated creation of models from figures can add them to the metamodel of the modelling method. Enhancing algorithms may also add attributes to the classes. Such cases are not directly triggered by users but applied during usage.

For the literature review itself, the following databases and platforms were used. These were chosen as they are well-known and used in computer science research.

1. SpringerLink
2. Elsevier Science Direct
3. ACM Digital Library
4. IEEE
5. Web of Science

4. Requirements for the Adaptive Modelling Environment

As search terms, different words for requirements or guidelines were combined with different words for modelling methods. Different combinations were first tried to find the right ones, but the terms listed below were used. Most of the terms focus on the conceptual creation and definition of modelling methods. Nonetheless, two phrases were added to find requirements for metamodeling platforms. The following list gives an overview of the used search terms:

- “design principles” AND “modelling methods”
- “design principles” AND “modelling languages”
- “design requirements” AND “modelling methods”
- “design requirements” AND “modelling languages”
- “design principles” AND “modelling methods”
- “design principles” AND “modelling languages”
- “designing” AND “metamodeling platforms”
- “requirements” AND “metamodeling platforms”

In the list, the quotation marks represent the terms that should be searched together, and the *AND* should be understood as logical *and* so that only publications are found that fit both parts. Therefore, the exact search term can vary between the different platforms, as they use different syntaxes and interfaces to perform the search.

Further, the spelling of *modelling* and *metamodeling* was adapted to the different platforms, as some of them did not make any difference if the terms were written with *ll* (double l) or *l* (single l) and others did. Therefore, the writing of the terms was also adapted to fit the platform.

Additionally, the searches were restricted to articles and conference papers from the discipline of computer science starting in 2010. This should enhance the quality of the results and consider more recent publications. As conceptual modelling is established in the field of computer science, restricting the search to this area improved the results.

Table 4.1 provides an overview of the publications found based on the provided search terms.

This leads to a total of 2.733 publications found, including duplicates. Afterwards, the titles and abstracts of the publications were screened to narrow down the list further. If the title or abstract indicated that the publication deals with defining, creating or adapting modelling methods, it was chosen for further investigation. Otherwise, it was removed from the list.

4.2. Requirements for Adaptive Modelling through a Literature Review

\Database Search terms	Springer- Link	Elsevier Science Direct	ACM Di- gital Lib- rary	IEEE Xplore	Web of Science
“design principles” AND “modelling methods”	194	220	55	23	37
“design principles” AND “modelling languages”	406	356	107	28	27
“design requirements” AND “modelling methods”	93	43	67	24	79
“design requirements” AND “modelling lan- guages”	152	109	49	24	117
“design principles” AND “modelling methods”	61	107	20	8	13
“design principles” AND “modelling languages”	116	20	33	10	20
“designing” AND “metamodelling plat- forms”	34	3	5	6	2
“requirements” AND “metamodelling platforms”	32	13	6	13	1

Table 4.1.: List of the overall found publications in the given platforms based on the provided search terms

4. Requirements for the Adaptive Modelling Environment

Based on the title and abstraction, the following exclusion criteria were applied: Some of them could be evaluated through the filters of the used platforms, and others were done by reading the title and abstract.

List of exclusion criteria:

- publications not written in English or German
- requirements only for the domain of modelling languages/methods and not on languages/methods in general
- redundant papers or duplicates
- paper that seems to just introduce one modelling method for a domain and not consider modelling method design as a whole

The result of this step is a list of 154 publications (including duplicates). Table 4.2 gives an overview of their distribution to the different platforms:

After this step, duplicates were removed, and the publications were screened more thoroughly, which means that their introduction and conclusion were read and evaluated for how well they fit the goal of the literature review. If they were not fitting, they were removed and read entirely otherwise.

The quality or ranking of the found publication itself is not a knock-out criterion but was considered in the analysis and consolidation of the findings. This means the literature review will consider publications with a low citation count not published in top-ranked journals or conferences. However, their quality will influence their impact on the review's conclusion.

After all duplicates were removed, 107 publications were still on the list of possible interesting literature. For those 107 publications, the introduction and conclusion were read, and if they seemed fitting, the whole paper was read. To this, 107 publications, 19 additional were added. Those were either found in the references from the 107 publications or suggested by colleagues. They can come from a prior search for the fundamentals or recommendations from a supervisor or other knowledgeable colleagues. Therefore, 126 publications were read more thoroughly. In this way, 33 papers were considered in the literature review, from which requirements were derived.

4.2.2. Requirements from the Literature Review

This section will introduce the requirements identified in the literature review, described in section 4.2.1. The introduced requirements will be consolidated. If requirements were identified in different sources focusing on the same aspects of modelling method creation, they are grouped, even if their authors used different names. Therefore, the denomination of the identified requirements is adapted to fit this thesis. This was important as some requirements had similar descriptions but different names, and some

4.2. Requirements for Adaptive Modelling through a Literature Review

\Database Search terms	Springer- Link	Elsevier Science Direct	ACM Di- gital Lib- rary	IEEE Xplore	Web of Science
“design principles” AND “modelling methods”	4	8	1	0	3
“design principles” AND “modelling languages”	14	11	7	0	4
“design requirements” AND “modelling methods”	2	2	9	0	3
“design requirements” AND “modelling lan- guages”	2	5	3	2	7
“design principles” AND “modelling methods”	5	14	1	0	2
“design principles” AND “modelling languages”	8	2	6	1	4
“designing” AND “metamodelling plat- forms”	5	0	2	3	1
“requirements” AND “metamodelling platforms”	1	5	3	4	0

Table 4.2.: List of publications after the screening of the titles and abstracts

4. Requirements for the Adaptive Modelling Environment

requirements had similar names but different descriptions. The consolidation was done to have an understandable denomination within this thesis. The order in which they are described does not reflect their importance, and here are the identified requirements introduced. This does not mean that all of them are used to design the adaptive modelling framework.

Modelling methods should aim at being **ontological complete** and **ontological clear** [29, 94, 20]. The first means that all the required concepts from the domain are abstracted into the metamodel of the modelling method. The second one means that singular concepts from the metamodel do not map to multiple concepts from the domain, and therefore, there are no two concepts from the metamodel that map to the same concept of the domain. Additionally, there should not be a concept in the metamodel that does not map to a concept of the domain. Which concepts are abstracted into the modelling method depends on the users and their goals [160]

However, identifying concepts in a domain is insufficient, as this knowledge must also be captured and described comprehensibly [161]. This can be achieved in different ways and degrees of formality, which can be applied to achieve different goals. Informal approaches can be beneficial for exploring and understanding a domain. However, if a domain should be rigorously described to enable communication or processing, then formal approaches are needed. Therefore, a dedicated definition language (e.g., metamodel languages [31]) should be used, which is understandable by the involved stakeholders.

The following identified requirement is called **adaptability**. Adaptability of modelling methods is important to keep up with changing domains [35, 28, 38, 162, 163, 19]. If the domain changes, the modelling method's concepts should also change. Therefore, the metamodel should be adaptable to evolve along with the domain. Such adaptations also ensure that the concepts of a modelling language are those with which users are familiar [29].

Creating or adapting a metamodel is a complex task often done by experts. To achieve usable adaptability, not the whole approach of metamodeling should be necessary for the modeller, but only selected parts should be adaptable, which makes adaptations more intuitive for modellers [38]. Additionally, the adaptability of a modelling method does not only apply to the metamodel directly but is also related (e.g., the definition of the notation) [162]. Therefore, the application of adaptability must also consider that the adapted metamodel and additional artefacts can still be used together and with already instantiated models.

The needed adaptations should be defined in basic operations and applied to the metamodel and the additional artefacts. Such basic operations for adaptation can be inspired by CRUD (= create, read, update and delete) operations [162]. Whereby operations may add something to the metamodel (create), change existing parts of the metamodel (update) or some existing parts are removed (delete).

However, not only the modelling methods must evolve alongside the domain, but also a modelling environment must be adapted to fit the needs of a changed domain or modelling method [34, 28]. For a modelling environment, this resembles a **fast prototyping** approach, where changes to the environment and the method should be easy and quick

4.2. Requirements for Adaptive Modelling through a Literature Review

to implement so that a prototype is quickly available for testing. Fast prototyping can also help create new modelling methods by creating them iteratively, which supports the evaluation during the creation [45].

In some cases, adapting existing concepts to keep the modelling method relevant is insufficient. However, the modelling method itself must be extended, and additional concepts must be introduced [28, 163, 164, 19]. We call this requirement **extendability**, and it can be triggered by changes in the system under study or the need for new functionality within the modelling method and its domain.

Extendability can be directly integrated into the modelling method or applied externally [163]. [29] argues that integrating the extension capability allows a more useful application. Nevertheless, how they are applied, they must influence the modelling method on the suitable abstraction layer [164, 163] and be supported by the modelling tool to fully utilise the extensions [163, 19]. Therefore, whether the extensions come from the same or different domains does not matter. It is essential that they are applied coherently and do not violate design guidelines. Braun and Esswein [164] argue further that extensions should be additive to the metamodel and should not change the metamodel itself so that the validity of the existing metamodel is not jeopardised.

Another important requirement that was identified is **consistency** [163, 161, 165, 19, 41]. The aspects of a modelling method that should be consistent are defined in constraints, which are based on the metamodel to capture domain knowledge which is not directly representable through the metamodel [161].

Consistency itself can be considered in different parts of the modelling method. Consistency is needed between models and their metamodel and between models and the purpose of the modelling method [41]. In a multi-view modelling approach, consistency is also crucial between different models [165]. Additionally, consistency of the visualisation of the modelling method is essential, especially if multiple modelling views are used [41]. This creates a common look and feel, supporting the modelling method's coherence. The modelling method and its tool must support consistency.

Allowing the users of modelling tools to adapt the modelling method to their specific needs under certain circumstances can increase the understandability and foster creativity, but also makes the models hard to understand for not involved stakeholders [19]. In this regard, a modelling tool can help by allowing the flexible adaptation of the modelling method and providing functionality that supports the consistency of the models. Having such consistency within the model eases the cognitive load of understanding them.

The following requirement focuses on the drawback of a modelling method covering too large a domain and offering too many concepts. In such modelling methods, the models can become too complicated to handle and understand by the users [163, 94, 20, 106]. Having too much information available can also come from adapting modelling methods and adding too much information. Therefore, it is essential to keep **simplicity** in mind when creating and adapting modelling methods.

Only those concepts from the domain must be integrated into the modelling method, which are needed in the model to describe the domain [41]. Therefore, the concepts used must be harmonised with the purpose of the language. This leads to a balancing act

4. Requirements for the Adaptive Modelling Environment

between keeping the modelling method simple enough to support comprehensibility and complex enough to capture needed aspects to fulfil its purpose.

Therefore, a modelling method should be tailored to a specific purpose [41, 20]. It should not contain all possible concepts, only those needed for concrete usage scenarios. The concepts should be kept to a minimum, and redundancy between different concepts should be avoided. Simplicity can also be supported by structuring the modelling method into different sub-methods or views.

The following identified requirement says that users of a modelling method and its tools should be able to use them over a more extended period so that the generated value exceeds the invested effort of creating the modelling method and tool [94]. We call this requirement **longevity**, and it influences not only the design of the modelling method but also its modelling environment. The modelling method must stay relevant to its domain and purpose, which can be achieved by adapting it. Longevity of the modelling environment focuses on updates to the environment and the available functionality.

To holistically represent a system under study, often different viewpoints must be taken into consideration, and such viewpoints should also be enabled by the modelling method [166, 37]. Viewpoints stand here for different focuses or lenses under which a system can be analysed, e.g., the focus can be set on processes or the IT of an enterprise. Each viewpoint can thereby be represented by its own models with their own concepts. However, to get a holistic view, a user must be enabled to link different viewpoint models together. Through these links, the users see where they are connected, and further, such links can be exploited when the models are automatically processed. We call the requirement for supporting multiple viewpoints **multi view support**.

The different views can be aligned horizontally or vertically in perspective to their level of abstraction [166]. Horizontally means that the different viewpoints are on the same abstraction layer but with a different focus. Here, the example with the processes and IT architecture (which was mentioned before) fits. Vertically means that the different viewpoints are on different abstraction layers. An example would be a high-level process map linked to concrete-modelled business models.

Even though supporting multiple views in modelling allows for a more sophisticated knowledge representation, it also brings additional complexity, as the different views must be consistent with each other [167]. Therefore, a modelling environment supporting multi-view modelling should also support the definition and evaluation of consistency between the different views. Therefore, **consistency** is identified as one of the requirements.

However, consistency is not only crucial in a multi-view modelling approach, as the limitation and specification of possible models can also improve the semantics of a modelling method [161]. Such additional knowledge cannot always be described with the metamodel itself but with constraints defined on top of it. Therefore, the consistency requirement can be separated into **multi view consistency** and **intra model consistency**.

The next aspect of consistency, which should be discussed here, will be called **adaptation consistency**. If the modelling method adapts over time, the adaptations must be consistent within the modelling method and the modelling environment [162]. Therefore, the adaptation operations must be defined so that the modelling environment can work

with the adapted concepts and the changes in the metamodel or additional artefacts stay consistent with existing models. Lastly, the metamodel and its additional artefacts should stay consistent.

The following requirement will be called **comprehensible notation** and will focus on the notation of the modelling method, as it is an important aspect of diagrammatic conceptual modelling methods. We want to add a sub-requirement of **adaptability**, which we will call **adaptability of notation**. This is important so that the notation of the modelling objects can better fit the specific domain, making it easier for domain experts to comprehend using the modelling method [163]. Adaptability of notation means that the visualisation of an object can be changed to tailor it to the users' needs. The models' notation can range from abstract line-and-box diagrams to visualisation, which uses more complex and sophisticated graphics [168].

The adaptability of notation not only means that the visualisation of the class with all its instances is changed but also that a specific visualisation of an object in a concrete situation can be helpful. Such a visualisation is called **secondary notation** by [169] and will be considered a sub-requirement to **comprehensible notation**. This specific information, which should be visualised, can be based on the object itself, the object in the context of the model, or other situations. However, such adaptations must not include a completely new visualisation of the object; they can change aspects of the existing visualisation, like a different colour.

As the visual representation of diagrammatic models is very important, sufficient time should be spent on designing the overall visual structure of the modelling method and the models which could be created with it [40]. To achieve this, Moody [40] introduces the following nine principles, which can help to define a useful notation of a modelling method. We will mention the principles here, as they can be used to derive requirements for the modelling environment. The principles introduced by Moody [40] are:

1. Principle of Semiotic Clarity
2. Principle of Perceptual Discriminability
3. Principle of Semantic Transparency
4. Principle of Complexity Management
5. Principle of Cognitive Integration
6. Principle of Visual Expressiveness
7. Principle of Graphic Economy
8. Principle of Cognitive Fit

The first two principles state that notations should not represent multiple concepts and that the different notations should be distinguishable ([41] also states the importance of distinguishability). These principles are nothing that a modelling environment can directly

4. Requirements for the Adaptive Modelling Environment

support, as these are important design decisions. However, the modelling environment must enable the users to change and define their wanted notation, which belongs to the previously defined **adaptability of notation** requirement.

The *Semantic Transparency* principle says that the visualisation of a concept should hint at its meaning [170, 40, 41]. This is especially true if people who are not experts in the modelling language should use it. Expert users who have learned the modelling method already can also interpret diagrams with abstract symbols as visualisation. However, novice users often struggle if the visualisation does not hint at the meaning. For example, BPMN uses diamonds for decisions, which is easily understood by experts in the language. However, for novices, this is not intuitive.

On the other hand, UML uses stick figures to represent users in the use case diagrams. The visualisation through stick figures hints at the meaning of the concept. Therefore, a group of users should be able to adapt the notation of the modelling method to support their understanding.

The fourth principle says that diagrams can be hard to understand if they are too complex, usually because there are too many objects in the diagram [40]. To tackle this problem, ways of managing this complexity are needed, which must fit syntactically and semantically into the modelling method. One way to achieve this is by separating one complex model into multiple smaller parts. This is connected to the before introduced **multi view support** support. This also supports the understandability of models by not overloading them [171, 40]. Having too many objects in one model hinders the users from understanding them, which can be tackled if a model is separated into different diagrams.

To further support the understanding of models, graphical and textual encoding can be combined to create synergies (*Dual Coding* principle of [40]). For a modelling environment, this means that it should support the requirement of **dynamic notation**. This means that the information saved in the modelling objects should be able to be visualised with the notation of the object. For example, show the text of an attribute next to the object.

The eighth principle says that a modelling method should not contain too many concepts with their own visualisation, as they can overload a user, especially if they are novices [40]. Keeping the visual concepts to a minimum means that the modelling method should lose these concepts and that the information could be encoded in another way and then shown graphically on the model. Through the adaptability requirement, users can add and remove concepts as needed and, therefore, are enabled to keep the concepts to a minimum. However, the requirement can also support this principle, which we call **filtered view**. This means that if users use a predefined set of concepts in a modelling method, they should be able to change what is shown to them without directly removing them from the modelling method.

The visualisation of the modelling method should not be seen in one setting, e.g., used with a computer-based modelling environment, but models can also be created with pen and paper [40, 20, 22]. This leads to the requirement, which we call **different interaction capabilities**, which means that users should be allowed to use different ways to create the digital models, like using the GUI, haptic elements or their voice. This requirement will be subsumed under the **usability** requirement.

4.2. Requirements for Adaptive Modelling through a Literature Review

Modelling methods already exist which represent concepts in a certain way and are understood by domain experts [41]. This familiarity can be exploited in the modelling method's notation. This leads to our requirement of **reuseable notation**. The reused notation must not be overloaded with entirely new meanings, which can lead to misinterpretations. The reusable notation will be subsumed under the **comprehensible notation** requirement here.

Regarding notation, we want to mention syntactic sugar. Even though syntactic sugar, parts of the language that do not contribute directly to the purpose of the language but improve the understandability, can be helpful but should be used with caution [41]. However, too much syntactical sugar can overload the language and increase its complexity.

However, there is not necessarily a one-to-one relation between the metamodel and the notation (aka. the abstract and concrete syntax), as they can also be mapped in an n-to-n fashion [172]. If a user should be able to map them or re-map them, then both need to be defined in a usable manner [173] so that humans and machines can process them.

To improve the utilisation of the models, a modelling environment and the modelling method should support the representation with **multiple concrete syntaxes** [107, 37, 160], which is our following requirement. This means the creation and adaptation of a modelling method should focus on its abstract and concrete syntaxes. Supporting multiple concrete syntaxes also supports easier reuse of models for different scenarios. For example, a modelling environment can support a diagrammatic concrete syntax for human modellers, a graph-based one for internal processing, a XML for sharing with other applications or a RDF export to enable semantic reasoning. As this requirement focuses on how the knowledge in the model can be represented, it will be subsumed in the **comprehensible notation** requirement.

A wide range of modelling methods exist, focusing on different domains created by experts in their domain. The following requirement calls that users should be able to integrate existing modelling methods into their currently used one to enhance the capability of the modelling method [28, 94, 19, 20, 160]. This can be necessary if novel aspects are discovered in a domain, which should be captured, but the used modelling method is not conceptualised for these aspects. However, instead of creating the metamodel for these new aspects from scratch, this requirement states that it should be possible to integrate existing modelling methods or parts of them in the used modelling method. This requirement is called **integrability**.

If this approach is expanded, the basic notation can be used to create modelling methods by reusing (parts of) existing modelling methods [174, 35, 41, 39]. This is so that already created parts can be reused and must not be built from scratch. The ability to adapt reused concepts to the specific case further supports the creation of modelling methods. This leads to the following requirement, which we will call **reusability**.

Reusability is closely related to integrability, but the first one focuses on how modelling methods are defined so they can be reused. The second one focuses on the ability of a modelling method to interpret definitions and load them to an existing modelling method. Further reusability focuses on creating new modelling methods or new versions. Allowing

4. Requirements for the Adaptive Modelling Environment

to build a new modelling method out of existing parts can ease the creation of the method and the implementation of its tool [174, 41].

A repository of existing modelling methods can be used to achieve reusability, where available modelling methods can be stored [174, 28]. To reuse modelling methods from such a library, the languages must be modularised [160].

Modelling methods should support the representation of systems under study of different sizes without losing expressiveness [94, 20]. This characteristic of a modelling method is captured in the requirement of **scalability**. This can be done either through the structure of one model, which enables the usage of larger and smaller models, or through a multi-view approach, which allows the separation of large models into smaller ones. Additionally, scalability can be supported by dynamic hiding and showing certain concepts to match the model to specific circumstances [20]. A modelling environment should, therefore, possess different approaches to support scalability.

Modelling methods can be used in different scenarios, from rigorously describing a system under study to capturing knowledge more freely and intuitively [37]. Many modelling methods focus on the first scenario, but in many aspects, it is also beneficial to gain the user flexibility to allow the user to describe what they need. In this thesis, we call this requirement **flexibility**.

The rigorous definition of modelling methods also influences their semantic formality. It ranges from very formal (e.g., computer understandable) to low formality (e.g., interpretation through humans) [37]. Formal languages often follow a rigorous definition and a detailed model of the system under study. Models with low formality have more relaxed rules but leave room for interpretation. Both aspects have the benefits and drawbacks.

Flexibility also influences modelling procedures (as introduced by [2]). Here, users must receive help and are guided to create useful models if they want to. However, the users should not be restricted and should be enabled to use a modelling tool that best fits their needs.

Even though a rigorously defined metamodel and a corresponding modelling method have their benefit for processing, domain experts often want to express their view of the system in an unrestricted way and therefore want to start with the instances rather than the metamodel [45, 19]. This can be helpful, primarily if the models' domain and goals are not known yet. Here, sketches of models or model examples can facilitate the understanding of the domain and can be used as a starting point for iteratively creating a metamodel [45]. Such sketches can also facilitate communication between domain experts and method engineers.

During the exploration and creation of the modelling method through the usage of example models (which is called *bottom-up meta-modelling* by [45]), inconsistency between a part of the metamodel and the example models can occur. Such inconsistency should be possible during the development but must be removed before the finished modelling method is deployed.

After the domain experts have created instances of their models, modelling environments can help by allowing them to create parts or drafts of the metamodel [19]. However, as domain experts are usually not modelling method engineers, the modelling environment

should hide the metamodeling from the users and provide certain functionalities with which the users can create metamodels within a language they can understand. Therefore, some aspects of metamodeling are applicable here, like creating classes and their attributes, but others may be too complex, like sophisticated inheritance structures or constraints. Therefore, users should be supported in creating models of their domain knowledge, but method engineers can then use the models to create sophisticated modelling methods out of them. In this thesis, we will refer to this requirement by the term **bottom up metamodeling**, which will be subsumed under the **flexibility** requirement of this thesis.

As domain-specific modelling methods represent objects of the system under study, similar objects should be represented by the same concept [175, 20]. In a modelling method, this means that one concept should be represented by one class, as the usage of multiple classes for the same concept leads to unwanted redundancy in the modelling method, which should be avoided to ease the usage and maintenance of the modelling methods. However, if redundancies occur in a modelling method, the user should be able to get rid of them by adapting the modelling method [175]. We call this requirement **avoid redundancy** in the rest of the thesis.

Even though to fully realise the value of a modelling method, a digital tool is needed, the conceptualisation of the modelling method should be implementation agnostic [45]. The platform on which the modelling tool is implemented often requires a specific metamodel structure to utilise the semantics. However, such platform-specific design decisions should be made later in the modelling method creation process and not at the beginning. The beginning should focus on the domain itself and what is needed from the users of the modelling method. This requirement will be called **implementation agnostic** in this thesis.

Creating a modelling method can be a time-consuming and cumbersome task, and therefore, the value of the modelling methods and its models must exceed the effort [160, 34]. For this thesis, we will call the requirement that a modelling method creates sufficient value, **model value**.

According to [34], model value consists of two parts:

- *Instrumental Value*: The value a model creates by being processed. Through this, additional knowledge can be derived, or it can be executed in the real world. In that way, the model is instrumentalised to achieve a specific goal.
- *Intrinsic Value*: The model's value as a knowledge representation. Models capture aspects of the world and represent them in a comprehensible way. This supports understanding and communication, which gives the model value.

For modelling environments, the notation of the modelling method is one part which influences the usefulness and, therefore, influences the usability for the user [28, 94], as the notation determines how well a human user can understand the models. To understand the models, a user must also interact with the modelling tool to create and process the models. This leads to the second part of usability, as the user must be facilitated to interact with the modelling tool to fulfil its goals easily. Therefore, **usability** will also be defined as one of the identified requirements.

4. Requirements for the Adaptive Modelling Environment

How users interact and use modelling methods can also differ, as modelling methods which offer tool support are often rigorous and formalised so that they can be used and understood by multiple stakeholders and the modelling environment [19]. However, creativity is needed in the early development and design stages to find the best solutions [22, 19]. This need for creativity calls for another approach to modelling, which is not as restrictive and supports the feeling of sketching, which engineers often do.

Such sketches can be created on whiteboards and paper, on which abstract graphics are shown to structure the knowledge and understanding of the system under study. The creation of such graphics can also be used to share understanding with a small group. However, the deep understanding of the visual presentation fades as time goes on. Afterwards, it can be challenging for engineers to thoroughly understand their idea if they did not have it in mind for some time [19].

These sketches relate to usability, as users should be enabled to use the modelling method as it fits their best needs, whether with a digital modelling tool or a pen-and-paper approach. Therefore, this requirement is also related to the **different interaction capabilities** requirement, which will be seen as a sub-requirement of **usability** in the context of this thesis.

However, usability in a modelling environment can also be interpreted in the direction of software engineering and human-computer interaction (HCI) in more general, as they also apply to modelling tools. However, this would go beyond the scope of this thesis. Here, the focus is set on the usability of using a modelling method and enabling the user to get the most value out of it.

To support modelling in the early stages of the design of new systems, a supportive tool should not restrict the users but facilitate their freedom [19]. Such tools should also allow the creation of models with non-defined constructs. This means that not all objects in the modelling area must be assigned a class from the beginning, as the class should be derived later during the design process.

Additionally, the modelling tool should not enforce a restrictive procedure onto the users [19]. The modelling tool supports the user by offering helping functionality. However, it should enable as much freedom as possible so that the users can follow their familiar way of designing systems.

To increase the value of models and the usability of modelling methods, it will be of central importance in the future that modelling tools are offered [35, 94]. However, tooling has to support the creation and visualisation of models and the processing of the models. This processing can be generic functionality like saving a model in a file or domain-specific functionality like simulation or model transformation [35, 2]. This requirement will be summarised under the term **tooling**.

However, not only tools for creating the models are needed, but also tool support for creating modelling methods and corresponding tools are necessary [160, 176, 2]. One way to create appropriate modelling tools efficiently is using a metamodeling platform [2]. These allow to focus on the modelling method and offer out-of-the-box functionality to specify modelling methods and automatically create tools for them. Additional value can be created if the platform allows the creation of additional functionality needed for

domain-specific processing. Therefore, the tooling requirement will encapsulate the need for software to create models and modelling methods.

Tools are also essential if models should be connected to the real world [35, 177, 178]. Here, real-world means information systems, which can be digital or connected to the physical world (e.g. sensors and actuators). The core idea is that the knowledge of the models can be used to trigger events in other information systems. To achieve this, the model must be represented in a digital form and be processable by algorithms. This knowledge can then be exchanged with the information system in both directions. The models can be presented to the system that acts upon them, or the modelling tool receives data and presents it as a model.

In the tooling context, deployment is also an important aspect, as modelling methods are created to be used by a single person and shared with stakeholders to share comprehensible knowledge [28]. If the metamodel is hardwired within the tool, the whole deployment and installation process must be repeated for each update, which can be troublesome for users [163].

Therefore, **deployment** is considered a requirement, where the usage of the modelling tool should be made accessible for the user and fit their need. Here, particular attention must be paid to how deployment can be made if the metamodel or the modelling tool can be adapted [28]. Ease of deployment and usage should motivate potential user to use the modelling tool and the modelling method for their problems. The deployment requirement will be subordinated to the **tooling** requirement.

The representation of knowledge through models possesses value by itself, but the value can be improved if the models are understandable by machines [37, 2, 28, 179, 161]. Suppose models and their metamodel are described in a machine-understandable way. In that case, they can be processed to support users in fulfilling their goals like deriving explicit knowledge and making it implicit, checking consistency, or executing the model in an environment external to the modelling tool (e.g., creating a database based on an ER-model). We call the requirement that the models are automatically processable by machines **processability**.

Such processing of models can be included directly in the modelling environment itself but can also include the usage of the modelled information in other systems. To support the execution of models, the captured knowledge must be transmitted to an execution environment [20]. Two ways of achieving this can either be a platform where code is created [161] or an environment where actors can receive information directly from a model without the intermediary step of code generation [178]. Independent from the approach to achieve executability, the models must be connected to a specific execution environment, which can influence the underlying modelling language or the implemented modelling tool.

The second approach also implies that existing information systems should be enabled to interpret and use models during their runtime [28, 178]. Allowing the consumption of models through a system external to the modelling tool requires that the models be offered in a compatible way. This offering and how it is implemented influences the modelling tool.

4. Requirements for the Adaptive Modelling Environment

Processability also includes feeding information into the models. This means that the modelling tool must be able to consume external information and show it in the model. These features of the processability requirement are also inspired by *seamlessness* and *reversibility* from [20].

However, processability must not only mean that algorithms are implemented in code and then executed with the models as input but also the support of other tools and approaches by allowing the integration with other paradigms [35]. As languages are used in other disciplines for other purposes, e.g. logic in AI, it can be useful to allow the integration of the abstract syntax of a modelling method into such a language paradigm (e.g., logic) to reuse capabilities of this language paradigm, like reason upon the models.

In general, the value of modelling methods can be enhanced if automated processing is enabled [180, 2]. This presupposes that a modelling tool is available and that algorithms can be defined on meta-model or meta² model level and executed on the models directly. Therefore, the requirement for **tooling** and **processability** are closely connected.

Knowing the metamodel is insufficient to comprehend the modelling method fully [181]. Users must understand why the concepts were chosen, what the goal of the modelling method is and why the design decisions were made as they are. [181] introduces the term of *meta model provenance*, with which the define an approach to capture the evolution and creation process of a modelling method. This knowledge not only supports the understanding of the modelling method as it is but also supports its further evolution and development.

Modelling methods (especially domain-specific modelling methods) can represent any domain for which they are created. Domains can be vastly different and contain many different concepts. However, if we apply linguistic abstraction often enough, there will be some common abstraction for all conceptual modelling methods. This includes the abstract constructs, like class, relationship and attribute [180, 32, 6, 168, 38]. Suppose a modelling environment allows the adaptation of the used concepts, but this circumstance must be considered in other functionalities. Therefore, a modelling environment, where the metamodel should be adaptable, must offer a basic set of concepts so that the other domain concepts can be derived from them. We call this requirement **fundamental concepts**.

In model instances, it can be helpful to differentiate between relations and references [38]. Relations are visible in the model through a graphical edge. References are attributes of objects and are used as pointers to other objects. References allow for a more versatile application, as they do not have graphical representations. This means they can also reference objects in other models or models. Therefore, a modelling environment should be able to define this using different concepts.

Providing this set of fundamental concepts to describe the metamodel and lay the basics of the models is one requirement a modelling environment should possess. This requirement will be called **fundamental concepts** in this thesis.

To better understand models, it is often helpful to have a comprehensive notation, but sometimes context is needed, like it is done with code comments [41]. This context must be provided with the models themselves so that users can access and understand

the context. This context does not necessarily have to be part of the modelling method or models directly but can be seen as an addition, which can be used as an addition. Different ways exist to add such a context to models, such as using additional objects, like a visual object in the diagram with a textual note. Alternatively, attributes are added to modelling objects, which are not part of the modelling method but are only for additional information. An example would be that it is possible to add open questions in textual to each modelling object.

Additionally, containers are often used in modelling methods to allow one concept to include multiple others [168, 38]. This inclusion must be defined in the metamodel to be useable in a modelling tool. Moreover, [180] underlines the importance of defining modeltypes, which capture different viewpoints in modelling methods. Therefore, the so-called model types must be on par with model views.

As these concepts are not necessarily a core part of a modelling method, we will call them **supportive concepts**, and this term is also used to name this requirement in this thesis.

Table 4.3 provides an overview of the requirements identified in the literature. Only the main requirements are shown in this table, and the sub-requirements discussed in the text above are omitted. Sub-requirements are those requirements which belong to another requirement and are therefore subordinate to it.

The table separates the requirements into *requirements* and *supported tasks*. Requirements directly influence a modelling environment and must be directly considered in its design. Supported tasks in this context are focused on tasks a user should be able to accomplish. Therefore, the modelling environment must not directly ensure them but offer functionality to the users to achieve these tasks. For example, being ontologically complete and clear is nothing the modelling environment can support, enforce or check automatically. However, it provides users the tools to define the concepts in the modelling methods that best fit their needs.

The requirements listed in Table 4.3 are ordered alphabetically and not after any other property. Therefore, their place in the list does not correspond with their importance or other characteristics of the requirements. Additionally, it should be mentioned that all the requirements identified in the literature are mentioned here. That does not mean that they are adapted one-to-one in the design of the adaptive modelling environment. This is because the goal here differs from modelling methods, which focus on a formal, complete, and exact description of a system under study.

4.3. Requirements for the Design Thinking Digital Twin

The section before introduced requirements based on a literature review, focusing on creating and adapting conceptual modelling methods. This section will introduce additional requirements for the *Adaptive Modelling Environment* and using the design thinking digital twin in workshops. These requirements are based on the introduction to the environment provided in chapter 3. The requirements introduced here may correspond to or use the requirements introduced above.

4. Requirements for the Adaptive Modelling Environment

Requirements	Adaptability [34, 35, 28, 38, 162, 163, 19, 45], Comprehensible Notation [163, 170, 168, 169, 40, 41, 107, 37, 160, 171, 20, 22, 41, 173, 172], Consistency [163, 161, 165, 19, 41, 162], Flexibility [37, 45, 19], Fundamental Concepts [180, 32, 6, 168, 38, 41], Implementation Agnostic [45], Integrability [28, 94, 19, 20, 160], Multi view support [166, 37], Processability [37, 2, 28, 179, 161, 20, 178, 35, 180], Tooling [163, 35, 94, 2, 160, 176, 2, 177, 178], Usability [28, 94, 22, 19]
Supported Tasks	Avoid Redundancy [175, 20], Extendability [28, 163, 164, 19, 29], Longevity [94], Model Value [160, 34], Meta Model Provenance [181], Ontological Complete and Clear [29, 94, 20, 160], Reputability [174, 35, 41, 39, 28], Scalability [94, 20], Simplicity [163, 94, 20, 106, 41]

Table 4.3.: Table with an overview of the requirements found in the literature review

4.3. Requirements for the Design Thinking Digital Twin

Workshop participants should be able to explore, structure and visualise their conceptual understanding of the problem and design innovative solutions. Therefore, we must provide an environment which supports them in structuring and concretising their understanding, allowing them to produce coherent models of their ideas. There, the *Adaptive Modelling Environment* must enable the definition and adaptation of the used concepts during its usage to support exploring the idea space. To support the communication of ideas and exploration, especially in workshop settings, the models must be intuitively comprehensible by the users. This calls for an expressive notation, which has meaning for users in the workshop in the form of semantically rich graphics. We call this requirement **intuitive notation**.

Therefore, concepts should be presented through textual labels and graphical icons, which humans can interpret. As these icons must fit concrete application domains and their current understanding through the user, the visualisation should be adaptable. This leads to the requirement of an adaptive visualisation or an **adaptive notation** to keep the terminology consistent with the one introduced in section 2.3. The notation should be achieved using existing icons and pictures, as manually creating each representation would impair the workflow of creating meaningful models.

The visualisation will follow a diagrammatic structure, like diagrammatic modelling (as discussed in section 2.3). However, the goal here is not to have rigorously defined concepts that can be used to thoroughly describe and analyse scenarios, as often done with diagrammatic models. The models should be on a higher abstraction level to support creativity and co-creation, as in design thinking (see section 2.1 for more details). Therefore, digital means of visualisation, as usually done with modelling methods, and physical means, which facilitate the usage in workshops, are needed.

However, the physical means, like paper figures, are insufficient, as they do not allow for automated processing of the presented knowledge. Therefore, one requirement is that the adaptive modelling environment supports both physical and digital modelling. Different supportive algorithms can be executed based on the digital model's structure. Like allowing to implicitly save discussed knowledge, send and gather information from social media platforms or even ask generative AI for ideas and suggestions. Such automated processing can only be made through a digital twin in the form of a conceptual model of the physical model. The digital twin must be created automatically and dynamically for workshop use.

The visualisation through semantic-rich graphics is not enough to capture the knowledge and make it processable in the digital twin from design thinking. Structural elements like attributes and relations must be available to make the implicit knowledge provided through the proximity of the paper figures or discussion available to the model. The users must also tailor these to meet their understanding. Therefore, not only pictures arranged on a two-dimensional canvas should be saved, but also an abstract structure in the background, aka a metamodel. The structure must be saved so the machine can understand and process it to offer supportive functionality. This leads to the requirement of **processability**, meaning that the machine must be able to process the information of the digital twin, which is a representation of the physically created model.

4. Requirements for the Adaptive Modelling Environment

The digital twins are used during the workshop alongside the physical models. To make this useful, the physical models must be automatically transformed into the digital representation, which is used as input for further processing. Therefore, the environment must offer ways to mirror the physical model in the digital model and make it available within the modelling environment. We call this requirement **digitalisation**, a prerequisite for making the processability more useful.

The modelling environment should allow the representation and adaptation of concepts and support the integration of **supportive concepts**. Supportive concepts are elements added to the modelling method, which help annotate or add additional context information. Therefore, they are not part of the modelling method but can be used by users to provide context to models. This is similar to making comments in source code. Examples of supportive objects are making textual annotations to modelling objects or models as a whole or visually grouping objects together to explain them.

To extend the possibilities for capturing knowledge, users should be able to add generic supportive objects to their models and additional information to the models using a different modelling language. Such an extension can be used to capture further knowledge of the domain. This requirement is derived from extending modelling methods, as described in section 3.4.2 and is important to elaborate on additional aspects of the model. This requirement will be called **integrability**, as the extension means that an existing modelling method can be integrated into the modelling method.

The *Adaptive Modelling Environment* uses models on a high abstraction level applied while exploring the domain and ideas. As these ideas should be implemented, if they are deemed useful, the knowledge of the models on the high-abstraction level needs to be enriched with additional knowledge. Here, the **integrability** requirement calls for the capability to extend these high-level models with concepts of existing modelling methods. The objective behind the extension is not just to use the high-level models as input for manually creating models with other modelling methods. If a user manually creates the derived models and saves them separated from high-level ones, only the user knows how they are related. Therefore, linking the high-level models to those used later should be possible. This is done by transforming the high-level ones in other modelling languages and linking them together so that users can find the relation independently if all models are created in the same or different environments.

Not all the knowledge must be represented in one model, but the users should be able to structure it by separating them into different models and linking them together. Such a structuring is also essential as different users must extend models with different concepts. This leads us to the requirement of **filter**. Models should be used throughout the development procedure and support the interaction of multiple stakeholders. The right modelling method should be used for the right usage scenario to facilitate the utilisation of models. However, as not every stakeholder needs to have the same information in the same detail, this can overwhelm the users. Therefore, the filter requirement allows users to define filters, which generate a specific view of a model, influencing what aspect is shown. This can range from a more general level, e.g., that only certain models are available to a specific user, to a more granular level, where the view influences which

concepts or objects are shown in a model or even which attributes are available for a specific object.

Filters are limited to the interface of a specific user or user group. The metamodel in the background should not be reduced, as the information should still be available to other users. Filter in this way should be adaptable and combinable so that users can also see concepts from other filters to explore a more extensive model of the system.

However, the modelling environment must first support multiple modelling languages to use filters. Then, different models can be generated for the different languages corresponding to different viewpoints. This requirement will be called **multi-view modelling**.

Additionally, not all users need the same algorithms and mechanisms. Therefore, these should also be adaptable to user needs. This leads to **configurable mechanisms**, meaning the mechanisms should be tailored to users in two ways. This also allows the reuse mechanisms in the form of extensions. These can be loaded into the modelling environment, configured and then used.

The following requirement will be called **automated adaptation** and contains the need for adaptations that can be applied in an automated way. Therefore, the mechanisms and algorithms created for a modelling method can include the adaptation of the metamodel and the modelling environment. With this requirement, adaptations can be included in the modelling procedure (as defined in [2]) and support users in fulfilling their goals. For example, this is important if a new mechanism should be included in the modelling environment, which needs specific information in the modelling method. Alternatively, the adaptation of the metamodel should be included and tailored to the usage in a modelling method. No generic adaptation mechanisms should be used, but ones implemented and integrated into the modelling method. This way, the method engineer can integrate the adaptation functionality, and the users do not have to prepare the metamodel themselves.

The following requirement is connected to the above-introduced one and is called **adaptable modelling environment**. Here, the focus is set on adapting the modelling environment itself and not only the modelling method implemented in the environment. Therefore, the functionality offered in the modelling environment and how users interact with it should also be adaptable to meet the users' needs.

The **adaptable modelling environment** stands for mechanisms' ability to be included in a modelling environment without overburdening the users. A big part of this requirement focuses on tailoring the interaction of users with the modelling environment. This is important as in different situations where users create models, different interaction possibilities can be useful. Users should then be able to adapt their interaction to their needs. For example, if the models are created in a workshop setting, as discussed in the example for the settings (see chapter 3) or to use a drag and drop based GUI if the user is alone. If physical modelling is supported, this is connected with the **digitalisation** requirement.

Different ways of creating models for the user can support them in different scenarios. For example, Baltes and Diehl [22] found that sketches and diagrams can be used in software engineering to design, understand and explain aspects of the system that should

4. Requirements for the Adaptive Modelling Environment

Requirements	Adaptable Modelling Environment, Adaptive Notation, Automated Adaptation, Comprehensible Notation, Configurable Mechanisms, Digitalisation, Filter, Fundamental Concepts, Integrability, Intuitive notation, Multi-view modelling, Processability, Supportive Concepts, Usability

Table 4.4.: Table with an overview of the requirements derived from the *Adaptive Modelling Environment* as introduced in chapter 3

be created. The sketches themselves were usually informal and produced on paper or whiteboards. Additionally, these informal sketches and diagrams are often related to high-level representation and not on the abstraction level of coding. After the sketches are created, they are often archived and forgotten. However, connecting such high-level sketches with high-level digital models can increase the model value for the users.

However, not only are the different interaction capabilities important here, but they are also important in general to allow the modelling environment to adapt the available functionalities to better reflect the user's needs. Therefore, such a modelling environment must support adding, hiding and configuring different functionalities to tailor the modelling environment.

Table 4.4 summarises the requirements discussed in this section. The title for the row *requirements* is used to make the summary table fit to the ones established for the requirements from literature (see Table 4.3) and for the consolidated requirements (see Table 4.6).

Here, the focus is set on the requirements, as the focus of the *Adaptive Modelling Environment* itself is set on how modelling environments can support the creation of models in uncertain domains. Therefore, the identified requirements will directly aim at functionalities the modelling environment should offer.

The requirements listed in Table 4.4 are ordered alphabetically and not after any other property. Therefore, their place in the list does not correspond with their importance or other characteristics of the requirements.

4.3.1. Requirements for Using Design Thinking Digital Twins in Workshops

This subsection details the requirements supporting the usage of *Adaptive Modelling Environment* in workshops and improving the processing of the created digital models, focusing on which semantics the modelling environment must understand to support the users. The here described requirements are based on the *Adaptive Modelling Environment*, as described in section 3. The semantics that must be understood are highlighted here, as they are essential to support the users by reducing the burden put on them.

Understanding the semantics includes the tangible models, which must be understood to create and utilise digital twins of design thinking. The goal is not to define what humans must understand but to discuss what semantics computer systems must understand to support human users. If the machine better understands the idea created in the workshop, let the participants focus on the crucial aspects, like ideating and designing the innovative idea.

In the context of the adaptive modelling environment, three phases were identified where semantic understanding can be beneficial: preparation, design and exploitation. Section 5.1.3.2 will discuss these three phases in more detail.

Possible concepts used during the workshop must be identified to prepare a workshop. These concepts are prepared as paper figures to create physical models. These prepared concepts must be tailored to the workshop and available within the modelling environment. This is a prerequisite to use the digital twin.

An IS can support a workshop organiser by proposing concepts applicable in the workshop context. Here a prerequisite is that a description of the workshop's goal is available. This description should not be created explicitly for the supporting IS but should be reused if such a description was created for human actors. For example, suppose an abstract or short introduction text was written to be sent to the workshop participants. In that case, this text can suggest possible concepts to a workshop organiser. Here, the IS must understand the text written in natural language for the human actors so that no separate computer-understandable description must be created.

However, to further improve the support for a workshop organiser, individual concepts should be recognised, as well as their relations to each other and the attributes that describe them. If this information can also be extracted, it can be used within the workshop and integrated into the algorithm, digitalising the physical models into digital ones. Therefore, the supportive IS needs to be able *propose workshop concept structure*, which calls for the need to understand natural language. However, with the *propose workshop concept structure* requirement, not only the direct suggestion of concepts is meant, but also the relation between and attributes of concepts.

The result is the set of concepts described in a computer-understandable way, which can be used in the modelling environment. Having it in such a structure facilitates the knowledge exchange between the different components within a modelling environment. One way to do this is ontologies (as introduced in section ef sec:ontology), which allow the machine to understand and traverse the information. We call this requirement *computer understandable encoding*. This includes the format in which it is saved, like ontologies and the structure, e.g., the used classes.

4. Requirements for the Adaptive Modelling Environment

During workshops, the modelling environment can increase the value for the workshop participants by digitalising their physically created models for further processing within workshops or later exploitation. These digitalised models must be more than mere pictures to support a sophisticated exploitation. Therefore, the modelling environment must understand the semantics of the physical model and transfer as much knowledge as possible into the digital model.

This includes the information explicitly represented in the physical model and implicitly encoded information. Explicitly is, for example, the use of physical figures, their position and handwritten text, which is added to the figures. Implicit information in this context means that humans can automatically derive knowledge because of their cognitive abilities or the information discussed in the workshop but not directly represented in the physical model. Semantics that humans can easily derive but not as easily by computers are, for example, relations between two physical figures by using the semantics of another physical figure or by the proximity of two physical figures. Information provided only through discussion could be properties of the used physical figures, like the role and age of a human.

To automatically create the digital model, the modelling environment must first understand which physical objects were used and their relative position to each other. In the context of this thesis, this understanding will be encoded in the *object detection* requirement, which stands for identifying the position and type of the used physical figures. In this context, the type of the physical figure means an identifier so that the modelling environment knows which figure was used. The type is also essential so the modelling tool can set the appropriate notation in the digital model.

Based on the types of figures and their position, another meaning must be derived to improve the digital conceptual model. These are the relations that the physical figures have with each other. The modelling tool should make these relations explicitly available by adding a relation between the digital figures. We call this requirement *relation identification*. Which figures are related or connected is based on their position to each other. This can be that they are directly next to each other, or a special figure is positioned between them. Both approaches must be able to calculate the positional relation between two or three figures.

Another meaning that can be provided over the used physical figures is if they are enhanced with handwritten text. This represents flexible added information in the workshop, which is needed to understand the created model better. Additionally, many creative methods are based on using sticky notes with text written on them. Therefore, the modelling environment should be able to recognise handwritten text and translate it to computer-processable strings. Additionally, the recognised text must be assigned to the object in the modelling tool, which represents the physical figure, as adding the text to the physical object includes meaning. We will call the requirement for interpreting handwritten text *text identification*.

Not all information is directly visually encoded, but it is provided through interpreting a concept in a context or through the discussion between the workshop participants. More information than visualisation is needed to increase the knowledge captured in the

4.3. Requirements for the Design Thinking Digital Twin

digital twin. This additional information can be encoded as attributes of the concepts and relations. Understanding the physical figures can be interpreted in two ways. First is what we call the *provided structural information*, which means that the modelling environment understands essential attributes of the concepts and relations and offers them to the users. In this requirement, the attributes of the created objects are empty and can be filled in by the users. Building on this requirement, we introduce the *provide instance information* requirement, which encodes the need to provide empty attributes and values for them. These two requirements are not essential but supplementary requirements, meaning they are not mandatory, but if they are fulfilled, they can provide an enrichment.

The information provided can be defined in advance and generated or gathered during a workshop. Here, AI methods, like generative AI, can be used. To achieve this, information gathered in the digital model can be embedded in a prompt and sent to a generative AI, which then returns attributes, part of the idea or questions, which the workshop participants evaluate and then add to the digital model or not.

However, not all additional information can be derived automatically or added in advance. Therefore, the workshop participants should be able to add information to the created digital models naturally. The most direct way for the workshop participants to provide this information is via speech. Therefore, the following requirement is called *speech understanding*, which encompasses that the workshop participants should be able to tell the modelling environment information about an existing digital model and its objects, and the modelling environment should then be able to add this information to the digital model so that it can be captured and exploited later.

To improve the quality of the created idea, not only input from AI can be used, but also from other involved stakeholders. Because the amount of workshop participants should not be too large [17, 18], other ways to gather input from additional stakeholders may be needed. Therefore, it must be able to send the created idea to other stakeholders, which can be achieved through the digital twin, giving them a way to provide feedback. Then an IT system is needed, which analyses and aggregates the provided feedback.

We call this requirement *feedback analysis*, which provides the prerequisite that the feedback is provided in a textual statement to allow more freedom for the feedback provider. Therefore, the supportive IS must be able to understand the feedback and assign them categories on how it can be aggregated. For example, if the comment was positive or negative.

For some problems, multiple workshops with different groups of stakeholders may be necessary. In such a case, it would be helpful if the results of these workshops could be compared and merged automatically. In the context of *Adaptive Modelling Environment*, the labels used in the model and the metamodel can differ. Therefore, a supportive IS must be able to match the different terms based on their semantics. This leads to the requirement of *concept similarity*, which measures how similar two concepts are. This can either be done based on the semantic similarity between their labels or their structure.

To compare the labels, the supportive IS must understand the meaning of the labels and compare them. Therefore, an understanding of natural language is necessary. If similarity should be defined based on the structure of the concept, then this structure

4. Requirements for the Adaptive Modelling Environment

Requirements for processing semantics	Computer Understandable Encoding Concept Similarity, Feedback Analysis, Model Transformation, Object Detection, Propose Workshop Concept Structure, Provide Instance Information, Provide Structural Information, Relation Identification, Speech Understanding, Text Identification,

Table 4.5.: Table with an overview of the requirements to process semantic for the *Adaptive Modelling Environment* as introduced in chapter 3

must be available in a computer-understandable way.

After the workshops are finished, the knowledge created should not be saved and forgotten but used to support the idea's implementation. For this implementation, other modelling methods can be used to concretise the idea and make it implementable. These additional models can be created by humans manually after they interpret the models from the workshop. Alternatively, a supportive IS can create a starting point for the additionally used modelling method through model transformation. To achieve this, rules must be defined, which map the concepts used in the workshop model to the metamodel of the additional modelling method. This falls in the requirement of **Computer Understandable Encoding**, as such an encoding is needed to share the knowledge with other modelling methods.

Table 4.5 shows an overview of the identified requirements of this subsection, which can improve the usage of digital conceptual modelling in workshop settings. The requirement stands for capabilities that support the usage but are not necessarily mandatory for the modelling environment, as they can also be used without them. However, then the functionality is limited. The requirements are ordered alphabetically, and their order does not allow any conclusions to be drawn about importance.

4.4. Consolidated Requirements

In the last two sections, we introduced general requirements that can be found in the context of modelling methods and their tools. In this section, these requirements will be consolidated and aligned with the goal of this thesis. Therefore, an overview of the requirements will be provided and insights into how they can be interpreted in the context of this thesis. Additionally, it will be discussed which requirements identified in the literature review (see section 4.2.2) can be applied to this thesis and which are not fitting.

Table 4.6 shows an overview of requirements consolidated from the literature review (see

Requirements	Adaptable Modelling Environment Adaptability, Automated Adaptation, Comprehensible Notation, Configurable Mechanisms, Consistency, Extendability, Flexibility, Implementation Agnostic, Integrability, Fundamental Concepts, Multi View Support, Processability, Tooling, Usability
Supported Tasks	Avoid Redundancy, Longevity, Meta Model Provenance, Model Value, Ontological Complete and Clear, Reusability, Scalability, Simplicity

Table 4.6.: Table with an overview of the requirements consolidated from the literature (see section 4.2) and the requirements identified for the design thinking digital twin (see section 4.3)

section 4.2) and the ones identified for the design thinking digital twin (see section 4.3). The table separates the requirements in *requirements* and *supported tasks*. They have the same meaning introduced in section 4.2.2. Requirements will directly influence the modelling environment and must be directly considered in its design. Supported tasks in this context mean that these requirements should be enabled for the users, but the modelling environment must not or cannot directly enforce it. They represent goals that the users should be capable of reaching.

The requirements listed in Table 4.6 are ordered alphabetically and not after any other property. Therefore, their place in the list does not correspond with their importance or other characteristics of the requirements. Further, the table does not reference the literature sources that influenced them, as they were adapted to this thesis. The mapping of requirements with their inspiration in the literature can be found in section 4.2.

Next, we will discuss how and if the identified requirements relate to the context of

4. Requirements for the Adaptive Modelling Environment

this thesis. Especially the requirements from the literature must be analysed in how they relate to the goal of this thesis, as they also include general requirements for modelling method engineering, which may not translate directly to requirements for the *Adaptive Modelling Environment*.

4.4.1. Adaptable Modelling Environment

This requirement is vital for the goal of this thesis. User should be able to adapt the modelling environment to their needs, including the modelling method and the available concepts. However, as the value of models can only be fully utilised through an automated processing of their content, such aspects must also be tailored to the user. Therefore, the interface and the available processing functionalities of the modelling environment should be adaptable to the user's needs.

Not only should the user be able to influence it directly, but a provider of modelling methods and mechanisms should be able to offer them in a way that they can be plugged into the modelling environment and made available for the users. This is connected to the deployment requirement, as the complete modelling environment should not be updated, rebuilt, and redeployed, which would lead to unnecessary overhead for the users. Integrability also influences this requirement, as a modelling environment offers the possibility to integrate existing modelling methods (or part of them); it should also allow the inclusion of the mechanisms that belong to these modelling methods.

Modelling environments can be used in different settings, like alone by a user, in a workshop, by different users and so on. These different settings should also influence the modelling environment to tailor it to the users' needs better. A method engineer or workshop organiser can do this in advance, or the environment should be adaptable during its usage. This does not mean that the adaptations must be possible by a novice user, but they should be possible in general.

4.4.2. Adaptability

The modelling environment must support the metamodel's adaptation to allow the modelling method's tailoring and the environment to the users' needs. Operations must be provided, which then execute the adaptations in the modelling environment. These operations are also connected to the consistency requirement, as these operations must ensure that the modelling environment is still working after the adaptations.

The focus is not set on how useful specific adaptations are for certain modelling methods but on what the users must be offered to make adaptations without making the modelling environment unusable. However, if the modelling method is more helpful to the users after the adaptation, it depends on the users.

This requirement focuses on the adaptations of the modelling method and its metamodel during its usage. The adaptation operations should be executed in the modelling environment and applied directly during the runtime. Here, the adaptation capabilities of a user must not be the same as those of the method engineer, as the method engineer can

implement the adaptation capabilities to fit the modelling method's needs. This guides users and enables novice users to use the adaptations.

Adaptation will also consider multiple aspects of the modelling environment, like the modelling method and its notation, available concepts and their characteristics. It is also a prerequisite for fast prototyping, allowing us to make and evaluate changes quickly.

4.4.3. Automated Adaptation

Automated Adaptation is closely related to adaptability and stands for the capability of a modelling environment to not only offer the adaptation capabilities via a user interface directly for the user but also to apply them through automated means so that they can be used in implemented mechanisms. For example, this can be offered through special commands in a scripting language to enhance the modelling environment or through an API endpoint.

The automated adaptation is useful as the adaptation of the metamodel can be included in modelling procedures and, therefore, integrated into the modelling method. Additionally, this allows for automatically adapting the metamodel of a modelling environment to fit the prerequisites of an implemented mechanism that is added to the modelling environment.

By integrating it into the modelling procedure, the adaptations become more intuitive to the user and, therefore, can be easily used.

4.4.4. Avoid Redundancy

This requirement focuses more on defining the metamodel than something easily enforced in a modelling environment. The goal is that multiple concepts in the metamodel do not refer to the same domain objects.

A modelling environment can support this requirement by allowing adaptations that merge two concepts that refer to the same objects or other adaptations the user sees fit. Therefore, the environment should offer functionality to avoid redundancy.

4.4.5. Comprehensible Notation

This requirement is essential in creating diagrammatic modelling methods as it provides the base for human interpretability of the models created with the modelling method. However, in this context, the notation is heavily based on the design decisions of the method engineer and not on a modelling environment.

Here, the modelling environment must provide certain functionalities so that users can create the wanted notation. Users should also be enabled to use notations that fit their needs. This leads to the requirement of a modelling environment to render the graphical notation, defined in the modelling method. However, it also allows the usage of different notations for the concepts to meet the needs of specific cases of users.

As defined in chapter 3, the focus of the *Adaptive Modelling Environment* is to use models on a high abstraction level, which are easily understandable by humans. This leads to the requirement for allowing the usage and adaptation of intuitive notations of the

4. Requirements for the Adaptive Modelling Environment

concepts. Therefore, the usage of complex icons should be possible and not only the usage of abstract forms, as their representation does not directly incorporate an interpretable meaning. Only by learning the abstract form can the meaning be understood.

As the notation is the primary channel on which semantics is related to the human users, the notation should not only be able to represent a concept but also allow interpretation of the concept in the context of the model. This means that the modelling environment should allow showing information within the models in their notation. Therefore, a dynamic notation is needed, which can automatically change based on information saved in the model.

However, the created models should not only be pictures, which can be visually interpreted, but real models, which can hold knowledge of the domain and process the models to increase the value. Therefore, the models as a whole and the concepts and relations within them must be enhanced with saved information that makes sense in the context of the goal of the modelling method. Additionally, the modelling environment should be able to process the models to allow for the application of mechanisms to increase the value of the users.

4.4.6. Configurable Mechanisms

This requirement focuses on the mechanisms offered in the modelling environment that can be added. The mechanisms can be directly integrated into the modelling environment (e.g., through a script) or included by calling an external API endpoint. This requirement is essential because the mechanisms can be configured to different usage settings. This may be necessary if the mechanism is loaded at runtime and must be adapted to the current metamodel or if the tools are used with different endpoints, for example, if the modelling environment is used in different workshop settings and must be configured to fit the different settings.

However, the requirement does not aim to force creators to make their mechanisms configurable but to provide them with capabilities of the modelling environment so that they can apply configuration capabilities if needed.

4.4.7. Consistency

Consistency in the context of a modelling method can be interpreted in different ways. In the context of this requirement, we focus on the consistency of the adaptable artefacts in the modelling environment. These artefacts can be models, the metamodel, the user interface or mechanisms. Consistency is essential so that executed adaptation operations do not break the modelling environment. For example, deleting a class should not lead to existing models not being accessible anymore. Alternatively, after renaming an attribute, mechanisms cannot be executed anymore. Therefore, consistency in the context of the adaptation operations is the core focus of this requirement. This also includes the fact that an inconsistency between the metamodel and the models should be avoided.

Additionally, a modelling environment should allow users to define consistency restrictions, which are more focused on the domain knowledge. These can be, for example, the

restriction of values for attributes or relations forbidding or enforcing relations between different objects. In this context, the requirement focuses not on meaningful consistency constraints in the domain but on offering the users the ability to define their own, which can then be enforced by the modelling environment.

4.4.8. Extendability

This requirement is related to adaptability, as it also changes the modelling environment and the underlying modelling method. The difference is that adaptability focuses on changing existing artefacts within the modelling environment, and extendability focuses on adding new concepts and artefacts to the modelling environment, especially in the context of adding new mechanisms.

In this way, the capabilities of the modelling environment itself can be extended to tackle new needs of users. If the modelling environment is extended, it must be held consistent with its current state to avoid unwanted behaviour.

4.4.9. Flexibility

This requirement states that users should be allowed to use the modelling environment as they see fit. Therefore, users should get as much freedom in usage as possible to meet their needs. This includes modifying the modelling environment and modelling method to their needs by adapting and extending it. Flexibility is essential if the users work in a domain which is not yet completely understood and designed. In such contexts, freedom is essential so as not to restrict the solutions that can be found with the help of the modelling environment.

Flexibility can even go as far as no concepts are available in the modelling environment, and the users start from scratch in identifying and modelling objects, which can then be grouped or abstracted into classes. In this case, the users can create objects of their liking without the need to define classes beforehand. Additionally, they are free to create their own classes, which they need.

Additionally, the modelling environment should be adaptable to the users' needs regarding its modelling method and how users can work with it. This means that users can choose different ways to interact with the modelling environment to create the models. This can be through a GUI like often, but also other approaches like using physical objects, which are automatically transformed into a model. Other examples would be users wanting to interact with voice commands or automatically create models from log files gathered from a real system.

However, the flexibility requirement does not mean there cannot be any restrictions on what the user can do; instead, users can choose how they want to do certain things. However, in some cases, the users have to fulfil certain restrictions. For example, if simulation should be used, it may be necessary that the models have a particular structure and specific information provided. However, the user should not be forced to comply with this structure if no simulation should be used.

4.4.10. Fundamental Concepts

This requirement focuses on the fundamental concepts available in the modelling environment to define the metamodel. They are important as the basic handling of all concepts used in the modelling environment will be based on the fundamental concepts. We will denominate this basic handling of the linguistic semantics of the concept. We do this to distinguish it from the domain semantic, which is the meaning assigned by mapping a concept in the modelling tool to a concept of the domain. However, linguistic semantics mean the meaning and behaviour that an instance of this concept holds in the modelling environment.

For example, the three concepts *class*, *relation*, and *attribute* (cf. [6]) build the fundament to model ontological aspects from a domain by representing the objects in this domain. However, they also need a definition of how they are interpreted within the modelling environment. For example, instances of classes are objects placed on the drawing area of the environment, attributes are saved to the objects of the class and can be filled out, and relations connect two objects. The assignment of attributes to objects and the relation between objects must also be exploitable by the functionality of the modelling environment.

However, classes, attributes and relations are not enough to capture everything. Another fundamental concept is a *container*, which groups objects. The container should not only be used to visually group objects but also allow the modelling environment to retrieve those objects that are visually added to the container. This should be possible without the user manually adding additional information, like a relation between the objects and a container.

Another important concept is the *modelling view*, which is needed to define sub-sets of the concepts in a modelling method used together to describe a specific aspect of the domain. The modelling environment should interpret them as different diagram types with their own set of classes available to the user. However, the view type should be understood by the modelling environment as it can influence which mechanisms can be applied to the models. Additionally, this information should be used to define links between different views, as these links must understand to which view they are pointing to check the validity.

Last but not least, the modelling environment should offer supportive concepts that do not directly represent concepts of the domain but can be used to provide context to the models. Two important ones are the ability to add textual notes to a model to provide information for human readers and the second is to visually group objects. However, not necessarily as part of the domain description but to highlight them for human interpretation. One way to achieve this is to provide a generic container without domain semantics. These supportive concepts should be usable to all modelling methods in the environment so that not each modelling method needs its own implementation.

4.4.11. Implementation Agnostic

This requirement focuses on the creation of modelling methods themselves. However, it is only marginally applicable as a requirement for a modelling environment, as it says that the definition of a modelling method should be independent of a concrete implementation. As this thesis focuses on implementing a modelling environment, the implementation agnostic requirement cannot be directly supported.

4.4.12. Integrability

This requirement focuses on enhancing the capabilities of a modelling method by allowing it to integrate existing modelling methods or parts of them into the ones now used. This perquisites other modelling methods already available in a format that the modelling environment can interpret. However, integrability does not only mean that different modelling methods can be used in parallel in the same modelling environment but also that the different methods can be linked together to exploit synergies.

The requirement is important for the context of this thesis, as it supports the user in capturing the knowledge they need. The idea here is that a new aspect of the domain is identified while working with the modelling environment. To capture this, an existing modelling method is loaded into the modelling environment. Then, the models of the new modelling method are linked with the previous existing one so that the knowledge can be connected.

4.4.13. Longevity

This requirement focuses on increasing the utilisation of the modelling methods implemented into the modelling environment. In the context of this requirement, this should be achieved by enabling a long usage period for a modelling method by keeping the modelling method relevant to the domain. Therefore, this requirement is supported by the adaptability, integrability, and reusability requirements, as they enable the evolution of the modelling environment within the domain. By allowing users to adapt the domain concepts and the functionality for the modelling method how they needed, they are motivated to use it for a longer time.

However, this requirement is not a core requirement for this thesis, as it also focuses more on the usage of a specific modelling method and not more generically on a modelling environment.

4.4.14. Meta Model Provenance

This requirement focuses on keeping track of the evolution of a metamodel to understand the changes and capture the context. This information is helpful for method engineers who adapting and designing modelling methods or for modellers who formalise complex and important systems. A modelling environment can support this requirement by tracking the changes and by allowing the provision of contextual information.

4. Requirements for the Adaptive Modelling Environment

However, this information will be of secondary importance for the adaptive modelling environment used in workshops, as novice modellers use it. Additionally, the system under study in such cases is very volatile, which can lead to many changes that are only made for a short time.

4.4.15. Model Value

This is a very high-level requirement that models of modelling methods should create value for its users. The value of the model influences how often it is used and how well users receive it. How such a value looks in a concrete situation depends on the modelling method, the modelling environment, the situation and the user. Here, no general way of creating value can be defined.

A modelling environment can help to utilise models to increase the value. This can be achieved through an automated processing of models. Either to reduce the work on creating models, as the modelling environment can automatically create (parts of) models or to execute algorithms on top of the models to create additional knowledge and value. Such a utilisation can only be reached through tooling support. Therefore, the functionalities a modelling environment offers can improve the model's value.

4.4.16. Multi View Support

This requirement focuses on capturing different viewpoints about the system under study. This means that the modelling environment supports the definition of different types of models, containing different concepts to describe the system under study. Therefore, the modelling environment must be able to define such viewpoints and manage them. It is important that the views are not just available next to each other but can be linked to each other to create synergies. In the modelling environment, there must be a way to capture and process such links.

4.4.17. Ontological Complete and Clear

This requirement focuses on creating a modelling method and the design choices that are made in creating it. Therefore, it is not a requirement that a modelling environment can directly support, as it is based on decisions by the engineer.

However, the modelling environment can support its users by allowing them to adapt the metamodel of the modelling method to achieve this requirement. Ontological completeness can be supported by allowing users to add concepts that are needed but missing. For example, if users want to model their system and identify a concept not yet covered by the modelling method, they can add it.

Ontological clearness can be supported by adapting the metamodel if concepts too similar to each other are identified. For example, users add a new concept and, after refining it, recognise it is similar to an already included one. In such a case, the metamodel can be adapted by merging the two concepts and strengthening the clarity of the modelling method.

4.4.18. Processability

This requirement means that the models should be processable by the modelling environment to enhance their value. First and foremost, this means that the visual concepts represented in the models must be usable by the modelling environment in order to process the models. This means the internal structure of the model must contain more than just a picture. The processing can either be some generic mechanisms offered by the modelling environment out of the box or more specific ones implemented for more concrete cases and metamodels.

An adaptive environment should adapt the available mechanisms to the user's needs. This means that the mechanisms must be provided so that the modelling environment can execute them and that including a new mechanism does not require rebuilding and redeployment of the whole modelling environment.

4.4.19. Reusability

This requirement means that modelling methods should be defined in ways that they can be reused in other modelling methods. The focus is set on the domain concepts that can represent a system under study. The creation of the domain concepts will be eased, as not everyone must be created from scratch, but existing ones can be reused and adapted to the current needs. This includes the description of the concepts so that they can be made available in the modelling editor, and it can additionally contain information on how they can be used in certain mechanisms within the modelling method.

However, not only should domain concepts be reusable, but reusing available mechanisms is also essential. A Prerequisite is that the mechanisms fit the used metamodel. The reusability of the mechanisms enhances the model value by offering functionality for the users. It also supports extendability by offering a way to extend the functionality through available building blocks.

4.4.20. Scalability

This requirement identified from the literature applies to the method engineering process rather than something a modelling environment can directly support. Other requirements like adaptability and multi-view support can enable the users to create scalable models, but the modelling environment cannot ensure it directly.

4.4.21. Simplicity

This requirement from the literature states that it is important that a modelling method is conceptualised as simply as possible, meaning that it only contains concepts important for its goal. This requirement focuses on the engineering of modelling methods and cannot directly be supported by a modelling environment. A modelling environment can support users by allowing them to adapt the modelling method to fit their needs minimally. This can be done by removing or hiding concepts they need or only adding a concept if needed.

4. Requirements for the Adaptive Modelling Environment

4.4.22. Tooling

This requirement calls for a computer-aided tool that supports the usage of the modelling method and maximises the model value. This requirement encapsulates the essence of *Adaptive Modelling Environment*. It is always fulfilled, as the thesis aims to provide tooling support for models in circumstances where the domain is not fully known.

4.4.23. Usability

This requirement focuses on how usable the modelling method within the modelling environment is to the user. The focus is not on the usability of the modelling environment's interface but on how well the interaction with the modelling method (through the modelling environment) can fit the users' needs.

This can either support the creation of valuable models, which, in the context of this thesis, include the freedom to integrate different modelling methods or adapt existing ones to changed requirements. Additionally, the modelling environment should support the creation of valuable models so that users do not manually define each aspect of the model but have automated support. Automated support means that parts of the models can be created, e.g., from data or another representation, or other models can be derived from a certain state.

Last but not least, users should be able to interact with the modelling environment based on their circumstances and needs. This may not always be a GUI, where models can be created using drag and drop mechanisms. In workshops, users may want to add information about the model through natural language or use physical representation, like paper figures, sticky notes or drawings on a whiteboard, as models, which should be easily captured digitally.

Therefore, the core of the usability requirement in this thesis is that users are supported in creating usable models for them based on supportive modelling environments.

5. The Adaptive Modelling Environment

In the previous sections, the general idea of this thesis and requirements derived from the idea and the literature were introduced. This section will introduce the *Adaptive Modelling Environment*, designed and prototyped within this thesis. Therefore, first, we will discuss how such an environment can be used, and afterwards, the metamodel, which will be the foundation for the digital twin from design thinking, followed by the architecture and offered functionality.

The modelling tool will be discussed, but the whole environment needed for using it efficiently in a workshop will be discussed. Therefore, the tangible model can be transformed into the digital one and how the physical workshop environment must be adapted. A particular focus is set on how the digital model can be created, adapted and used, as it is the core of the digital twin from design thinking.

The section will discuss the concept behind the *Adaptive Modelling Environment* and, therefore, will have a broader scope. The prototype and examples mentioned will focus on the archetypical design thinking technique introduced in section 3.3. In short the prototype and examples focus on workshops where paper figures, representing concepts from the domain under study, are placed on tables in workshops to visualise the idea and support the communication between the workshop participants.

However, before we detail the different aspects, a general direction was assumed on how the digital twin from design thinking should be designed. The basic premise of creating the digital model should be as minimalistic as possible. In the most basic form, this is only the identifier or notation of the physical object and its position. In such a case, the *Adaptive Modelling Environment* should also work, as it can already create added value for the users. However, if more sophisticated mechanisms should be used, additional information must be provided, leading to additional tasks the users must perform.

5.1. Adaptive Modelling Environment Procedure

This section focuses on how the *Adaptive Modelling Environment* can be used and what additional tasks must be performed to use it. This includes the preparation and tasks that must be done during the workshop.

5.1.1. Interaction Setting for the Adaptive Modelling Environment

This section will discuss the interaction setting in which the *Adaptive Modelling Environment* will be used and what must be considered to support the users. The aim is for users to be supported in their way of working within a workshop and to minimise the additional

5. The Adaptive Modelling Environment

tasks they must perform. Therefore, the models should be created using tangible means, like paper figures.

Tangible techniques can take on different forms, such as graphics and sketches drawn on whiteboards, sticky notes on a wall and structured in a pre-defined framework or ad-hoc fashion, or tangible figures on a desk. Such techniques also provide the users with freedom, which should be considered when interacting with the *Adaptive Modelling Environment*, as additional ways of visualising can be added in an ad-hoc manner. More such techniques are introduced in section 2.1.

However, the *Adaptive Modelling Environment* not only supports the creation of models through tangible means, but it also offers a graphical user interface (GUI), which can be used for configuration or advanced interactions. This can be used to create and enrich models manually, where the metamodel adaptation capabilities must be integrated. They must be provided in a way that is intuitive to use for users who are not experts in method engineering.

Even though the GUI can be used in workshops, it is more focused on using it afterwards or for preparation. During the workshop, the focus is set on using the tangible means and the GUI only in supporting tasks, which includes triggering the automatic transformation from the tangible to the digital model.

However, to facilitate such an automated transformation, computers must be enabled to understand what the users (e.g., workshop participants) create with the physical aids. The minimum information computers must understand in this context is identifying the physical means used and their relative position to each other. Afterwards, the recognised concepts must be mapped to the metamodel, which is available in the modelling tool, and if the metamodel is sophisticated enough, additional mechanisms can be supported.

Therefore, to support the physical interaction, the *Adaptive Modelling Environment* must be capable of identifying the physical concepts, their relative position to each other and additional information that can be identified. The information that can be concretely derived from the tangible model depends on the prepared transformation, which will be explained later.

Figure 5.1 visualises two tangible techniques that can be used and that the positioning of the figures in the physical and digital representation must not always be completely accurate, but the same semantic can be represented if the concepts are positioned relatively to each other. It shows on the left-hand side two abstract representations of possible tangible models. The top should represent sticky notes on a whiteboard, and the bottom should be paper figures on a table. The right side of the figure visualises digital models in a modelling tool. Both the physical and the digital represent the same idea. Even though the positions of the objects are not entirely the same, the same groups of objects can be identified.

This means the automated transformation must focus on capturing the position precisely, but is sufficient if the relative position from each object to the other is approximately the same as in the physical. This increases the range of possible techniques that can be used to achieve this. Alternatively, it even allows a transformation from a three-dimensional space to a two-dimensional representation, which is often used in conceptual modelling

5.1. Adaptive Modelling Environment Procedure

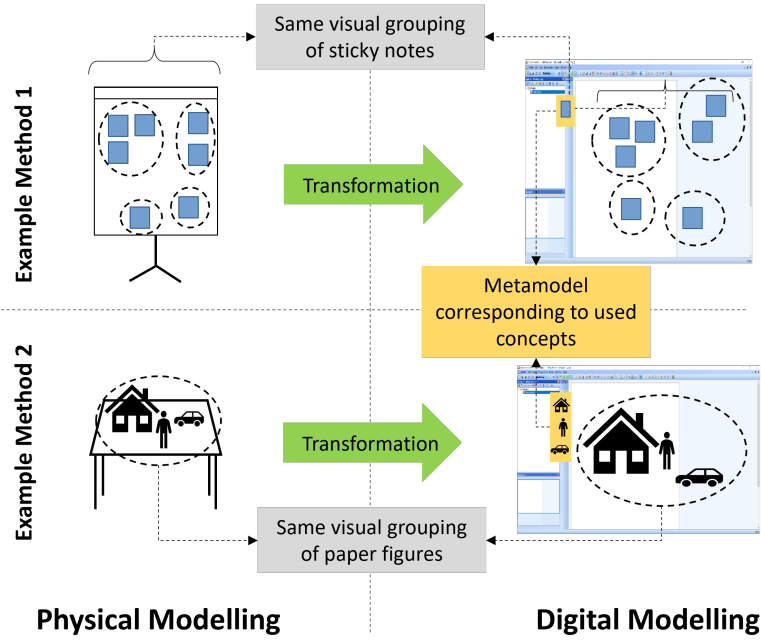


Figure 5.1.: Visual representation of physical and digital models and their encoded spatial information

environments. An example would be that paper figures on a table can have a position influenced by three axes: their height and their position on the surface. This may not be directly mapped to the two-dimensional space of a conceptual modelling tool. However, as we do not need such accuracy, an approximation is sufficient for the recognition part of our transformation.

Therefore, for the interaction, the workshop participants must be able to position the physical objects within the physical space, and the adaptive modelling environment must enable the automated transformation through a trigger coming from the users. The transformation must be able to automatically create the objects based on the available metamodel or with some generic object if no fitting concept is available in the modelling tool.

The yellow boxes in Figure 5.1 represent the metamodel with which the used concepts can be instantiated within the digital model.

If a generic object is used to represent physical objects, which are not available in the metamodel. In that case, they must be abstract enough to represent every object within the used method. They also must be capable of holding additional information for the workshop. For example, they must represent a label for the concept or a digital picture must be added as the graphical representation of the object. Such an object can also support the user in adding new concepts to the metamodel, as it is described in more detail in section 5.1.2.

5. The Adaptive Modelling Environment

5.1.2. Instance Based Metamodelling within the Adaptive Modelling Environment

This section introduced an approach non-experts can use to create models and the corresponding metamodel in a bottom-up fashion. Often, during the creation of the modelling method, a top-down approach is used, where modelling method engineers analyse the domain and then derive the concepts needed to create the models [29]. In some cases, it can be beneficial to not start with the modelling method engineer but to allow domain experts to create models and then derive the modelling method from the models [19, 45]. This can be especially useful if the domain is not yet completely known.

This section will discuss how such a bottom-up approach is implemented in the *Adaptive Modelling Environment*, which we call instance-based metamodelling. The classes used in the modelling method are not created through a form or wizard, but a generic object represents the concept from the system under study. This generic object is then enriched with all the needed information, e.g., which class it should belong to and which attributes should be available. Afterwards, the modelling environment takes an instantiated generic object and derives a class from it, which is added to the metamodel of the tool. Creating the models and metamodels this way should also allow non-experts to use the tool, as they focus first on the model and what they want to represent and only afterwards enhance their models with the meta information.

The general goal is to let the users create models representing their system under study without needing a metamodel or at least no complete metamodel. These instance models are diagrammatic, visually representing the system under study. Here, objects will represent the concepts, including a user-defined notation, for example, provided through a picture. The empty generic objects can be derived from tangible models, but the user must provide additional information about the classes, attributes and notation through the GUI.

Not the whole metamodel should be created from scratch based on the instances, as this would take too much time during the workshop. A part of the metamodel can be defined before the workshop; only the adaptation must be provided during the workshop. For this, the instance based metamodelling approach offers an intuitive way to create and add classes to the metamodel.

Defining the meta aspects like classes, relations, and attributes allows users to reuse them in the same model or other models in the same context. For example, if a user wants to model two scenarios where two different persons interact with a car. Here, the two persons belong to the same concept and have common attributes. If just a visual representation is created, this knowledge is not captured but can only be derived through visualisation, which is possible for humans but difficult for machines. In this way, the modelling environment would have a hard time understanding the sameness of the objects and, therefore, cannot use the information in processing.

Further, attributes must be added to each person individually if attributes are assigned to a mere graphic. If a common class is created for both persons, then this knowledge can be captured centrally and reused if a third person is needed in the models. This example and instance-based metamodelling can be visualised in Figure 5.2.

5.1. Adaptive Modelling Environment Procedure

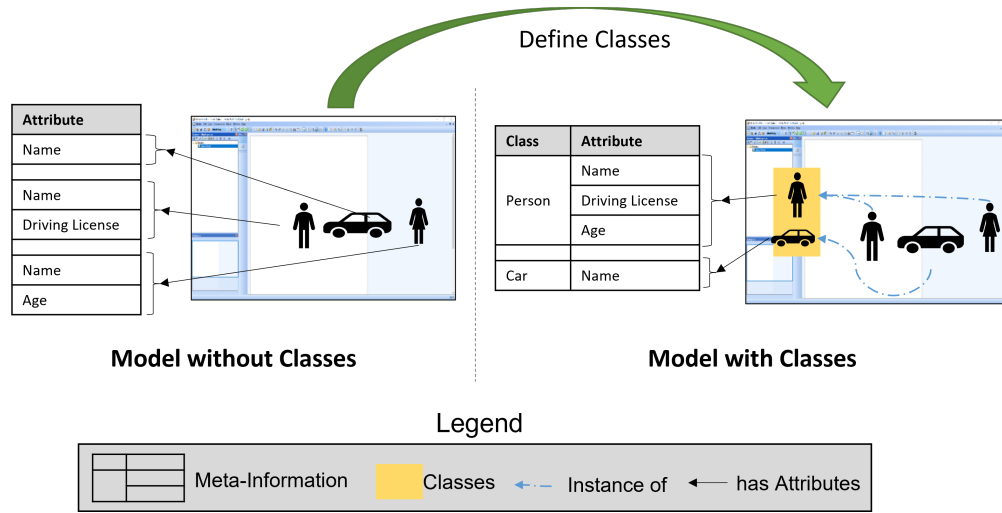


Figure 5.2.: Visualisation of an example for instance based metamodelling

Figure 5.2 shows a model with just an instantiated generic object on the left side, without any meta information available. Different attributes are defined directly to the instances. This model can already be helpful for users and is more than a picture, as it has domain information saved directly to the objects in the form of attributes.

The reusability of the knowledge and the captured domain knowledge itself is limited. Therefore, the right side shows the same model, with the same information captured, but with defined classes. For the car object, one class was created, and the other two objects, representing persons, were grouped under a class person. This class contains then the attributes of both objects. The *Driving License* attribute forms the model's left person, and the *Age* attribute from the right. The yellow box on the right side of the graph represents the set of classes defined for the model. This metamodel can then be used to add objects in the same or another model with the same metamodel.

Users must be able to abstract the classes from objects. Therefore, the modelling tool must be able to create objects that do not belong to a domain class and can take any form and add attributes. In the context of this thesis, this includes an intuitive way of defining a meaningful notation of objects, for example, by adding pictures to them.

As the users of such a modelling tool are usually not experienced method engineers, the modelling tool should support the creation of classes by taking the information provided for one object and translating this into a class. In the example of Figure 5.2, this would mean that the graphical representation of one person object is used to create the class. The assigned object attributes should be used as the names for the class attributes. Additionally, the graphic provided for the object can be taken as the class notation. After the class is created, the instance of the generic object must be changed to an instance of the new class.

If the second-person object is transformed into a class, it would be beneficial if it is added to the existing person class. This means the second object's attributes must be

5. The Adaptive Modelling Environment

integrated into the set of existing attributes. Further, a decision should be made on whether the notation of the second object stays the same or if it takes the notation from the class. If the notation of the object should be kept, the class must be able to handle subtypes, provided the notation of the first object should also be kept.

Therefore, even though the class creation process is automated, decisions must be made by taking the information from the instances. The human user should make decisions like how multiple notations for the class are handled, as this depends on the domain and can not be decided automatically.

Therefore, the modelling tool needs the ability to automatically derive classes from instances and integrate instances in existing classes with all their defined attributes.

This approach does not aim to create highly sophisticated metamodels, as is used in meticulously designed modelling methods by experts. Such modelling methods and the corresponding modelling tools are often evaluated and improved to maximise the model value. This is a cumbersome undertaking that needs not only domain expertise but also advanced knowledge in method engineering.

Nonetheless, adding meta-information as a metamodel eases the understanding of the created models for humans and machines. An explicitly defined metamodel within a modelling environment enables its utilisation through automated processing.

Suppose the adhoc created metamodels should be used for automated processing. In that case, the applied mechanisms must either be adaptable, implemented on a generic level to be used with multiple metamodels, or utilise semantic-rich mechanisms that can use the limited information provided and provide added value to the user.

The instance based metamodels can be further developed and enhanced to create a meticulously designed metamodel and a corresponding modelling method. Therefore, the instance based metamodels can be used to explore the domain and create a first draft for a later modelling method. In this context, the domain and usefulness of the new modelling method must be analysed to decide if this effort is worth the benefits. Here, the first draft of the modelling method, which can be created with the instance based metamodelling approach, can support this evaluation by offering a prototype.

5.1.3. Using the Adaptive Modelling Environment in Physical Workshops

This section introduces how the *Adaptive Modelling Environment*, including the instance based metamodeling approach, can be used in physical design thinking workshops. This includes the roles that must be assumed and how the preparation, usage, and exploitation can be done when users use both tangible and digital modelling.

However, this section will not focus on how individual design thinking techniques can be used within workshops but on how the *Adaptive Modelling Environment* can be used, thereby focusing on setting up and using the automatic transformation from the tangible object to the design thinking digital twin.

Further, the aim is not to provide a rigorous method but guidelines on what tasks must be done and some temporal dependencies between them. This was done as how it is used depends on the users and the chosen design thinking technique and to follow the overall

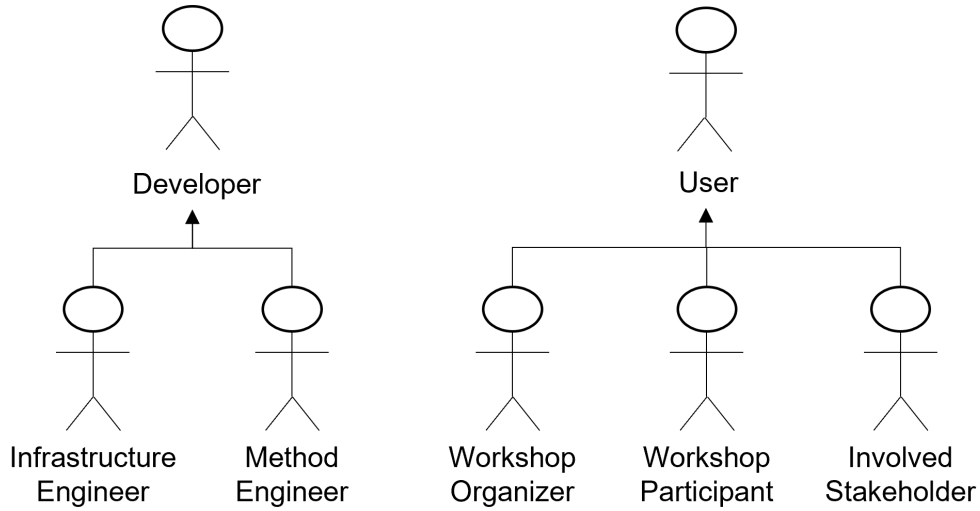


Figure 5.3.: Visualisation of the roles involved in using the *Adaptive Modelling Environment* in workshops

aim of this thesis to provide an environment that can be used fitting to the user's needs and not to enforce a particular way of working.

Before the proposed usage guideline is introduced in section 5.1.3.2, the roles of involved users will be discussed in section 5.1.3.1.

5.1.3.1. Roles For Using the Adaptive Modelling Environment

In the adaptive modelling environment context, multiple persons are involved and can take on different roles. Based on their roles, the persons must fulfil different tasks.

The roles are not mutually exclusive, and one person can take on multiple roles. For example, a workshop participant can also be an involved stakeholder who needs the information afterwards. Additionally, multiple persons can take one role simultaneously, as a workshop will have more than one participant.

A visual representation of the available roles can be seen in Figure 5.3. The stick figures represent the different roles that are defined. The two roles at the top are abstract superclasses used to structure the available roles. The roles on the bottom are those which can be directly taken. The arrows visualise the membership of the concrete roles to the abstract super roles.

Involved persons can roughly take on two roles: *developer* and *user*. The first provides the software and the technical infrastructure to facilitate design thinking digital twins. The roles combined in *user* focus on the persons who use the adaptive modelling environment for their purposes.

5.1.3.1.1. Infrastructure Engineer are people who develop the applications used in the workshop. One central component must be implemented is the modelling tool with

5. The Adaptive Modelling Environment

which the models will be created. As it is not efficient to start implementing modelling tools from scratch every time, a metamodeling platform must be provided [2]. This platform must also be implemented and further developed. This is a critical task of the *infrastructure developer*.

Not only must the modelling tool be implemented, but the infrastructure must also be developed to transform the physical objects into their digital representation and to process and utilise the design thinking of the digital twin. Therefore, the *infrastructure engineer* role is also responsible for implementing the applications, which are used for recognising the physical objects, mapping them to the metamodel and mechanisms that can be used within and after a workshop. Therefore, such implementations must also consider the integration of these different components.

5.1.3.1.2. Method Engineer is the role which creates the method which can be used in the workshop and the corresponding modelling environment. The method engineer in this context is not focused on identifying and implementing all possible domain concepts, as the users do this before, after, or during a workshop. Nonetheless, the modelling method's basic structure and mechanisms must be created. For example, basic attributes that every domain object must have must be implemented so that it can be used within the modelling method. For the archetypical case, every domain concept has a type which defines the subtypes of the concept, and generic objects can be created using a picture as notation.

Therefore, the method engineer will use the metamodeling platform created by the infrastructure developer and create the basic structure of concepts needed for the specific method. These can be different modeltypes or concepts needed independently from the domain. For example, a concept that links to another modeltype.

Additionally, the components from the *infrastructure developer*, which handle the recognition and mapping of the physical objects to the digital objects, must be tailored to fit the concrete method. For example, the mapping component must be tailored so that the recognition information fits the modelling method and can be automatically shown in the modelling tool.

5.1.3.1.3. Workshop Organizer prepares and leads the workshop. The person taking on this role must not be deeply familiar with the domain, only so far that the *Adaptive Modelling Environment* can be prepared so the participants can work efficiently. For example, they are preparing an initial set of concepts that can be used within the workshop.

During the workshop, the organiser must explain the method, the environment, and guide its usage. Additionally, an organiser must operate the modelling environment, for example, triggering the automated transformation from the physical to the digital objects. Alternatively, other functionalities can be executed to support the creation of the innovative idea. Otherwise, the workshop organiser must support the cooperation and flow of ideas, for example, by asking questions.

5.1.3.1.4. Workshop Participant are the people who contribute expert and domain knowledge in the workshop and design the solution, which should later be implemented or at least further analysed. Therefore, they must discuss with each other and use the physical objects to design the idea. The exact way the workshop's result looks depends on the applied method.

5.1.3.1.5. Involved Stakeholder are those involved in implementing the workshop results or with other interests in the workshop's outcome. These include people who did not participate in the workshop themselves. Usually, the number of workshop participants is limited to under ten people [17, 18], but for the implementation often more people are needed. These people involved with the workshop result are taking on the role of *involved stakeholder*.

Examples of involved stakeholders are the software engineers who implement the idea, marketing experts who must evaluate the marketability, law experts who check the legal feasibility, or project managers who monitor the progress. The involved stakeholders will receive the knowledge captured in the models and use this in their own work or provide feedback which can be used for the implementation.

5.1.3.2. Usage Guideline for Tool-Aided Adaptive Modelling in Workshops

This section discusses proposed guidelines for using the *Adaptive Modelling Environment* in workshops. We called it proposed guidelines, which should not be seen as a rigorous procedure that must be followed meticulously but define the tasks and their temporal dependencies needed to use the environment. How the procedure is defined, and the environment's overall design should fit the freedom its users need in the workshop. Therefore, the guidelines and subsequent adaptive modelling environment are created with freedom in mind.

Nevertheless, this section introduces a procedure that can be seen as input for usage and should guide users. In this context, not only will the usage in the workshop be discussed, but also how such a workshop can be prepared and what can be done after a workshop. Figure 5.4 visualizes the procedure.

The proposed procedure is divided into three main phases: preparation, design and exploitation. The central one is the design phase, where the workshops are held, and their results are captured. The preparation phase focuses on preparing the workshop by creating the tangible figures, establishing the recognition system for automated transformation and preparing the modelling tool. Lastly, in the exploitation phase, we will discuss how the models from the workshop can be further utilised to support the implementation of the created idea.

5.1.3.2.1. Preparation Phase: The preparation phase primarily falls under the purview of the workshop organiser and encompasses two key tasks, namely understanding the domain and creating the icon library (see section 5.4.1.1.4 for more information on the icon library). The domain must be understood to a degree so that useful objects and

5. The Adaptive Modelling Environment

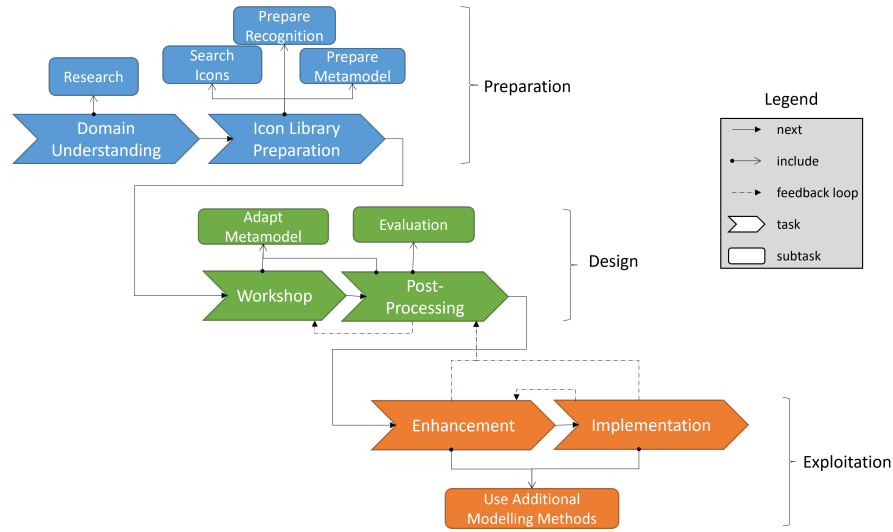


Figure 5.4.: Visualisation of the guideline for using the *Adaptive Modelling Environment* in workshops

their figures can be prepared. Here, the workshop organiser should interact with the participants to get an idea. One possibility is that the participants must provide a short abstract on what they want to achieve in the workshop. However, the goal is not for the organiser to find every needed concept but to provide a sophisticated foundation so the participants can start the workshop.

After the domain is understood, possible objects that can be used in the workshop immediately must be prepared. A core task here is to find fitting icons for the identified domain concept. An icon is a picture that represents an object. Here, an icon should be searched that semantically represents the object and is interpretable by the workshop participants, which correlates with the *semantic transparency* principle introduced by [40]. Based on the icons, the metamodel and the recognition must be prepared (see section 5.4.1.1.4 for more information). Additionally, the icons must be prepared as physical objects (e.g., printed and cut out) for the workshop. To improve the mechanisms that are later executed upon the prepared domain concepts, the icons and additional information like possible relations, attributes, or other meta information can be prepared.

The identified concepts must also be prepared to be used in the modelling environment. Therefore, they must be added to the metamodel, which must at least contain the identifier used by the recognition to allow mapping from the physical objects identified during the workshop to model objects. If additional information, like attributes or relations, is provided, it can be used within the automatic transformation. When the icon library is prepared, the workshop can be held.

5.1.3.2.2. Design Phase: In the design phase, the innovative idea is created and designed on a high abstraction level. This design should capture the essence of the new

5.1. Adaptive Modelling Environment Procedure

idea, including its general goal and the important concepts. The design should not be detailed enough to be directly implemented, but it should be on a high abstraction level as a communication facilitator between experts from different backgrounds.

The workshops are held in this phase, where experts come together and co-create novel ideas. Here, the physical method, prepared by the workshop organiser in the *preparation phase*, will be applied to support the cooperation of the workshop participants. The design thinking technique will result in understanding the novel idea within the workshop's participants and tangible models, capturing the participants' knowledge. A design thinking digital twin will be created based on the tangible model to support the later exploitation of the created knowledge.

Therefore, the design thinking technique is applied, and once a stage is reached that should be captured, the automated transformation is triggered. When it is triggered concretely depends on the technique used and the workshop. After creating a digital model, it should be checked, and crucial information should be added to the fly to avoid losing it.

This information can be saved in the model or metamodel. Therefore, not only should the provided metamodel from the preparation phase be used, but the metamodel should also be adapted to fit the needs. This can be done by using a wizard to adapt the metamodel or by using the instance-based modelling approach introduced in section 5.1.2.

The available models and the corresponding metamodel can then be used during the workshop to trigger supportive functionality, inspiring new ideas within the participants. For example, checking semantic constraints based on the metamodel, sending a picture of the idea to stakeholders and instantly gathering their feedback or using AI to let the system suggest additional information.

Afterwards, the post-processing task, which contains improving the workshop models, is executed. The main goal of this step is to improve the workshop model. This can be done by reworking the models to make them visually more appealing or harmonise the naming and writing style of the texts added to the models and their objects. Additionally, the icons of the objects can be adapted if needed. For example, this can be necessary if a new icon was spontaneously added during the workshop, but it is not completely fitting. Such icons can then be changed during the post-processing step.

The first post-processing can be done at the end of the workshop, with all participants going through the models again for a first evaluation of the idea. During this evaluation, the models can be adapted if needed.

However, post-processing can also be done after the workshop is finished. To further evaluate the models, they can be sent to involved stakeholders for feedback. This can be done by sending the models to the stakeholders so that they can import them into their modelling tool. Alternatively, a picture can be sent to many people to gather feedback. Here, the modelling environment can support the users by supporting the posting of a picture to social media and later retrieving the answers and analysing them. Such an approach is discussed in section 3.4.2.

If a serious mistake or false assumption is identified during the post-processing, the workshop model can be adapted, or even new workshops can be conducted. At the end

5. The Adaptive Modelling Environment

of the design phase, finished models should be available, which can be used to start the design implementation.

5.1.3.2.3. Exploitation Phase: This phase focuses on using the ideas created after the workshop and finalising the models. Therefore, there is now a sophisticated understanding within the participants' minds and the enriched processable artefacts in the form of digital conceptual models.

This knowledge and the processable artefacts can then be used to implement the novel idea. In this section, the concept of implementation is understood broader than software creation. Here, aspects like organisational restructuring or work instructions are also included. Anything that makes the models from the workshop executable in a real-world scenario.

How the idea is implemented depends on the user. However, due to the nature of this thesis, we will assume that other modelling methods are used during the design and implementation of ideas, as it is often done. The additional modelling methods used are not pre-defined and should be tailored to the users' needs and experience, including newly created or already established methods. Therefore, there are no concrete steps on using specific methods, like BPMN, together with the workshop models, but general guidelines are discussed.

Therefore, the exploitation phase starts with *enhancement*, which refers to the created idea. Even though this included enhancing and adapting the models from the workshop, it does not focus on them. The goal is to make the created idea more sophisticated, as its design in the workshop is on a high abstraction level, which is not detailed enough to implement. Here, other modelling methods should be used to enhance the design created during the workshop.

The so-created models should not be used completely separately from the workshop models, as they are all related. Therefore, a linkage between the additional models and the workshop models should be established. This allows the workshop model to establish relations between the additionally created models, and the workshop model can be used as a gateway between them. How such a linkage can be established is discussed in section 5.3.7.

However, the workshop models can not only be linked to models from other modelling methods but can also provide input for them. As the additional models will describe the same idea from another perspective or on another abstraction level, parts of the workshop models can be reused as a starting point. Thereby, the creation of the additional model can be done manually if the user analyses the workshop models, understands them and then creates them by hand. This can be cumbersome for users, as they must recreate the knowledge already available in the workshop model.

This pain point can be alleviated by enabling the modelling environment to transform the workshop model into a starting point for the additional model. Therefore, the modelling environment must understand the additional modelling method and must apply a mapping between the metamodel of the workshop model and the metamodel of the additional modelling method. How this can be done will be discussed in section 5.4.2.

5.2. A Metamodel for the *Adaptive Modelling Environment*

The users can then enhance the starting points from the model transformation so that the workshop design can be implemented. This model can then also be linked to the workshop model. Therefore, the enhancement can be achieved by linking additional models to workshop models and transforming them.

Even if no additional modelling methods are used for the implementation, the knowledge of the workshop must be transformed to fit the chosen implementation approach. However, the transformation does not mean that the models from the workshop should be ignored once they are transformed. The workshop models should also be used during the transformation and implementation phase, providing feedback on the design defined in the workshop. This way, the high-level design models can be adapted to show the latest version.

No concrete procedure is provided for the implementation task, as it must be adapted to the concrete case. Established implementation paradigms, like software engineering or model-driven software development, can be used here. However, concrete implementation is not within the scope of this thesis.

5.2. A Metamodel for the *Adaptive Modelling Environment*

An abstract metamodel must provide a set of fundamental concepts to enable the conceptualised *Adaptive Modelling Environment* to create and use modelling methods for uncertain domains flexibly. Behaviour and functionality of the modelling methods can then be defined based on the fundamental concepts, and the concrete metamodels can then be derived from them and reused the behaviour [2].

This section will introduce the fundamental concepts used in this thesis to allow modelling in uncertain domains and discuss how they were designed. These fundamental concepts will be proposed as an abstract metamodel, which can then be used to derive the metamodels for concrete modelling methods, which can be used in workshops to create novel ideas. The functionality and the conceptualisation of the rest of the modelling environment will be based on the here-introduced abstract metamodel.

To create the abstract metamodel, first, a domain was chosen that contains the two main aspects that this work should cover. This is an uncertain domain, meaning that we search for an approach used in situations where the context and goal are unclear and should be used by groups working together in a physical space. Therefore, design thinking was chosen as the broad domain, and physical workshops within design thinking were chosen as the concrete domain. As input for the abstract metamodel, tangible creative or design thinking techniques used in physical workshops were taken. Tangible in this context means that the design thinking technique is used to create a physical artefact, representing the knowledge created in the workshop. Therefore, sticky notes on a canvas or paper figures are used to create this physical artefact in the workshop. Examples of such tangible design thinking techniques are introduced in section 2.1.2.

The goal here is not to create an abstract metamodel for a specific design thinking technique but to extract common patterns along multiple techniques and then integrate them into an abstract metamodel, which can be used as the basis for multiple modelling

5. The Adaptive Modelling Environment

methods, representing the design thinking technique.

To build the foundation of the abstract metamodel, the techniques listed in Table 5.1 were analysed, commonalities were extracted, and then the abstract metamodel was designed. As no “best” or most used design thinking techniques could be identified and analysing all available techniques is not feasible, a selection of commonly used techniques was searched and taken as the starting point for defining the abstract metamodel. The selection was based on covering typical techniques. Through this, a wide variety of techniques should be supported. However, the abstract metamodel can also be adapted to cover additional techniques.

We make a broad distinction is made between text-based and graphic-based techniques. Text-based concepts are represented during the workshop as text, often written on sticky notes. Graphic-based use figures, icons or other graphical representation of concepts in the workshop.

5.2.1. Fundamental Structures in Design Thinking Techniques

The following fundamental structures were extracted from the analysed design thinking techniques (see Table 5.1). A fundamental structure in this context means the generic structure used to organise the knowledge created in a workshop within a design thinking technique. The structure also influences the visualisation of the physical artefact created in the workshop. One technique can use multiple fundamental structures in combination.

The different fundamental structures are then consolidated in section 5.2.2 to propose an abstract metamodel for design thinking techniques. This abstract metamodel can be used to derive metamodels for concrete design thinking methods. The abstract metamodel is the basic structure and provides the fundamental concepts for the computer-aided modelling environment so that design thinking techniques can be defined and used with the modelling environment.

To achieve this in section 5.2.3, the introduced abstract metamodel will be adapted to an implementation-oriented one, meaning that the structure and the available information will be more tailored so that it can be used in an implementation, and the needed functionality can be applied to it.

However, the abstracted metamodel should not be understood as an unchangeable core, which has to be used for each design thinking method. It should be adaptable and extendable to meet new requirements coming from new design thinking techniques.

The visualisations of the fundamental structures and the metamodel were created with the *CoChaCo* modelling method (see section A.4.13 for more information).

5.2.1.1. Concept Types

The first fundamental structure of this thesis is called *concept types* and is visualised in Figure 5.5. This basic structure focuses on defining the basic concepts which must be available to create concrete metamodels for design thinking techniques.

5.2. A Metamodel for the Adaptive Modelling Environment

Text-based	Graphic-based
Business Model Canvas (see section 2.1.2.1)	SAP Scenes TM (see section 2.1.2.3)
Service Blue Print (see section 2.1.2.2)	Desktop System Mapping [182]
Customer Journey [182]	Town Planning [183]
Stakeholder Map [182]	Lego [©] Serious Play [©] [184]
Persona [183]	
Value Exchange Map [182]	
Empathy Map [183]	
Brainstorming [183]	
Feedback Capture Grid [183]	
Cause and Effect Diagram [185]	
Value Proposition Map [186]	

Table 5.1.: List of design thinking techniques which were analysed to build the abstract metamodel

5. The Adaptive Modelling Environment

Domain concept is an abstract class in which subclasses can represent objects from the domain. This concept has structural purposes to reduce the visualisation of similar attributes and relations. The *relates* connector represents the capability to establish a relation between the objects in the design thinking technique.

One crucial aspect of why design thinking techniques are applied is that they help structure the knowledge gathered in the workshop. This is achieved through their visual representation on the one hand and through categorising the objects alongside common features. The fundamental structure allows the assignment of a category through the *type* attribute.

A concept of the workshop's domain can be represented in three ways, represented through the three concepts which specialise *domain concept*. *Text-based concept* and *graphic-based concept* are the two main ways objects or singular concepts are represented in workshops. The first one is that a written term is used for representation. This is often done through sticky notes on which the term is written. The *graphic-based concept* uses a picture or other visual representation within the workshop. The participants will interpret this visualisation and assign meaning to it.

A container represents a collection of concepts which belong together. This can be a group of individuals or an entity of different concepts. Whether the containers are pre-defined by the method or worked out in the workshop does not matter for this fundamental structure. A container can contain all domain concepts, meaning it can also contain other containers. Containers will also have a visualisation to easily identify which concepts belong to which container.

In the visualisation, the types can be encoded by the container to which they belong or through visual cues. For example, different colours of sticky notes or the visual positioning of a concept in a container can be used to represent a type. However, types can also be implicitly available, encoded into the graphical or textual representation, which users interpret and then derive a type. For example, a picture of a bus driver is assigned to a group of people.

The *canvas* is a special container which provides a context for a collection of domain concepts. It represents one artefact that was created in the workshop and is an aggregation of the identified concepts. The goal of the canvas aligns with the specific design thinking technique for which it is created. The canvas in the fundamental structure differs from a canvas for a specific design thinking technique. Within the physical workshop, the canvas of this fundamental structure can be understood as a limited physical space, which is used to position the physical representations of the identified concepts.

5.2.1.2. Concept Description

Within a design thinking technique, physical objects represent concepts from the workshop's domain. However, the concepts are more than the presentation for the participants, as they assign them properties in the workshop's context. Such properties stem from the discussions and must be recorded to capture the whole idea. Therefore, this fundamental structure focuses on how such additional information can be saved to the domain concepts, and it is represented in Figure 5.6.

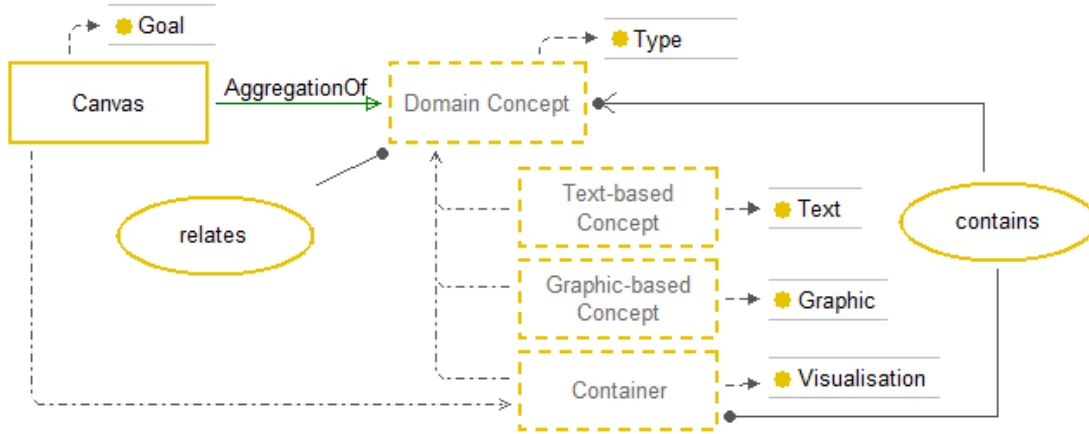


Figure 5.5.: Fundamental structure of design thinking techniques: Concept Types

First, the concept must be visualised, which can be done through a term or a graphic, as discussed in section 5.2.1.1, for which the *Visualisation* characteristic can be used. Additionally, a concept should have a label, which can be used to identify it and provide additional information that is interpretable by the participants. This attribute is represented through the *denomination* characteristic in Figure 5.6. The category of a concept, to which they are often assigned in design thinking techniques, is encoded through the *type* characteristics.

The attributes and information that enhance the saved knowledge can be saved in the *descriptive attributes*, representing a collection of attributes that must be defined for the concept used in the design thinking technique. The concrete attributes must be defined for the domain of the workshop. Further, descriptive attributes can be grouped via a *attribute category*. These categories are assigned to domain concepts and contain several attributes that belong thematically together.

Attributes can be defined as mandatory, allowing the method engineer to express that they must be filled out to complete the concept in the design thinking technique. However, mandatory attributes should not disturb the users in their way of working but allow them to check if they have added important information. This means that the filling in of mandatory attributes should not be forced after a concept is created, but users should be informed that important information is missing.

The domain concept introduced in this fundamental structure is the same as the one introduced in *concept types* (section 5.2.1.1). This means that the description capabilities introduced in this section can also be applied to *text-based concepts*, *graphic-based concept* and *containers*, as introduced in the *concept types*.

5.2.1.3. Nested Canvas

Concepts represented within design thinking techniques represent objects from the workshop's domain. These can sometimes be complex themselves and need additional un-

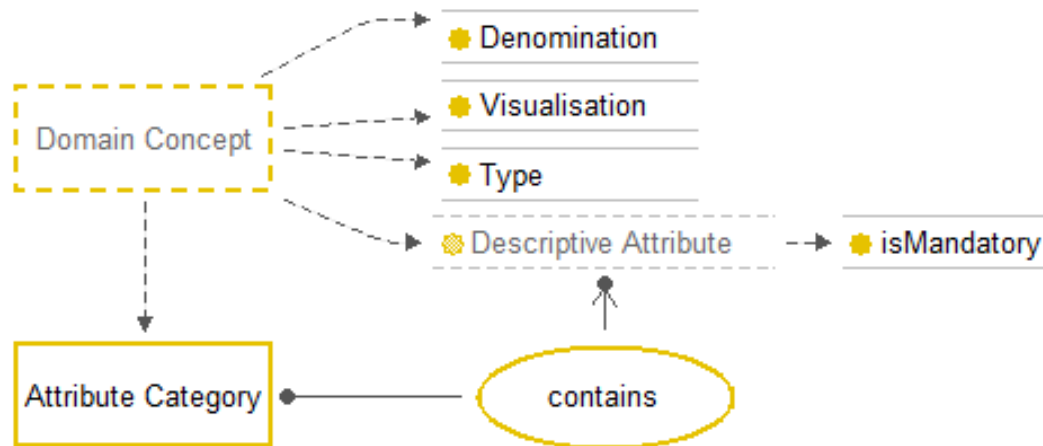


Figure 5.6.: Fundamental structure of design thinking techniques: Concept Description

derstanding. Therefore, there is a need for an object used in one canvas to be further detailed within another canvas. This linkage between canvases is represented through this fundamental structure and is visualised in Figure 5.7.

A canvas in this context consists of a collection of domain concepts structured and represented as guided by one design thinking technique. Therefore, it represents the artefact created with the applied design thinking technique.

A link between a domain object in one canvas, instantiated from *domain concept* and another canvas, must be established to achieve the nesting of canvases. It does not matter if the linked canvas is from the same design thinking technique or a different one. The idea behind this structure is that a concept used in one canvas is described in detail in another. For example, if a human is represented on one canvas, a persona or an empathy map can describe this human in more detail.

5.2.1.4. Layouts of Design Thinking Techniques

Design thinking techniques use spatial arrangement to encode information from a domain within a workshop by providing meaningful relations between the identified concepts based on their position. The arrangement and position mentioned here refer to where the concepts are positioned on the canvas. For example, where on the table a figure is placed during the workshop.

Meaning can be derived by the relative position of the objects to each other and the interpretation of this proximity. This interpretation depends on the concrete case and the agent interpreting the visual representation. To further increase the meaning of the position, design thinking techniques use pre-defined areas, which assign an object positioned within such an area to a group, category, or property.

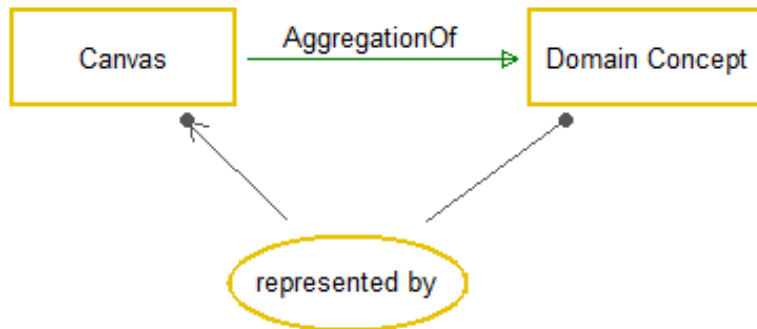


Figure 5.7.: Fundamental structure of design thinking techniques: Nested Canvas

The pre-defined areas can also have the meaning assigned to them, and the proximity of the areas can also encode additional meaning. Therefore, categories next to each other can be related to each other. Further, the size of the predefined area can encode their importance.

If pre-defined areas are used in a design thinking technique, a template is usually provided containing the pre-defined areas that can be used. This template can then be used within a workshop to assign concepts from a workshop's domain to the areas.

Additionally, meaning can be integrated into the visualisation or structure of the created artefact using a procedure within the design thinking technique that creates a certain structure, like a sequence or tree. Then, the position of the domain concept within this structure also contains meaning.

Below, we will introduce different versions of such layouts or structures used by design thinking techniques introduced in Table 5.1. They will contain some common elements, like domain concepts, which are abstract placeholders for the objects identified for a specific workshop, or a canvas, a self-contained aggregation of domain concepts created for a specific purpose.

5.2.1.4.1. Sequence Sequences are often used to encode the passage of time and provide a step-by-step representation. The steps do not encode concrete time jumps, for example, one hour or two days, but represent a logical order in which the step happens over time. However, a sequence does not necessarily have to be based on a temporal order but on any property of the concepts which can be ordered.

The abstraction of the fundamental structure is visualised in Figure 5.8 and contains a canvas, an aggregation of domain concepts. The sequential order can be visualised through their position or a dedicated symbol. Position-based means that they are graphically ordered from one side to another, e.g., from left to right. An example of a dedicated symbol would be an arrow pointing from one domain concept to another.

For the position-based arrangement, the objects must contain a position that can be interpreted meaningfully. A dedicated symbol can be interpreted as an established

5. The Adaptive Modelling Environment

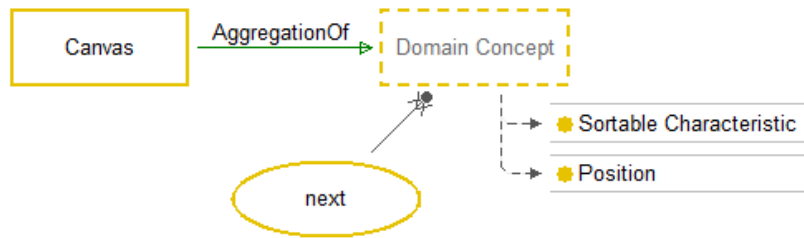


Figure 5.8.: Fundamental structure of design thinking techniques: Sequence

relation between two objects. To create or interpret the order, the domain concepts must have a sortable characteristic assigned to them, meaning that the characteristics can be compared, and an order can be established based on them. For example, the values of the characteristics are smaller or bigger than others. Using a dedicated symbol or the position is not an either-or decision, but both can be used in parallel.

Utilising this structure, the objects used can be sorted and automatically visualised. If a concept is added later, it can be placed in the right spot based on its sortable characteristics. Additionally, the two encodings can be transformed into each other, e.g., the dedicated symbols can be added automatically based on their position.

To ensure that the used structure is a sequence, it must be checked if each domain concept has one ingoing and one outgoing next relation, except one domain concept is allowed without an ingoing (the start). One is allowed without an outgoing (the end). The domain concepts must be aligned based on a line along one axis for the position-based representation. On this axis, two concepts cannot be represented in parallel.

5.2.1.4.2. Tree Tree structures can be used to start at a general level and further specialise a concept in more detail. This structure can also be visualised using a design thinking technique. The fundamental structure is visualised in Figure 5.9.

A tree consists of a node, which is the *domain concept* in our case and a relation defining which node belongs to which parent node. For this, the *childOf* relation was introduced in this fundamental structure. The *childOf* relation points from the child node to the parent node. Each node must have one *childOf* relation to another node, except the root node. Therefore, in a tree, only one node is allowed not to have a parent node. A node is also not allowed to have multiple parent nodes. The nodes themselves must contain information from the domain, which was represented in the fundamental structure through the *label* characteristic of the domain concept.

Having this basic structure allows us to process it to extract additional information. For example, the most specific defined information can be gathered by searching for those nodes that are not the parent of another node, called leaves in the tree structures. Alternatively, a list or sequence can be created, containing all the nodes from a leaf (or another chosen one) to the root and containing the path of specialisation for this domain

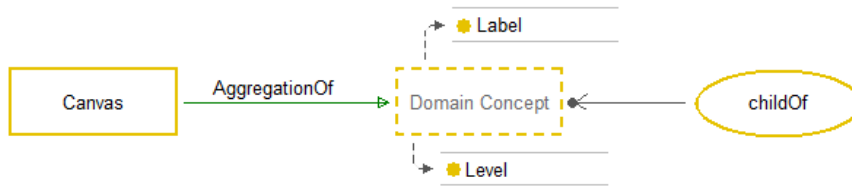


Figure 5.9.: Fundamental structure of design thinking techniques: Tree

concept.

Based on the number of parent nodes, a level can also be assigned to the domain concepts, which allows easy comparison of nodes on the same specialisation level.

The fundamental structure also contains a canvas, representing one tree created for a specific purpose.

5.2.1.4.3. Layer Layers can be used to group domain contents and assign these groups an order, which delivers meaning for a concrete case. For example, layers can represent abstraction levels or group components of an information system based on the distance to the user. For example, the first layer is a user interface, the second is a client application, and these layers are continued till the back-end data storage is reached.

The fundamental structure for layers is visualised in Figure 5.10. A layer containing objects from the domain is represented as *container* to keep the terminology consistent in the other fundamental structures. A canvas is a container containing all the layers that belong to one case.

Layers have a meaning assigned to them, which they pass on to the contained domain concepts. This meaning is provided by the *type* characteristic, defined by the used design thinking technique or the user creating the layer.

The visual representation of layers is presented in an order which can be visually represented. Therefore, a layer must be assigned coordinates and size to be visually presented. The size of a layer can also be used as an indicator of its importance.

The visual order of the layers can be interpreted as above or before or the counterparts below or after. This depends on whether the layers are represented horizontally or vertically. This relation between layers can also be made explicitly available through a relation represented in this fundamental concept through the *above/before* relation.

To allow the creation of a template, the layers must also pose a pre-defined position containing their position and size. This pre-defined structure and a provided type capture the basic meaning of the template.

Layers can also have different shapes, like rectangles or circles. Rectangular can be used to show the layers above or next to each other, and the user can interpret them along the horizontal or vertical axis. However, layers can also be aligned circularly. Then, the layers go from a core concept in the middle outward, representing each layer with a

5. The Adaptive Modelling Environment

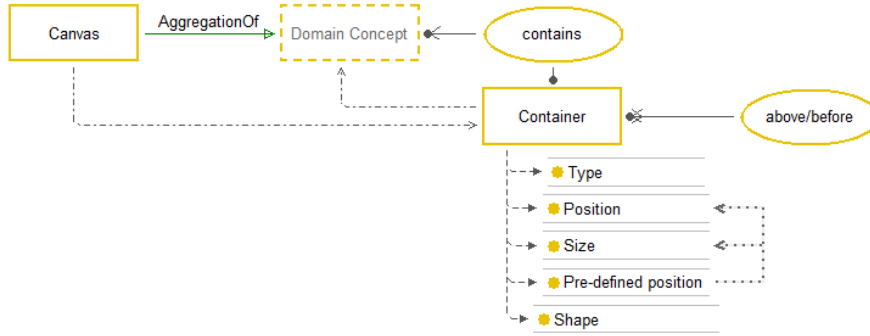


Figure 5.10.: Fundamental structure of design thinking techniques: Layer

bigger circle. Outward in this context means that the concept in this layer has a greater distance to the core than the ones in the inner circles. The meaning of the layers within the design thinking technique provides this distance.

5.2.1.4.4. Method-defined Layout However, design thinking techniques must not utilise pre-established structures, like the ones defined above, to define their internal structure. They can also utilise their own and define their meaning to the structure. These structures are the focus of this fundamental structure, which is visualised in Figure 5.11.

The canvas is again used to define the concepts and containers used within the design thinking technique. The canvas can also be seen as the provided template, which the users can fill out with their specific case. The concrete objects identified during the usage must be mapped to the provided structure. Therefore, the technique will provide the structure of containers to which the objects can be designed. However, this container must not only build a group of objects but also possess meaning, provided through the *type* characteristic.

To use it as a template, the technique's creator must be able to define the used containers and their meaning. Additionally, to utilise spatial information that can be gained from the positions of the pre-defined containers, the creator should also be able to define the position of the containers. This fundamental structure is done by providing the *pre-defined position* characteristic based on a container's size and coordinates.

5.2.1.4.5. Unrestricted This fundamental structure was added for completion, as not all design thinking techniques provide a template or structure on how the gathered information is structured or visualised. The design thinking technique can guide what information should be gathered and what important aspects are needed.

To use such techniques, users need basic concepts to visually represent their knowledge. These basic concepts are visualised in Figure 5.12 and contain a canvas and objects to visualise a specific case. For visual representation, the domain concepts need coordinates and size. Connections may need to be established within the visualisation to further capture knowledge in the workshop. This fundamental structure encodes this through

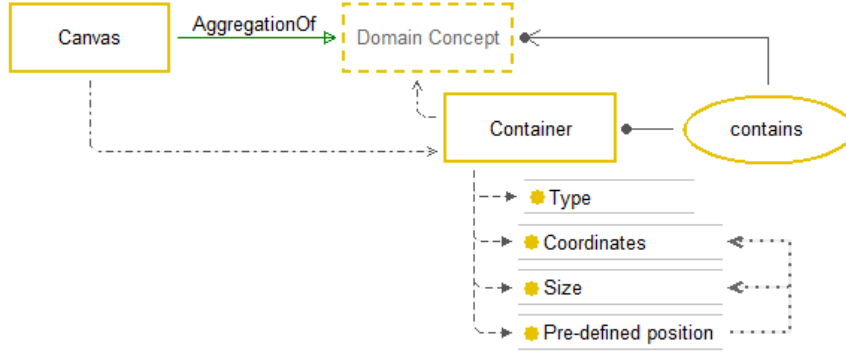


Figure 5.11.: Fundamental structure of design thinking techniques: Method Defined Layout

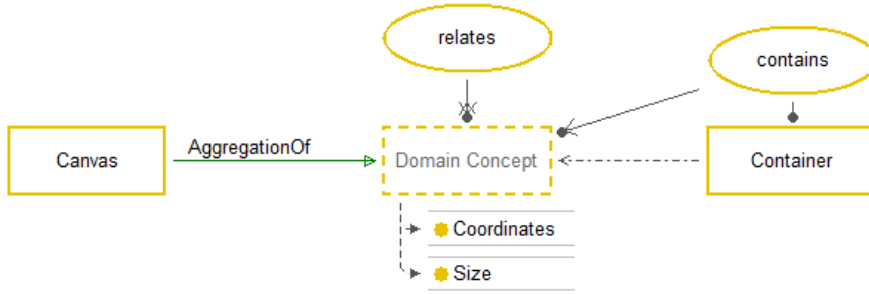


Figure 5.12.: Fundamental structure of design thinking techniques: Unrestricted

the abstract *relates* connector.

Additionally, users should be able to group similar objects. This can be done by positioning them near each other or using a separate visualisation for the group. This fundamental structure offers the *container* concept, which can contain other domain concepts and defines a group of objects in this way.

5.2.2. A Metamodel for the Adaptive Modelling Environment

Based on the discussed fundamental structures of design thinking techniques (see section 5.2.1), the metamodel for the adaptive modelling environment will be introduced. The goal is to provide an abstract metamodel from which metamodels can be derived for concrete design thinking techniques within the adaptive modelling environment. Therefore, a concrete metamodel for a number of design thinking techniques will not be introduced here.

Within this thesis, the abstract metamodel is the foundation provided, which users can use to create their modelling methods for design thinking techniques and make it usable within the adaptive modelling environment. Therefore, the fundamental structures identified in section 5.2.1 will be integrated to form a common metamodel, which is

5. The Adaptive Modelling Environment

visualised in Figure 5.13. This figure can be seen in landscape format in the appendix A.2 in Figure A.1. The metamodel is visualised with the CoChaCo modelling method (see section A.4.13 for more information)

At the core of this metamodel is the domain concept, which represents the most abstract element that can be used to represent objects from the workshop domain. It allows the reuse of its characteristics and connectors for its derived concepts. The domain concept offers different characteristics that can be used to describe its instances.

First, the *denomination* holds a label which provides a human-understandable identification of the instantiated objects. The *type* characteristic can be used to define the category of an identified object of the design thinking technique. The category can be assigned based on the object's position, e.g., if it is contained in a container, or the users can directly assign it. *Coordinates* and *size* are used to capture the position the object has in the visual presentation and is used by *pre-defined position*, which can be used to place some objects on a canvas automatically. This is mainly used by the layouts, which provide a template for a design thinking technique. The *sortable attribute* is less of a concrete characteristic which can be used but a category for an attribute which must be defined to allow certain functionality. This is because if the elements in a canvas should be sortable within a layout, the used elements need a sortable attribute to allow sorting.

Until now, the characteristics have primarily been based on using elements and supporting the design thinking technique. However, as the domain concepts represent objects from the domain, they must be described further, capturing additional knowledge. For this, the *description attribute* was added. It is an abstract attribute from which the concrete ones for the domain concept must be derived. To structure these attributes, a *attribute category* can be used to group attributes to gather. Further, attributes can be set to mandatory to underline the importance of an attribute. Mandatory in this context does not mean that an object of the domain concept cannot be created without the attribute, but a user should be notified that a value should be inserted here.

In addition to characteristics, a domain concept can be used within relations. Such a relation carries meaning based on its type. The type itself comes from the domain or the design thinking technique and allows to capture additional information. Creating an explicit relation between the object supports capturing information that otherwise may only be implicitly encoded, for example, through the position.

Based on the analysed design thinking techniques (see Table 5.1), three types of domain concepts were identified which can be used to represent the concepts from a workshop's domain. Single objects can be represented by *graphic-based concepts* or *text-based concepts*, differentiating the elements based on their visualisation. Graphic-based means an icon or other visual representation is used in a workshop. Examples are paper figures or figures built with Lego[®]. Text-based concepts are represented through a written term, where this term captures the meaning of this object. For example, this abstracts the usage of sticky notes with text written on them. The *container* concepts represent groups of other concepts which belong together. They can be visually represented by a box, circle or other geometric shape and elements positioned in the container are contained and belong to the group. Containers can also contain other containers. A container or

5.2. A Metamodel for the Adaptive Modelling Environment

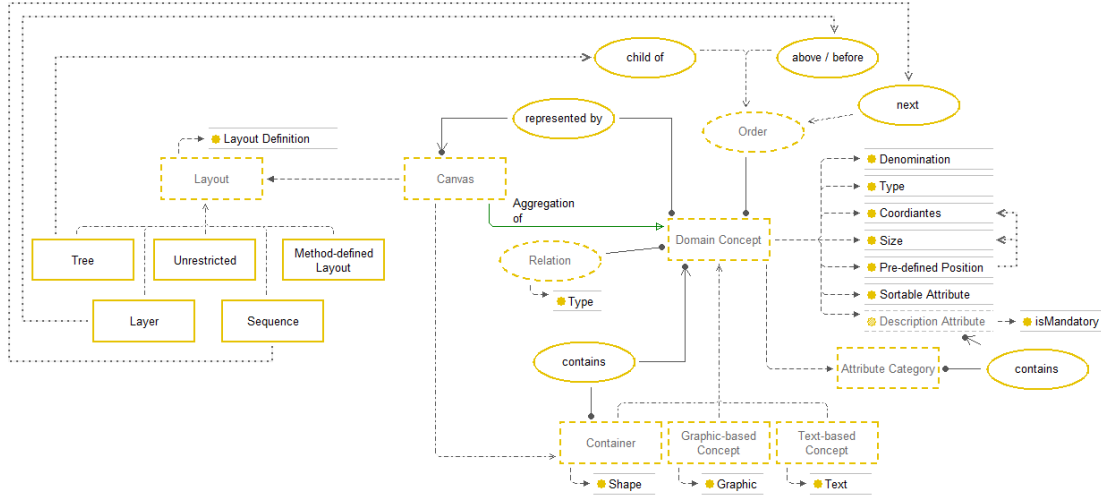


Figure 5.13.: Abstract metamodel for design thinking techniques

group can also be encoded in other ways, like using sticky notes with different colours or being implicitly assigned through the interpretation of the user. Elements can belong to multiple containers.

The *canvas* is a special container containing multiple domain concepts. It represents one self-contained artefact created as part of a design thinking technique to fulfil a specific purpose. The canvas aggregates the contained domain concepts and possesses a layout that influences the visual structure of the objects within. More information on the different layouts can be found in section 5.2.1.4. The *layout definition* describes how the objects should be visually structured in the canvas.

Domain concepts can also be *represented by* a canvas, meaning that an element of domain concept used in one canvas can be further described in another canvas. This can be a canvas from the same design thinking technique or a different one. Therefore, the object becomes an internal structure, facilitating the capture of more knowledge. For example, a person within a storyboard can be described in more detail using a persona.

A domain concept can pose special connectors in a specific layout to *order* concepts. The *child of* connector is used by the *tree* layout, the *above/before* connector by the *layer* layout, and the *next* connector by the *sequence* layout.

5.2.3. A Implementation-Oriented Metamodel for the Adaptive Modelling Environment

The section above introduced a conceptual metamodel for the *Adaptive Modelling Environment*, focusing on an integrated view of the identified fundamental structures for design thinking techniques. Based on this conceptual metamodel, this section will now discuss a more concrete metamodel oriented on implementing it and supporting the needed capabilities of the *Adaptive Modelling Environment*. Therefore, the metamodel

5. The Adaptive Modelling Environment

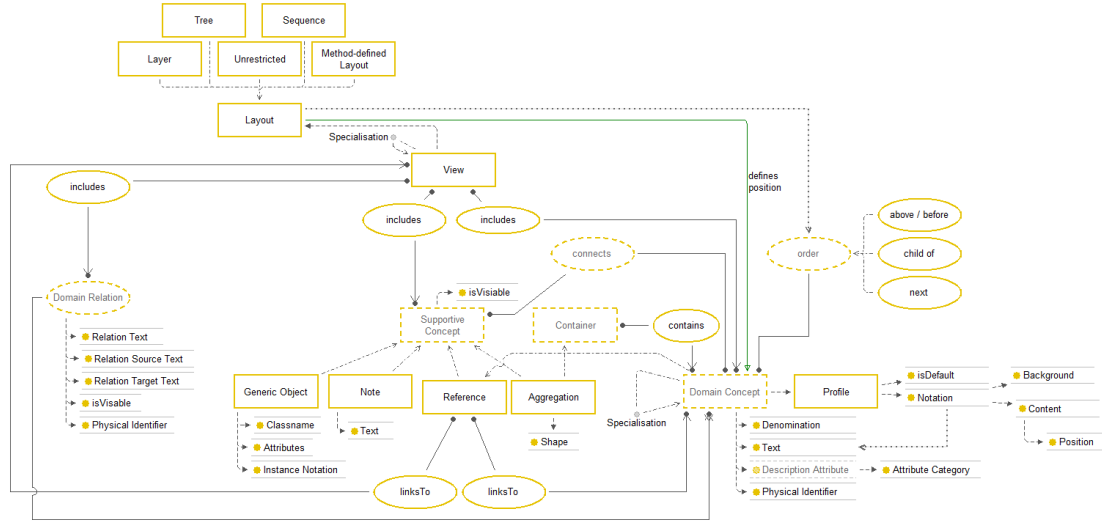


Figure 5.14.: Implementation-oriented metamodel for the *Adaptive Modelling Environment*

introduced in this section is an extension of the one discussed in the previous section.

It is extended to include aspects needed for implementation in a modelling environment. Additionally, the metamodel introduced in this section broadens the scope by including aspects needed to support the digital twin of design thinking, like adapting the metamodel and transforming tangible into digital models. This includes the importance of handling different visualisations of the objects within the tool, representing the used physical objects in a workshop.

Implementation in this context does not mean the metamodel can be translated one-to-one into code. However, it is the structure of the needed concept so that it can be used to realise the introduced functionality. For example, defining data types for attributes or providing attributes to save the position of an object were omitted so as not to overload the metamodel.

The implementation-oriented metamodel is visualised in Figure 5.14, which uses the CoChaCo modelling method (see section A.4.13 for more information).

The core of the metamodel is the *domain concept*, which is the abstract collection of all concepts represented in the models. This abstract concept holds the basic attributes needed by the modelling environment and can be utilised by method engineers to define their modelling methods. The domain concept can then derive the classes used in the modelling method.

The domain concept offers the following attributes:

- **Denomination:** The *denomination* attribute allows the assignment of a label or name to the objects created within the modelling tool.
- **Physical Identifier:** As one goal is the automatic transformation from the tangible

5.2. A Metamodel for the Adaptive Modelling Environment

to the digital model, the *physical identifier* attribute is used to save an identifier, mapping the created digital modelling object to its physical counterpart. The identifier allows for the identification of the concept in different components of the *Adaptive Modelling Environment*, like the one recognising the tangible figures.

- **Descriptive Attribute:** *Descriptive attribute* is not a concrete attribute used for instances, but it is a placeholder for the attributes which can be added by the method engineer or users of the modelling method. These attributes can be flagged as mandatory, signalling that this attribute should be filled out.
- **Text:** This attribute holds a text that can be used within the notation of the object in the digital model.

The different types or categories of the domain concepts can be encoded by creating a *specialisation* or subclass. The concrete objects are then instantiated from them and belong together. Belonging to a group can be encoded by using a container to position the object. This means a contained relation is established between the container and the object, and a group can be assigned to an object through this relation.

Classes or subclasses that can be instantiated need a notation to be visualised in the models. In the metamodel, the notation is not directly assigned to the domain concept but to a *profile*, which belongs to a domain concept. This distinction is made so that different modelling objects representing the same real-world object can have different visualisations. For example, in a storyboard, a person can be shown in different scenes, and depending on the scene, the representation can change. The facial expression can change, or the person can be shown once from the back and once from the front. Therefore, the same concept is meant, but with different visualisations. By using profiles, knowledge about what a different concept is and what a version of the same concept is can be saved.

Profiles possess the *isDefault* flag, which allows one to choose one of the assigned profiles as the default for the class. Then, the profile has the notation assigned, which represents the model. Here, the text-based and graphic-based visualisation from the metamodel for design thinking techniques (see section 5.2.2) can be defined. This is encoded by the *content*, which can either be a graphic that is designed, or the *text* attribute of the domain concept can be used as content. Additionally, a *background* can be defined to the content, which is shown for the object behind the content. The background fills out a complete visualisation of the object. The content is then positioned in the front, and the *position* can specify where the content is put. This contains the coordinates and the size. For example, a sticky note from a workshop can be visualised by a coloured background and the text used as content.

Domain concepts are also specialisations from *reference*. Therefore, they inherit the ability to create links to other views or objects which can belong to other views.

The *container* concept is abstract, meaning it can not be directly instantiated. It is used to allow other classes to inherit its behaviour. This is that the *contains* relation is automatically established between the container and all domain concepts visually put into the container's visual representation.

5. The Adaptive Modelling Environment

Domain concept is the collection of classes used to represent concepts from the system under study. To support the creators of the models, also *supportive concepts* are offered. These classes should support the user in increasing the expressibility of the models and are not bound to one modelling method. For example, they can be used by users of any modelling method to add context information and descriptive text through a *note*. The supportive concepts represented in Figure 5.14 should provide useful examples within this thesis but should not be seen as a complete list of all possible supportive concepts. As supportive concepts are an addition to a model, users should be able to hide them and make them visible again. For example, this can be used to hide them, if just the visual representation for documentation is needed.

The *note* class can add textual notes to a model aimed at other human users. *Aggregation* can be used to draw a visual frame about objects. This should not represent a group of a domain but a highlighting of a certain situation in which the model is used, for example, for sending the model to a colleague with a question about specific objects. *Reference* allows establishing links to models from other views, which may not fall in the domain of the currently used modelling method. Therefore, if a view from another modelling method is integrated, references can also establish links to them.

The *generic object* can represent any object with user-dependent information and visualisation. This allows dynamically representing an object for which no class is available. This dynamically added object should then be used to create classes, as discussed in the introduced instance based metamodeling approach, introduced in section 5.1.2.

For domain-specific conceptual modelling, it is important to represent not only objects but also their relation. Therefore, *domain relation* was introduced, which can connect instances from domain concepts. Concrete relation classes must be defined to use the relation between them, and then they can be used in the models. The visibility of relations can also be toggled so as not to visually clutter created pictures of the model. The relation also has a *physical identifier* attribute, similar to the one defined for domain concept. To allow an easy visual interpretation of the relation, they have a notation defining its notation. Additionally, the relation should be capable of showing text on it, which can either be positioned in the middle (relation text), at its beginning (relation source text) or at its end (relation target text).

The *view* concept represents the different types of available models. A view includes a set of domain concepts, domain relations, and supportive concepts, whereby the last are not permanently added to a view but are just included if the users need them. In a modelling method, multiple views can be defined and used in parallel. This allows us to focus on different viewpoints, which should be represented in each view. An instance of a view is then one model that can be created to represent an aspect of the domain.

Views can also be defined as *specialisations* of other views, meaning a new view can be derived from an existing one and inherit the set of included concepts and relations that can be adapted.

A view also has a layout which contains rules for creating model instances. The layout is used to create the model's starting point, meaning it can provide the general structure for a set of classes. This is the digital counterpart to a template provided by a design

thinking technique. Additionally, a layout can contain rules, which can be evaluated on a model to check if it fits the provided layout. The specialised layouts shown in the Figure 5.14 should be seen as examples used in this context and not all possibly needed. The provided layouts are taken from the metamodel for design thinking techniques (see section 5.2.2). Layout can also use special relations, which are used between domain concepts. These special relations are the ones derived from *order*. They are used in layout, and based on these pre-defined relations, the layout can be evaluated or established.

5.3. Modelling Tool Related Capabilities

This section will introduce the capabilities that were defined for the *Adaptive Modelling Environment* based on this thesis objectives introduced in chapter 3 and the requirements introduced in section 4.4. This section focuses on the capabilities of the modelling tool used within the modelling environment, which is adaptable to uncertain domains. The semantic driven capabilities, which focus on using the modelling environment in a workshop setting, will be discussed in section 5.4. Further, the capabilities will be based on the metamodel introduced in section 5.2.

5.3.1. Adaptation Capability

To support the adaptation of the modelling method during its usage, the modelling environment must offer *adaptation capabilities* for the metamodel, as it is the central artefact of the modelling methods definition. These capabilities should not be limited to the metamodel but to the additional artefacts needed to define the modelling method, such as the notation or mechanisms. Thereby, a user should not only be able to create the class from scratch through a GUI but that adaptation operations can be integrated into the functionality of a modelling method and, therefore, be carried out automatically. In this way, the adaptation capabilities can directly support a user by allowing a more natural interaction with these capabilities. Therefore, users or the modelling environment should create and trigger adaptation operations.

By allowing not only the adaptation of the available concepts but also the offered functionality through mechanisms, the value for the users can be increased further. This further allows us to tailor the available functionality of the modelling tool, fitting it better to the users' needs.

Suppose we see the implemented metamodel as a type of information storage of the domain. In that case, we can use the concept of *CRUD* operations to structure the adaptation mechanisms for our modelling environment. *CRUD* are the basic operations which can be applied to data and stands for:

- Create: Something new is created
- Read: The content of something is retrieved
- Update: Something existing is changed.

5. The Adaptive Modelling Environment

- Deleted: Something existing is removed.

In this context of the adaptation, the operations *create*, *update*, and *delete* are important, as they change the metamodel or additional artefacts. In this context, the *read* operation has a supportive character, as it does not change the metamodel but is needed to guide and influence the adaptation mechanisms.

These operations must then be applied to different items within the metamodel to be beneficial. The items used in this section are consolidated from the fundamentals (see section 2.3.3 and section 2.3), the thesis idea (see chapter 3) and the requirements (see chapter 4). The following list provides an overview of the items which are used in the context of this section:

- **Class:** The representation of a group of objects which belong together. A class represents a domain concept that models can use to describe the instances. The notation of the objects in the model is defined based on the class.
- **Attribute:** Characteristic of an object in the metamodel or model. Attributes can be assigned to classes, relations, or model views and can be of different types.
- **Relation:** A relation is a visual connection of two instances within one model, represented through a line, which is the notation of the relation. It represents the connection the two concepts have in the domain. A relation of concepts between models falls not in this item, but such a connection will be encoded as an attribute.
- **Model View:** Model views allow the group of the defined classes within a modelling method together, representing views on the system under study. The different model views together then represent the knowledge of the domain. Additionally, model views can be put in a hierarchical order to create subviews that define a subset of the classes of a model view for the user.
- **Notation:** Notation defines the visual representation of the instances in the modelling tool. The notation is assigned to the classes and applied when instantiated. It is assigned to a class but will be discussed here separately as it is important in communicating meaning to humans.

5.3.1.1. Add Concepts to the Metamodel

One of the fundamental adaptations which must be supported in the *Adaptive Modelling Environment* is the creation of classes representing the concepts of the domain. In addition to the instances, the classes are an important aspect of the domain abstraction, as they classify the instances and provide context. Therefore, a modelling environment in the context of this thesis must be able to create new classes for the modelling method, e.g., as used in the instance based metamodeling approach introduced in section 5.1.2.

However, to make the class usable in the modelling tool, it must not be just created and saved in the metamodel of the modelling environment. However, it must be configured so that it can be used appropriately. First, the fundamental behaviour must be established,

which can be done by inheriting it from an existing class. The user can provide the superclass directly or suggested from the context. As a dynamic creation of a new class will usually happen in the context of creating a model, this model can be seen as the context, and the modelling tool can use the classes and their superclasses, which are used in the model, as suggestions for the user. Only all or no (depending on the user) available classes can be shown without such a context. If the user does not choose or provide a superclass, then the most generic class should be used as the superclass.

The most generic class is at the top of the class hierarchy and provides the basic structure of a class. It is not meant to be used in a modelling method but is a class used internally in the modelling environment. This most generic class is the fallback for the created classes and ensures that all classes in one modelling method are used in the same class hierarchy. However, more information will be available if a more specialised class is used. For example, it can inherit the attributes of the superclass and can be used in the same relations as the superclass. The most generic class which can be used to create domain classes is represented by *Domain Concept*, or rather its direct subclasses for graphical or text-based objects, as introduced metamodel in section 5.2.

Another essential aspect that must be considered is assigning a newly created class to a model view. If the new class should be available to users, it must also be assigned to a model view. The user must also provide this information. Here, the modelling tool can suggest the model view of the currently opened model, as it is probably the model view from which the class should be used.

As humans will use the classes, they should have a unique human understandable identifier, usually called a class name. Allowing the creation of new classes includes the need to handle if two classes have the same name. This can occur if the same class should be used in multiple model views. Here, the modelling tool should resolve by not creating a new class in the modelling tool but by adding and tailoring the existing ones to the model view. The available attributes and the notation of a class need not be the same in all used model views.

The same thoughts must be made if a new relation class is created. A super relation class must be added to the relation class so that the wanted behaviour can be inherited, the relation must be added to the modelling view, and the modelling tool must check if the relation already exists. However, in addition to this information, during the creation of a relationship, two classes must be provided: one from which a relationship can start and one from where it can end. Here, users can enter classes they already know or all available classes in the model view, which is shown to them, and they can choose the classes. Here, the abstract classes must also be shown so that the endpoints of a relation are not restricted to only one class.

When creating a new attribute, it must be assigned to an already existing class, relation or model view. Therefore, during the creation, a user should be able to choose from the existing classes, relations and model view to define where it should be added. Then, a name and a type must be given, as this information lies at the core of each attribute. Afterwards, some optional or additional information must be provided. Optional information could be a default value automatically assigned or how big an integer value can be. Depending on

5. *The Adaptive Modelling Environment*

the attribute type, additional information may also be needed. For example, the possible values must be defined if an enumeration attribute is created.

Creating a model view must be separated between sets of classes and relations that belong together. This can be used to differentiate between different workshops or domains. A name must be provided to create a model view that identifies it. During the creation procedure, it should also be decided if a sub-view should be created. The super view must be provided, from which the list of classes and relations and the available view attributes are inherited and reused after the model view is created. It can instantiate models and fill them with the provided classes and relations.

Notation is a main factor in communicating meaning to humans, so it must be tailored to the domain. Therefore, a corresponding notation must be established if a new class or relation is created. How the notation for an element (class or relation) can be defined differs for the concrete case. However, after a new class or relation is created, the creation of the notation should be triggered, as otherwise, it cannot be used in the modelling environment.

As the modelling environment must dynamically render relations, no static notation in a picture is applicable. However, a static notation based on the modelling environment's capabilities is possible by allowing the user to choose the notation from available constructs. These usually include a line type like solid or dashed, a colour, a thickness, and the arrow or ending shown on which end. Suppose dynamic notations, e.g., based on an attribute value, should be defined. In that case, they must either be imported from an existing definition or created manually. It needs a more sophisticated knowledge of the modelling environment and is not part of the dynamic adaptability through novices.

A static notation in the form of a picture file can be provided for classes. During the creation, the user must decide to which aspect of the class the new notation should be assigned based on the metamodel introduced in section 5.2. This is the specific profile of the class, which can be applied to the instances of the class. Classes should also be able to offer dynamic visualisation, but for this, a repository with already existing ones must be available, or they must be created manually. For the latter option, the user needs more sophisticated knowledge, and more time is needed, which hinders its easy usage during workshops.

However, the notation of classes and relations both need the ability to be visualised with text. This needs an attribute in which the user can enter the text and the position and size of the text field relative to the picture's origin. This information must be queried from the user, who should be supported by the definition of the position, for example, by visually indicating the position in the text field and providing information to change the position in real time.

In some cases, a special visualisation of an object in a specific model can be beneficial. This visualisation should not be reused in other situations, but only in this. Therefore, a modelling environment should also provide a specific notation on the level of modelling objects. This is not changing the underlying class but overwriting an individual visualisation that can be used for a specific model.

5.3.1.2. Change Concepts in the Metamodel

It is not enough to create new elements; sometimes existing ones must be changed, which falls under *update* in the *CRUD* acronym. What exactly can be changed depends on the type of the item. All updates should ensure that the modelling environments and the available models stay usable after the change is applied.

Classes can be updated in two ways. One is to change its place in the class hierarchy, and the second is to change its class attributes. Changing the class hierarchy means changing the class from which they inherit their behaviour. This is because a new superclass is assigned for a class, and the class will be unified with another.

For the reassignment of a class, the attributes of the new superclass must be added to the class, and the attributes inherited from the old superclass should be kept and added to the class directly so that if instances use attributes of the old superclass, they do not lose them. Before the class can be changed, it must be checked if none of its instances or subclasses is used in a relation that is only allowed because of one of its superclasses. This means that the endpoint of the relation is set to allow instances of the superclass. Therefore, if the superclass is removed from the class, its instances are no longer allowed in the relation.

If users recognise that two classes are too similar, they can use the creation and the update operation to unify them. This can be done by creating a new class and then updating the superclass of both classes. Alternatively, one class can be integrated into another by transferring all the attributes or defining it as a subclass.

Allowing the hierarchy to change should also allow instances to be assigned to other classes. For example, if a class is split by introducing a new subclass, fitting instances of the old class should be assigned to the new class. If the new class of the instance does not possess all the attributes now used, the user should decide if the attributes are added to the newly assigned class or if the attributes should be discarded.

The last way to update a class is by changing its class attributes. These are attributes which have the same value for each instance. This is, for example, the classname, a flag if the class is abstract or the available choices for an enumeration attribute. If a class attribute is changed, this change must be applied to all instances which use the class. Before the update operation is executed, the modelling environment should check if this leads to inconsistencies. For example, if the classname is already used, if an instance for a class exists, which should be made abstract or if the choice of the enumeration attribute is used in an instance. Such checks can only be made if the attribute's semantics and usage are known. Without this, only a warning to the user can be issued, stating that problems could occur. Additionally, if the classname is changed, the updated class must be included in the model view.

The update operation applied to a relation can change its name or endpoints. If the name is changed, it must be checked if it is already used. If the endpoints are changed, it must be checked before if an instance of this relation exists, which is not allowed afterwards. The endpoint of a relation can be changed if no instances of this relation exist or if the new class is one of the super or sub-classes of the current class. If the endpoint is set to a superclass, it is no problem. Additionally, if the relation name changes, the

5. *The Adaptive Modelling Environment*

updated relation must be included in the model view.

Independently, if an attribute belongs to a class, a relation, or a model view, it can be updated by changing its name or type. If the name changes, it must check if it is already used in this class or one of its super- or subclasses. Additionally, if the attribute type is updated, it must be checked in advance if an instance has a value assigned to it. If so, it must be checked if a meaningful casting of the variable is possible. For example, the value of an integer can also be saved in a float variable.

Problems can occur in existing models and implemented mechanisms, which must be checked if the attribute is used. If the attribute name is changed, then it must also be replaced in the definition of the mechanism. If the type is changed and the variable is used in a mechanism, the user should be informed that it can come to problems through the change.

The update operation applied to a model view can change its name or the included classes and relations. If the name is changed, it must be checked if the name is already used and if the model view is used as the goal of a reference. If so, the name in the reference definition must also be changed. A class or relation can be removed from the model view. If a new item should be included, it must check if it is not already included or available.

The update of the notation of a class or relation can be seen as a new one is created, which overrides the existing one. Therefore, if the notation should be changed, the same considerations as for the creation must be taken.

5.3.1.3. Remove Concepts from the Metamodel

Removing a class in a modelling tool can be interpreted in two ways. It can either be removed from the currently used model view or deleted within the modelling tool. The delete operation should, therefore, first remove the class from a model view, and if the class is not used in any other view, it should be deleted from the modelling tool.

To remove the class from a view, the user must provide the class and from which model view. Here, it is reasonable to show a user a list of the classes available in the model view of the currently opened model and suggest the currently open model view as the one from which the class should be removed. However, before the class is deleted from the view, it must be checked if an instance of the class is used in one instance of the model type. If so, the class should not be deleted, and the user should be informed of which models have instances of the class. In addition, it should be checked if the class is used in a mechanism specified for the model type, and the user must be informed accordingly.

If a class is deleted from the last model view, there cannot be any instances of the class left. Here, it should be checked if the class is used in an available mechanism; if not, the class should be deleted from the storage of the modelling environment. If it is used in a mechanism, the user should be informed.

The same approach and thoughts can be applied to delete a relation. The first step is to remove it from the model view, and if it is not used anywhere else, it should be removed from the modelling tool, assuming all the checks described for the class are also applied to the relation.

A similar approach can be applied to deleting attributes. However, as described in the basic metamodel (discussed in section 5.2), not all attributes are accessible in every model view. Therefore, if an attribute should be removed from a class, it should first be removed from the class in the concrete view if it is not used in any instance. Then, if it is removed from the last view and is not used in any mechanisms, it should be removed from the modelling environment storage.

A model view can only be deleted if no instance of it is currently available. If the removal of a model view is triggered, the view of the currently open model can be suggested to the user. If the model instances of the model view are available, the user should be informed about the instances, and the delete procedure should be aborted. If the model view is deleted, it should be checked if the included classes and relations are used in another view. The delete procedure should be triggered for those classes and relations not used anywhere else, including asking the user for permission before deleting.

Additionally, if a view with sub-views should be deleted, it should be first checked if those are used and can be deleted before the current view is removed. If all sub-views are removed, the super view should be deletable.

If the notation of a class is deleted, it should be handled like an attribute. First, it is checked if the notation profile is used somewhere in this view. If the profile is removed from the last view, it should be removed from the modelling tool. However, if the profile is removed, it must also be checked if the class still has some notation left. If this is not the case, then the notation should not be removable, as a class without notation should not exist. Therefore, the last class notation can only be removed by deleting the class.

5.3.2. Providing Comprehensible Notation

The visual representation of information regarding an existing or planned system under study supports human understanding [96], as humans can interpret the visual encoding of information as provided by the position of the individual parts. This is exploited by diagrammatic modelling methods, which allow the creation of visualisations by positioning individual symbols on a canvas, which are interpreted together. The users' understanding can be further enhanced if the used symbol carries semantic form the concepts it represents in the system under study [40].

Also, creative methods like those used in design thinking (see section 2.1) often use the visual structuring of information by putting physical objects on the canvas. Their position and representation encode knowledge needed within the creative method. The physical objects thereby represent concepts from the system under study.

The capability of *providing comprehensible notation* discusses how the envisioned modelling environment can support the understanding of the users by providing a comprehensible notation for the users. In most modelling methods, meaning is assigned to its elements based on its metamodel and the definition (see section 2.3 and section 2.3.3 for more information). Therefore, the semantics are created in advance and must be learned by the user to use the method correctly.

This way of providing meaning to the elements used in the model does not apply to the context of the *Adaptive Modelling Environment*, as the domain and the used elements

5. The Adaptive Modelling Environment

are not fixed before its usage. Therefore, their meaning can not be sufficiently defined in advance. In order to still offer semantics for the user, semantically rich icons are used which can be understood by human users. These icons are not abstract shapes but pictures representing the concepts. The icons should then be used in the physical and the digital environment.

The modelling environment must, therefore, be capable of using complex figures as notation for the elements. The easiest way to achieve this is by allowing existing pictures to be used as icons for the elements.

The icons must be provided by the users, who are responsible for the semantics encoded in the icons. To utilise this goal, the users must be able to change the icons to their domain and their interpretation during the runtime of a workshop. This is possible if the modelling environment supports the usage of predefined pictures and implements the *adaptation* capability introduced in section 5.3.1.

Additionally, the concepts can be represented using a label shown in the model. In the physical model, this could be a sticky note with text written onto it; in the digital model, this can be a box with a label. This is also a feasible way to encode knowledge for humans, which has the benefit that it can be easily created, as no one must search for a fitting icon. However, an icon-based visualisation is easier for humans to interpret than text, especially if multiple objects are used.

Last but not least, the modelling environment must be able to create diagrammatic models, meaning that elements can be positioned freely on a canvas as individual objects which can later be moved. For example, it includes drawing relations between them as lines or that different figures can overlap. This functionality will not be discussed in detail in this thesis as it is currently a default functionality for many modelling and diagramming tools.

5.3.3. Extending the Modelling Method

Related to the adaptation capability (cf. section 5.3.1), the *extending the modelling method* will change the metamodel of the modelling method. However, during the extension, the goal is not set on changing the core of the modelling method but on extending it with certain elements that are useful in specific situations. Such extensions are not meant to become part of the modelling method as a whole but to support the users of a modelling method in a concrete situation.

Such elements include adding a supportive class, importing a modelling language, or integrating a new functionality. Such extensions can be created manually by the user or imported and used if finished elements are available to the user. Such finished elements can be provided in a repository or a user collection. We will focus on integrating existing elements for extension in the context of the *Adaptive Modelling Environment*. As this approach fits better with extending the modelling method during its usage, creating complex elements from scratch would hinder workflow.

An extension within the modelling environment will be executed based on the defined adaptation operation (cf. section 5.3.1), but the goal is different. Adaptation focuses on the change of the modelling method itself, and extension adds useful elements to it.

During the extension, the adaptation operation will not be executed individually but as a collection to extend the modelling method to the new element. This supports the flexibility and alignment of the modelling method with the users' needs.

Functionality can be added by allowing it to be saved in a way that can be loaded into and executed by a modelling environment. For example, the functionality can be saved as a script, which the modelling environment can interpret. The functionality can be generic (cf. [2]) so that it can be used on every model based on a specific structure. An implemented modelling environment means all the models and metamodels are created with the modelling environment, based on its meta²-model or a provided abstract metamodel. Therefore, generic functionality can also be based on the meta²-model or the abstract metamodel and used with every model without adaptations. In this case, only the functionality must be imported, and it can be used with all available models.

The metamodel must contain a certain structure if the functionality is more specific (cf. [2]). This structure can be added to the metamodel of the modelling method and then used in the functionality. In this way, the metamodel must also be extended. However, this is only possible if the functionality is created to allow the packaging of the needed parts of the metamodel to be done with them. An example of functionality coupled with the needed metamodel part is discussed in section 5.4.2.

Suppose the functionality should be executed upon the core of the modelling method. In that case, a mapping must be created between the aspects needed by the functionality and the aspects provided by the core metamodel. Here, additional user effort is needed, which can again hinder their overall workflow.

The extensions can be integrated into the modelling method by using them directly in the models of the modelling method, which should be extended. If the modelling method is extended with complete modelling languages to create separate models, then the *linking multiple models* capability (cf. section 5.3.7) can be used to create links to the extended models in the core models.

5.3.4. Configure Mechanisms

The conceptual architecture (see section 5.5) introduces that the modelling environment must not necessarily consist of only one application but can have different components which must work together. Here, the modelling environment must provide functionality to integrate the different components. Additionally, the modelling environment should be able to extend itself by dynamically loading mechanisms as discussed in the *extend concepts* setting scenario (see section 3.4.2).

To use them, the modelling tool must know where to reach the additional components in the modelling environment or how the loaded mechanisms must be used. Here, a configuration must be supported to allow the modelling tool to adapt to the user's context, for example, if the modelling tool should be used in another workshop location with different components for recognising the tangible figures.

The user must provide the values for configuration parameters so the modelling tool knows how to use the component. The parameters that are needed are defined directly by

5. The Adaptive Modelling Environment

the mechanism. Therefore, the modelling tool must be able to load the needed parameters and offer them to users so they can be filled in. To ease the usage, the modelling tool should save the configuration internally until it is changed.

This can be done by providing configuration files used by the modelling tool's mechanism to save and adapt them. Then, if need be, the user can open the file through the modelling tool, change the values, and save it for use.

5.3.5. Ensure Consistency

The modelling environment should be able to understand defined consistency constraints and support the user in respecting them. In the context of an adaptive modelling environment, adaptations must not lead to unnoticed states of the modelling environment where specific models or mechanisms cannot be used any more. How consistency can be integrated into the adaptation mechanisms is discussed in the introduction of the adaptation mechanisms in section 5.3.1. Here, the general approach to dealing with inconsistency was that the adaptations are not allowed to cause inconsistency in the existing models. For mechanisms, the user is warned if, after an adaptation, the mechanisms may not work correctly.

This consistency in adaptation must also be applied if the models with the metamodels are shared with other people. Suppose a user imports a model where the metamodel was adapted. In that case, the modelling environment must check if the adaptation can be applied or if existing models are rendered inconsistent. The check should show the problematic changes to the user and provide solution strategies, like the important part of the metamodel with a new name or a new version number.

The modelling environment should also be able to evaluate consistency constraints added by the user. These can be types of attribute values or instances between which a relation should be established are instances of the correct classes. The standard consistency checks available in the modelling tool should also work with the adaptation operations.

The modelling environment should also support the evaluation of other user-defined consistency constraints if the user is enabled to do so. However, the discussion of how domain-specific consistency constraints can be defined in metamodels would go beyond the scope of this thesis.

5.3.6. Instance Modelling

To increase the flexibility of the modelling environment, as required by the *flexibility* requirement (see section 4.4.9), two capabilities are used. One is the adaptability, which is discussed in section 5.3.1. The other one, discussed here, is based on the *Generic Object* of the metamodel introduced in section 5.2.3.

This class is a supportive concept, which means it is agonistic to a specific modelling method. It can be used to create a visual element in the model, which has no predefined structure and can be applied by the user to represent an undefined concept from the system under study. Here, attributes can be added, and a graphical representation can be made in text or an imported icon. A generic object can be used if something should be

represented, but the users are unsure which class to put it in or its exact characteristics. Therefore, the users can flexibly represent new concepts.

After information is added, the generic object should automatically create a class and be added to the model modelling method. Flexibility is essential initially when the concept is not yet fully known. However, when the concept is explored, it should be integrated into the metamodel to increase the captured domain knowledge.

However, the generic object can not only be used manually by the user but can also enhance the capabilities of mechanisms. For example, suppose the transformation from the physical to the digital models recognises a tangible object which cannot be matched to a class in the modelling tool. In that case, the information of the tangible object should not be discarded but instantiated as a generic object. In this way, the information from the workshop is captured and can be further utilised by the users.

5.3.7. Linking Multiple Models

If complex systems, like the ones designed in workshops, should be modelled, such models can become complex and challenging to understand, which cancels out why we want to use models. Model views can help by logically separating the complete model into different views and providing smaller, more easily understood models. How the views are defined depends on the concrete context. They can represent different points of view, like technical and organisational, or the views can contain different abstraction levels. To allow more sophisticated exploitation of the captured knowledge, the different views must also be linked with each other [109, 110].

To achieve this, the modelling environment must be able to create and work with different views from which models can be created. To link models and concepts between different views, a concept must be introduced which allows the storage of a reference to models as a whole or objects in other models. In the introduced metamodel of section 5.2, the different views are represented by the *View* concepts, and the linkage between different models is represented by the *Reference* concepts. The reference connects an object with a whole model or a singular object. This is similar to a relation, only that the reference is not visually shown in the model but only saved to be exploited over functionality, and a relation to a whole model cannot be created.

The reference is defined as an attribute in the class and can then be used to set links to other existing models and objects. Here, it must be possible for a user to follow the reference and jump to the referenced element. This can be done by adding a link to the notation of the object, or the user can click on an attribute to follow the link. The reference can be created, changed and removed by the operation introduced in the adaptability capability (see section 5.3.1). Linking different models allows the integration of the knowledge created in different models. Therefore, it can be better exploited than saving this knowledge in parallel, as humans must make the connections manually in their heads.

The model views can also contain different modelling methods if available in the modelling tool. Then, the models from the workshop can be linked to models from other modelling methods (e.g., business processes), which can be used to concretise the created

5. *The Adaptive Modelling Environment*

knowledge. Such additional models need not be created during the workshop but can also be established afterwards.

A reference can also be made to the model from the same view. Doing so allows the creation of submodels that can be used to structure the created knowledge further. This also supports scalability as the knowledge of complex systems can be modelled comprehensibly by separating the knowledge into understandable junks.

5.3.8. Processable Representation

To increase the value of the models, they should be processable by computers. This means that the modelling environment, primarily the modelling tool, must be able to process the information captured in the models. However, the other components in the modelling environment must be able to process the information. Especially the information from the physical models must be captured and processed to be transformed into the digital conceptual model, which can then be processed to support the workshop.

The information must be represented internally to allow the computer to access and process it. Therefore, the digital model must be more than a mere picture but a structure accessible to the machine. As this is available by default if a metamodeling platform is used, this will not be discussed further in this thesis.

The processing improves the value for users by reducing the cumbersome tasks they must perform and letting them set their focus on the important and value-adding tasks. For example, the transformation from physical to digital models eases the need for users to reconstruct the structure already created physically but allows them to focus on enriching the models so that all the necessary knowledge is captured.

To improve the processing capabilities, users must be able to add mechanisms which support the processing they need. This can be done manually by implementing the mechanism and adding it to the modelling environment. Even though this can be done and is useful, it does not align with the goal of the thesis. Therefore, the modelling environment must be capable of importing predefined mechanisms and offering them to users. Here is the assumption that an expert has implemented this mechanism and provided it for potential users to import. This can be achieved if the mechanism is implemented in a script, which can be interpreted and executed by the modelling environment. Another way would be offering a callable interface with which the modelling environment can communicate. In this case, the modelling environment must offer a mechanism for how this interface can be called. Independent of how it is implemented, the modelling environment must provide a trigger for users to interact with to execute the mechanism.

The processing of the modelled information makes sense within the modelling environment and should be shareable with external applications. Therefore, the modelling tool should be able to export the models using computer-processable syntax. In this way, other applications that process the modelled information can be created but are not directly integrated into the modelling environment.

The models resulting from a creative workshop will usually be highly abstract. This benefits the communication between stakeholders with different expertise within the

workshop. For them to be implemented, they must be enriched with additional information to make them more concrete. This specialisation can be done with additional domain-specific modelling language. In this context, processable representation means the workshop models can be transformed into starting points for other modelling methods and their tools. The model form workshop must be transformed, so its structure must be processable and encoded in a form that the other modelling tool can understand and import. Here, additional knowledge must be provided on how the information from the workshop model can be mapped to the structure of the other modelling method. This knowledge can be represented in transformation rules, which a transformation service can interpret. The rules must then be specified for each modelling method.

5.3.9. **Logging Changes**

The models created within the workshop and the used metamodel are prone to changes based on the uncertain domain in which they are used. The modelling environment should log the important changes to improve the understanding of how an idea develops over time. This can be helpful during the further design and implementation of the idea after the initial workshop.

Regular logging on model and metamodel levels during a workshop can help involved stakeholders understand the development of the idea. Logging can be achieved by regularly triggering the automated transformation from the physical to the digital models. This development of the models then captures the development of the idea. The logging should be parallel to the transformation triggered by the user, as these represent the user-defined states; they must be provided to represent the users' ideas.

Additionally, the execution of the adaptation operation on the metamodel can be logged to get a timeline of how the metamodel has changed, which provides a central abstraction. The user should be able to describe why this change was applied for essential changes.

5.3.10. **Model Sharing**

Users of the adaptive modelling environment must also be able to share the created models with other stakeholders. This is a standard functionality for many modelling tools, allowing the export of the model in a file. The structure of this file is dependent on the modelling tool and is not the focus of this capability. Instead, this section will discuss the distinct features of sharing models in the adaptive modelling environment.

As the metamodel can be adapted during usage, these adaptations must also be provided if the models are shared. Therefore, the metamodel with the additional artefacts like notation and mechanisms must also be saved with the model information file and interpreted by the importing model tool.

The importing modelling tool must then use the adaptation capabilities (see section 5.3.1) to prepare its metamodel to be able to show the imported model. This can be because the model view is not defined in the modelling environment and must be created from scratch. This can be done automatically if the metamodel and the additional artefacts can be interpreted and mapped to the needed adaptation operations.

5. The Adaptive Modelling Environment

If the model view already exists, then the modelling environment must be able to check if a difference exists between the available metamodel and the one that should be imported. If a difference exists, then the modelling environment must be able to identify these differences.

Such conflicts must then be handled, and the user must make decisions. First, the user must decide if a new version of the model view should be created for the model view or if it should be integrated into the existing one. If a new version is created, the imported models do not fit existing models. Additionally, too many model views in one environment can impede its usage.

If the imported metamodel should be integrated into an existing one, adaptation operations must be applied to the available metamodel. Here the consistency of the metamodel with the existing models must be ensured (cf. section 5.3.5). The consistency will be less of a problem if concepts, relations or attributes are added, but it can be problematic if these should be deleted or changed.

Additionally, the updates of concepts, relations, and attributes are hard to identify, except when a changelog is provided with the metamodel, inflating the information that must be exchanged between the modelling environments. However, without recognising the changes in the metamodel, different concepts, relations, and attributes for capturing the same information can be developed after merging the two metamodels. This would violate the requirement *ontological complete and clear* (see section 4.4.17).

Multiple concepts, relations and attributes will be better than creating inconsistencies or missing information. However, such decisions should not be automated. Instead, they should involve the user. Therefore, the modelling environment must report the conflicts to users comprehensibly and support them by resolving them.

5.4. Design Thinking Digital Twin Capabilities for Using the Adaptive Modelling Environment in Workshops

In the above section, capabilities focusing on the modelling tool were introduced. This section now introduced important capabilities of the modelling environment, focusing on those who improve the usage by utilising or creating semantics in the digital twin.

5.4.1. Transforming Tangible Models into Digital Twins

A critical capability of the introduced environment is automatically transforming tangible into digital models. This is essential to support the usage in workshops, where the physical objects foster cooperation between the participants. The automatic transformation is then used to capture the visual structure created with the physical objects and provide them as a digital, conceptual model that is easy to share and adapt after the workshop. This digital model is the core of the design thinking digital twin.

This transformation allows the user to create models more naturally in a workshop, as some tangible techniques are already used here. Therefore, the usability of the modelling

environment is increased as the creation of models is more native to how they create knowledge in workshops.

Here, we focus on ways to recognise the tangible objects that are used, which are based on computer vision, and therefore process digital pictures of the tangible objects to extract the needed information. This method was chosen because it fits naturally to the context, where tangible representations should be interpreted visually. Further, it does not impose high hardware requirements on the users and the hardware to take digital pictures is easy to transport.

The first step in identifying the used tangible objects and their position is digitising the tangible objects in the form of a picture. It does not matter how the picture is taken as long as it is provided to the object identification component.

The tangible objects and their position can be identified based on the digital picture. The following two possibilities will be introduced, with which the tangible objects can be recognised. The first one will be based on machine learning, where the visual representation of tangible objects will be taught to the computer so that it can find them in the pictures. The second will be based on predefined tags, which will be attached to the tangible objects and can be easily identified.

5.4.1.1. Identifying Tangible Objects with Machine Learning

Object detection is the computer vision task for identifying which objects are available in a picture and where they are positioned (see section 2.4.2 for background). State of the art for applying object detection is to use convolutional neural network (CNN) [130], which we also used in this approach for identifying tangible objects.

To use CNN, one has to collect a massive amount of pictures of the objects which should be recognised and train them into one CNN. The finished neural network can then be applied for object detection. If new objects with new labels should then be added to the CNN, the training must be started from the beginning. Adding a new object with a new label to an existing neural network is difficult if the neural network architecture is not changed, especially as the number of pictures per object should be approximately the same for each object.

For the adaptive modelling environment, the tangible objects should be tailored to the scenario and domain of a concrete workshop. Therefore, the figures which should be recognised should also be tailored to the workshop, as well as the objects available in the modelling tool and the recognition. As the same concepts must be used in these three components, they must be synchronised.

CNN are often trained to recognise groups of objects which belong together, like *person*. Each used physical object in a workshop gets assigned a specific semantic in the workshop context, which should be transferred to the digital model. Therefore, the aim is to train the network not with classes like *person* but to allow the recognition of concrete objects, like an accountant or electrician.

Therefore, this section introduces the algorithm created to support a semantic-rich automatic transformation from physical objects to digital models. The algorithm utilises CNN based object detection and maps this capability to the case of adaptive modelling

5. The Adaptive Modelling Environment

in workshops. The mapping between the recognised physical object and their digital counterpart will utilise a graph-based knowledge base, like RDF ontologies (see section 2.4.1 for more information.)

The general idea for object detection is that multiple trained CNN can be used together, allowing for a more straightforward adaptation of the physical objects that can be recognised. The objects identified with each neural network are consolidated so the modelling tool can provide a final set of figures. This way, the recognition of different objects can be dynamically changed without retraining one net for every possible figure.

The graph and logic-based knowledge base was chosen as it provides two benefits in the context of the adaptive modelling approach. This knowledge base is very flexible, as the A-Box and T-Box can be adapted at runtime, which allows for an easy adaption and supports its tailoring to the workshop. Additionally, the ontology part allows reasoning in the knowledge base, automatically enhancing the provided knowledge by adding a new one based on predefined rules. This is important to reduce the information a user must provide.

However, the required information in the knowledge base and trained network will be minimal for the algorithm. Therefore, even with only a little information, the algorithm will work but with poorer semantic information. Therefore, the more information is provided, the more detailed the results will be.

5.4.1.1.1. Different Abstraction Layers of Recognised Objects: The identification of the individual used physical objects may not always be possible. For example, the figure was dynamically added during the workshop and is not yet trained. Alternatively, the figure's training did not recognise it in every lighting and background condition. We want at least some information for the modelling tool in such cases. To achieve this, we introduce recognition of multiple abstraction layers. An example is visualised in Figure 5.15.

The example shows three abstraction layers to which recognised physical objects are mapped. The three layers in the example are concrete objects, classes, and generic objects. The concrete object layer contains these objects that can be directly linked to physical objects. The class layers contain a group of physical objects which belong together in the domain. In the example of the figure, this is that *accountant* and *engineer* belong together in the class of *person*. Last but not least, the *generic object* layer is the superclass of all physical objects used.

This behaviour is achieved by reusing the training data from the lower abstraction layers and using it to train the higher ones. In the example, the training data for *accountant* and *engineer* are joined, and the label for training is changed to *person*. Then, a new CNN is trained, which can be used. Therefore, each abstraction layer in Figure 5.15 has its own trained nets. The nets on a higher abstraction layer possess a more extensive and robust training set and identify objects better, even those with which they are not trained.

Within the algorithm for consolidating the results from the different trained nets, the received labels must be mapped to the abstraction layer, as this information is essential

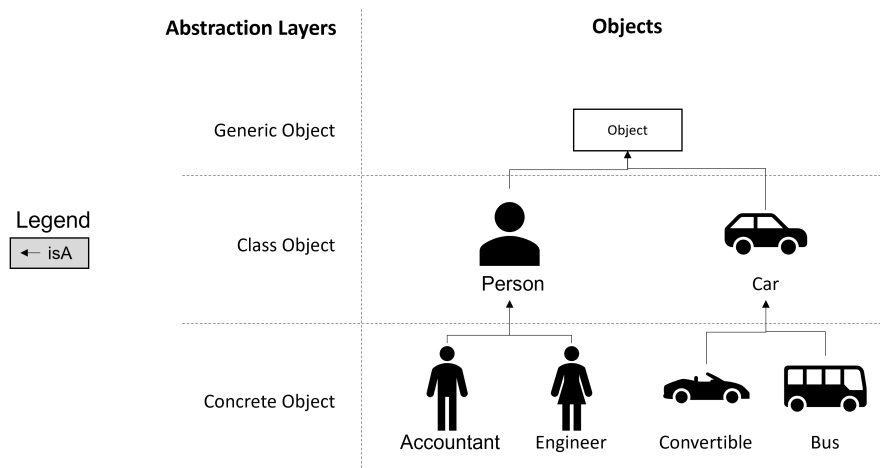


Figure 5.15.: Visualisation of an example of 3 abstraction layers for recognising physical objects

for choosing the correct label. Here, the labels from a trained net from a lower abstraction layer are prioritised to those on a higher abstraction layer.

Text written on the objects should also be recognised to increase the information that can be automatically transformed into digital models. Therefore, a trained model for recognising handwritten text is needed in addition to the CNN for object detection. However, this model must not be tailored to each workshop, but a general existing one can be used. The recognised text must then be mapped to the identified objects to be added to the digital model.

Assigning the found text to the object is done by using only the part of the picture where an object is found as input for the text recognition. This picture of the identified object will be cut out of the picture from all the physical objects. This cutting out is based on the results of the CNN based object detection.

5.4.1.1.2. Enhancing the Perceived Knowledge: Conceptual modelling not only focuses on finding symbols to represent a concept from the system under study. The relations between and properties of the concepts should be captured in addition to the symbols. Humans can visually interpret the digital model and derive additional meaning. In such a way, this knowledge is only available implicitly. However, this implicit knowledge should also be integrated into the models to increase the model's usefulness further. The automatic transformation can here use semantic-rich technologies to support the users.

Information cannot be created out of thin air. Therefore, some basic knowledge must be provided to the modelling environment, which can be used to apply to the created model. Therefore, the here-introduced algorithm will use the trained CNNs and the knowledge saved with the trained concepts in the specially created database. This additional knowledge should not be seen as an obligation but as an additional feature. The transformation will still work without it, but the results will be limited in comparison.

5. The Adaptive Modelling Environment

The concepts' properties will be loaded based on the identified physical objects and how they are mapped to the additional knowledge. The information about the properties can be on the class level, meaning only the empty properties are offered in the model. Therefore, a frame with which the concepts can be described is proposed. The workshop participants must fill in the concrete information. If additional knowledge is provided on the instance level, the properties and the concrete values can be automatically added to the conceptual models.

Two approaches will be discussed for this algorithm to create relations in the digital model automatically. The first assigns specific physical objects the role of being interpreted as possible relations if they are positioned in the middle between two other physical objects that are not relations. Based on this positioning, the recognition component will interpret this physical object as a relation. The visualisation of this concept can be seen on the right side of Figure 5.16.

The physical objects are enhanced with text to assign semantics to the relations. The text is then recognised, assigned to the identified objects, and added to the object using the modelling tool. The workshop participants must provide the semantics, and the text must make sense in the context of the domain. The recognition of the text includes the finding within the picture and the recognition of handwritten text.

This approach was created in cooperation with a master thesis at the research group Knowledge Engineer of the University of Vienna [187]. The idea was to identify the relations and information added to it with text recognition to transform the models into other conceptual models. Based on the provided information from the transformation, including the relations, in the master thesis, a transformation from design thinking models to models for the EcoViz modelling tool [188] was created.

The other possibility for automatically creating relations is based on the proximity of two physical objects, which can be visually interpreted as these two objects influence each other. However, with this knowledge alone, it is only possible to create a generic relation between the objects. The knowledge base can enrich the generic relation if a possible relation between the recognised objects is saved. The visualisation of this concept can be seen on the left side of Figure 5.16. This approach was combined with another master thesis at the research group Knowledge Engineer of the University of Vienna [189].

The proximity-based relation identification approach can further be supported if the knowledge base from which the semantic of the relation is derived must not be created by hand but can also be supported by semantic-rich technologies. For example, [190] introduces an approach where knowledge graphs can be generated from text. Such methods can be exploited if a short domain description is available before the workshop. The text can then create a knowledge graph to derive semantic relations between concepts. However, identifying concepts and their relations in the text is still a difficult problem [191].

Figure 5.16 contrasts the two approaches to identify the relations between the physical objects. The top part visualises what is recognised of the physical objects (black boxes) and what is needed for the relation identification (blue parts). The bottom of the figure shows then how it can look in a modelling tool.

5.4. Design Thinking Digital Twin Capabilities for Using the Adaptive Modelling Environment in Workshops

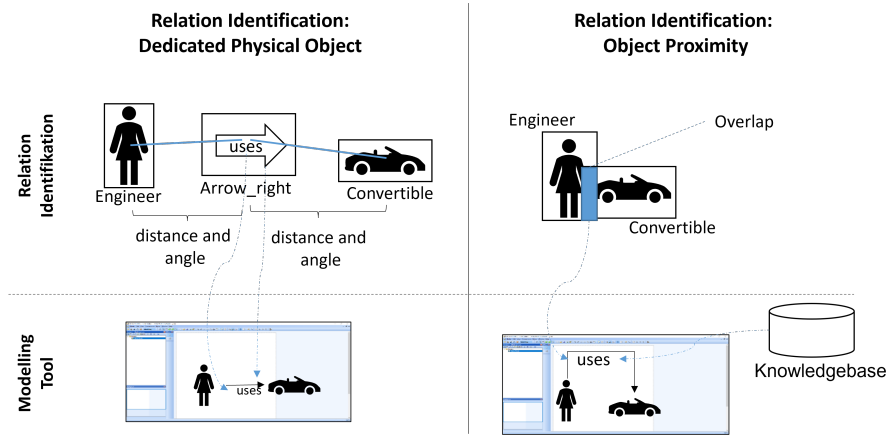


Figure 5.16.: Visualisation of the two concepts of how relations can be recognised in the physical objects

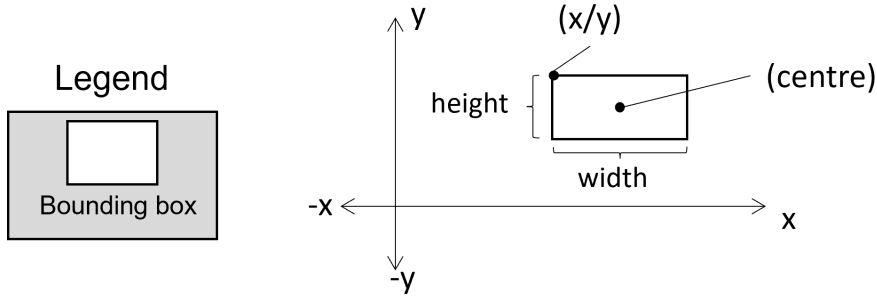


Figure 5.17.: Visualisation of how the coordinate system, the bounding box and the bounding box's parameters are used in this thesis.

5.4.1.1.3. Transformation Algorithm This section introduces the CNN based algorithm for transforming physical objects into digital models. The algorithm procedure will be depicted as pseudo code, and information based on reasoning within the knowledge base will be represented in predicate logic statements.

Before the description of the algorithm is started, basic concepts used in the algorithm will be introduced. As there exist different ways on how the y-axis and parameters of a bounding box can be interpreted, Figure 5.17 will show how they are used for this algorithm. Values on the y-axis will increase upwards on the coordinate system. The max value for y and x are limited by the picture size. The point of a bounding box, annotated with (x,y) representing the top left corner, is used to define the position of the bounding box. *Height* is the length of the side along the y-axis, and *width* will be called the length of the sides along the x-axis. The centre of the rectangle can then be calculated by $x + (w/2)$ and $y - (h/2)$.

The recognition with a CNN applied on a picture will return a set of identified objects

5. The Adaptive Modelling Environment

within the picture. Such an object will pose the following parameters:

- Label: Written text that is assigned to the class based on its training
- Bounding box: The object's bounding box in the picture, with the coordinates of the top left corner and its height and width.
- Confidence: A value between 0 and 1 encodes how certain the net is that the identified object fits the label it was assigned.

The basic structure of the algorithm is visualised in the form of pseudocode in Algorithm 1. The algorithm inputs a picture of the physical objects, a list of the trained CNNs, and a trained model for recognising handwritten text on the figures. The functions used in the pseudocode are explained in the text if they are more generic or in their own pseudocode.

The general procedure of the algorithm is that each provided CNN is applied to the input picture. Afterwards, each recognised object will be assigned an abstraction layer (as introduced in section 5.4.1.1.1), consolidating the identified objects. After the consolidation, the input picture will be enhanced with the recognition. Therefore, for each recognised object, the part of the picture which contains them is marked with the corresponding bounding box. Last but not least, additional information will be derived, and the results will be formatted. This was a very high-level overview, and the next steps of the algorithm, described in pseudocode, will be discussed in more detail. The pseudocode can be seen in Algorithm 1.

The lines 2 to 9 the basic needed arrays are initialised. The CNNs and the text model are loaded as preparation for the algorithm. *foundObjects* will store all the objects, which will be found with a CNN.

The arrays *concreteObjects*, *classObjects* and *genericObjects* are used to store the identified objects, to which the corresponding abstraction layer (see section 5.4.1.1.1).

The *preprocessPicture()* function from line 10 is a generic function which cannot be predefined but must be tailored to the concrete implementation. Therefore, it does not map to a predefined set of adaptations, which must be applied to the picture but should be interpreted as a placeholder, that here, the pictures must be adapted in the algorithm. The concrete adaptations to the picture depend on the concrete implementation and the used CNN. However, examples of typical adaptations are transforming it to another format, resizing it or changing the colours. This preprocessing must be done so that the CNN can be applied to the picture. The prototype used this function to resize the provided picture to fit the size with which the CNN was trained.

The central part of the algorithm starts by taking the provided picture and using it as input for all the loaded CNN. This is represented through line 12 in Algorithm 1. Then, for each object, an abstraction layer (see section 5.4.1.1.1) is identified, which is used to assign them to a set of objects which belong to the same abstraction layer. If no information about the abstraction layer is provided, all the objects will be put in the same set. The abstraction layer prioritises which identification should be deleted if the same objects are recognised multiple times.

5.4. Design Thinking Digital Twin Capabilities for Using the Adaptive Modelling Environment in Workshops

The assignment of the abstraction layer is based on the information available in the knowledge base. The knowledge base contains three layers for which the following classes are defined:

- GenericObject
- ClassObject
- ConcreteObject

These classes are available in parallel to the domain classes used, which contain information on the figures for the domain. The domain classes will be assigned to the abstraction layer classes. Here, the reasoning capabilities will be exploited by assigning the domain classes to the abstraction layer classes with the help of rules. Only the domain class on the generic layer must be assigned manually. The other two can be assigned based on their properties.

Let us assume we use the relation *subclassOf* to define subclasses within the domain classes. We can use this relation to write a logical statement, which can be interpreted as a rule. To do this, we use the predicate *subclassOf(a,b)* to state that *a* is a subclass of *b*.

The two rules are presented in predicate logic in formula 5.1 and formula 5.2. The first says that for every object, which is a subclass of another object, which is *GenericObject*, this is automatically a *ClassObject*. The second follows the same principle, defining a *ConcreteObject* as a subclass of *ClassObject*. A version of this example, which uses RDF, can be found in the appendix in section A.1.1.

$$\forall o \exists g [(subclassOf(o, g) \wedge GenericObject(g)) \Rightarrow ClassObject(o)] \quad (5.1)$$

$$\forall o \exists c [(subclassOf(o, c) \wedge ClassObject(c)) \Rightarrow ConcreteObject(o)] \quad (5.2)$$

If we now use the example, introduce in Figure 5.15, and define it in our knowledge base, as it was done in Listing 5.1.

```

1  GenericObject(object).
2  subclassOf(person,object).
3  subclassOf(car,object).
4  subclassOf(accountant,person).
5  subclassOf(engineer,person).
6  subclassOf(convertible,car).
7  subclassOf(bus,car).
```

Listing 5.1: Example from Figure 5.15 insert in knowledge base

Then the rules defined in formula 5.1 and formula 5.2 will derive the knowledge defined in Listing 5.2. This knowledge can then be used to query the abstraction layers from the knowledge base and assign them to the identified objects.

5. The Adaptive Modelling Environment

```
1 ClassObject(person).
2 ClassObject(car).
3 ConcreteObject(accountant).
4 ConcreteObject(engineer).
5 ConcreteObject(convertible).
6 ConcreteObject(bus).
```

Listing 5.2: Reasoned abstraction layer assignement to the knowledge base defined in Listing 5.1

To use such a knowledge base, the label used by the CNN must match the identifier used in the knowledge base. If RDF is used, this can be achieved by using the URI as a label or adding the label to the concept in the knowledge base.

After identifying the abstraction layer, this information can assign the recognised object to the corresponding set. This information is then used to consolidate the identified objects on the different abstraction layers. Therefore, the consolidation function is called with the three sets of identified objects, shown in line 21 of Algorithm 1. The function is further explained in Algorithm 2.

Consolidating objects is essential, as each physical object should be matched to one object in the digital model. However, using multiple CNNs on one picture, one physical object may be recognised more than once. This could be from trained nets on different abstraction layers or nets of the same abstraction layer. The pseudocode for consolidating all the recognised objects can be seen in Algorithm 2.

The algorithm takes the three arrays with the recognised object on different abstraction layers as input and needs an additional configuration parameter called *overlapDifferenceThreshold*. The algorithm checks which recognised objects reference the same physical object by calculating the significant overlap between their bounding boxes. The *overlapDifferenceThreshold* is a percentage value which defines how significant the overlap between the two bounding boxes can be before two bounding boxes are interpreted as being for the same physical object. The result is a list with all the recognised objects, with removed doubles.

The objects in the list will be compared to each other individually. First, all the objects in the *concreteObjects* list will be compared with the *genericObjects* and the *classObjects* list. Then, each object on the class layer will be compared to each in the generic layer.

First, the overlap between the two objects' bounding boxes will be calculated for each comparison. How the overlap is calculated is visualised in pseudocode of Algorithm 3 and will be discussed later. If the overlap of the objects is greater than zero, then it will be checked how significant the overlap of the objects' bounding box is in the form of their percentage. This is not the same as the result from the *calcualteOverlapArea* function. This function returns the area of the overlap itself but does not relate it to the original bounding boxes of the comparison. This relation is calculated by the *overlapDiff* functions, which get as input the two objects which are currently compared.

The *overlapDiff* function is not separately described in pseudocode as it calculates the ratio between the two areas of the bounding boxes. If this ratio exceeds the *overlapDifferenceThreshold*, the object on the higher abstraction class will be removed from its list.

Algorithm 1: Pseudocode for the CNN based identification of physical objects

Data: *pic*: picture of physical objects**Data:** *avaialbleNets*: set of the available trained CNNs**Data:** *textRecognitionModel*: trained model which can be used to recognise and understand handwritten text in a picture**Result:** *identifiedObjects*: list with the identified objects

```

1 init
2   foundObjects  $\leftarrow$  newArray();
3   concreteObjects  $\leftarrow$  newArray();
4   classObjects  $\leftarrow$  newArray();
5   genericObjects  $\leftarrow$  newArray();
6   nets  $\leftarrow$  newArray();
7   foreach net in avaialbleNets do
8      $\lfloor$  nets.append(net);
9   textRecognitionModel  $\leftarrow$ 
     loadTextRecognitionModel(textRecognitionModel);
10 preprocessPicture(pic);
11 foreach net in nets do
12   foundObjects  $\leftarrow$  net.recognise(pic);
13   foreach obj in foundObjects do
14     assignAbstractionLayer(obj);
15     if obj.abstractionLayer == "ConcreteObject" then
16        $\lfloor$  concreteObjects.append(obj);
17     else if obj.abstractionLayer == "ClassObject" then
18        $\lfloor$  classObjects.append(obj);
19     else
20        $\lfloor$  genericObjects.append(obj);
21 identifiedObjects  $\leftarrow$ 
     consolidateFoundObjects(concreteObjects, classObjects, genericObjects);
22 foreach obj in identifiedObjects do
23   pic.draw(obj.boundingBox, obj.label);
24   obj.icon  $\leftarrow$  cutIconFromPic(pic, obj.boundingBox);
25   obj.text  $\leftarrow$  textRecognitionModel.recogniseText(obj.icon);
26 identifyRelationsDedicatedRelations(identifiedObjects);
27 foreach obj in identifiedObjects do
28    $\lfloor$  obj.loadAdditionalInformation();
29 identifiedObjects.append(identifyRelationsProximity(identifiedObjects));
30 formatResult(identifiedObjects);

```

5. The Adaptive Modelling Environment

The duplicates on the same abstraction layer will be removed for the next step. This is important as multiple CNNs can be used on the same abstraction layer. After only one recognised object per physical figure is left, the array will be joined and returned. The function *removesDuplicate* is described in Algorithm 4.

Next, we will discuss how the overlap between two bounding boxes will be calculated, shown in the pseudocode of Algorithm 3. This function consumes two bounding boxes, with their x , y , w and h properties as input and returns the size of the overlapping area between them if it exists. The x and y properties encode the coordinates of the top left corner of the bounding box, and w contains the width and h the height. These parameters are visualised in Figure 5.17.

The overlapping algorithm compares the sides of the rectangle with each other. First, the left side of *bounding box 1* is compared with the left side of *bounding box 2*, then the top side of the two bounding boxes, which is repeated for each side. If the boxes do not overlap, the width or height of the overlapping area will get a negative value, indicating that the two bounding boxes are not overlapping.

Next, we will discuss how the issue of recognised object duplicates on the same abstraction layer can be resolved. The approach is visualised in the pseudocode of Algorithm 4. The function needs three types of data to work. First, it needs the list of recognised objects, with the duplicates removed. Secondly, it needs the *overlapDifferenceThreshold* parameter, used in Algorithm 2. It provides a threshold for how big the overlap between two bounding boxes is allowed to be before it is interpreted as the same recognised object. Thirdly, it needs information on how to decide which of the duplicates should be removed.

The primary and domain-independent way to decide which recognised object on the same abstraction layer should be kept is to use the confidence of the recognition. The recognised objects with a higher confidence are preferred. As mentioned at the beginning of section 5.4.1.1, this is one part where the transformation can be improved if additional information is available. This is encoded in the third input parameter for this function, which is called *workshopDomains*.

This parameter contains a list of domains in which the workshop is held. The knowledge base must also assign a domain to the recognised objects to use this information. Then, the workshop domain can be compared to the domain of the concepts, and use this knowledge to decide which duplicates to remove.

The function compares each object in the provided list with another object in the same list. The comparison of a recognised object with itself will be skipped. If the two objects should be compared, then the overlap calculation function from Algorithm 3 will be used. If the two objects overlap and the overlapping ratio of the bounding boxes of these two is greater than the *overlapDifferenceThreshold*, then the comparison will be executed.

The comparison starts by retrieving the context of the recognised object from the knowledge base. The domain knowledge here uses a tag-based approach, where tags for domains are assigned to the recognised objects and the workshop. A matching between the domains of the workshop and those of the recognised object prioritises this object above others with less fitting domains. To achieve this, users must add this knowledge to the workshop preparation. The domain tags for the workshop must be provided in a list, as

Algorithm 2: Pseudocode for the *consolidateFoundObjects()* function from Algorithm 1. The functions used in this algorithm can be found in Algorithm 3 and Algorithm 4.

Data: *concreteObjects*: list with the identified object, to which the “concreteObject” abstraction level was assigned

Data: *classObjects*: list with the identified object, to which the “classObject” abstraction level was assigned

Data: *genericObjects*: list with the identified object, to which the “genericObject” abstraction level was assigned

Data: *overlapDifferenceThreshold*: percentage threshold for when the overlap is interpreted as the same physical object

Result: *consolidatedObjects*: list with objects, where multiple identifications for the same physical objects are removed

1 Function

```

consolidateFoundObjects(concreteObjects, classObjects, genericObjects):
2   foreach conObj in concreteObjects do
3       foreach genObj in genericObjects do
4           overlap ←
                calculateOverlapArea(conObj.boundingBox, genObj.boundingBox);
5           if overlap > 0 then
6               if overlapDiff(conObj, genObj) > overlapDifferenceThreshold
7                   then
8                       genericObjects.remove(genObj);
9
10          foreach classObj in classObjects do
11              overlap ←
                    calculateOverlapArea(conObj.boundingBox, classObj.boundingBox);
12              if overlap > 0 then
13                  if overlapDiff(conObj, classObj) > overlapDifferenceThreshold
14                      then
15                          classObjects.remove(classObj)
16
17          foreach classObj in classObjects do
18              foreach genObj in genericObjects do
19                  overlap ←
                        calculateOverlapArea(classObj.boundingBox, genObj.boundingBox);
20                  if overlap > 0 then
21                      if overlapDiff(conObj, genObj) > overlapDifferenceThreshold
22                          then
23                              genericObjects.remove(genObj)
24
25          consolidatedObjects.append(removeDuplicates(concreteObjects));
26          consolidatedObjects.append(removeDuplicates(classObjects));
27          consolidatedObjects.append(removeDuplicates(genericObjects));
28          return consolidatedObjects

```

5. The Adaptive Modelling Environment

Algorithm 3: Pseudocode for the *calculateOverlapArea()* function, which is used in Algorithm 2

Data: *boundingbox*: rectangular boundinbox of the object

Result: *overalpArea*: the area of the overlap between the two boundingboxes

```

1 Function calculateOverlapArea(boundingbox1,boundingbox2):
2    $x\_new \leftarrow \max(\text{boundingbox1}.x, \text{boundingbox2}.x);$ 
3    $y\_new \leftarrow \min(\text{boundingbox1}.y, \text{boundingbox2}.y);$ 
4    $x2\_new \leftarrow \min(\text{boundingbox1}.x + \text{bouboundingbox1}.w, \text{boundingbox2}.x +$ 
       $\text{boundingbox2}.w);$ 
5    $y2\_new \leftarrow$ 
       $\max(\text{boundingbox1}.y - \text{boundingbox1}.h, \text{boundingbox2}.y - \text{boundingbox2}.h);$ 
6    $w\_new \leftarrow x2\_new - x\_new;$ 
7    $h\_new \leftarrow y\_new - y2\_new;$ 
8   if  $w\_new > 0$  AND  $h\_new > 0$  then
9     return  $w\_new * h\_new$  ;
10  else
11    return 0 ;

```

one workshop can belong to more than one domain.

Domain tags in the knowledge base can be assigned directly to a concept, or rules can be used to assign tags based on the context of the concept. The rules support users by reducing their manual tasks by automatically deriving knowledge the users can then exploit. However, one concept can also have multiple domain tags, as it can be used in different domains.

The context for deriving domain tags can be either provided by the context in which the concept is added or by its super concepts. The discussion of the rules and the knowledge base will be based on the example concepts which are introduced in section 5.4.1.1.1 and visualised in Figure 5.15. It is the same which was used for the knowledge base discussion for the *assignAbstractionLayer*. In the discussion and the provided logical statements, the *hasDomainTag(a,b)* predicate will be used to assign *a* the domain tag *b*.

To summarise, the three ways of assigning domain tags to concepts, which will be discussed in this thesis, are:

- Directly assigned domain tag: The user manually assigns the domain tag to a concept
- Inherited domain tag: The concept inherits the domain tag from its super concepts
- Context-based domain tag: The domain tag is assigned to the concept because it is used in a certain context.

The first way is straightforward, as this information is provided as a fact to the knowledge base, where not much information can be reasoned about the concept. We can

5.4. Design Thinking Digital Twin Capabilities for Using the Adaptive Modelling Environment in Workshops

use these statements to automatically reason that the domain tag belongs to the class of domain tags, allowing us to query all domain tags. This set of available domain tags can then be used to suggest defining domain tags for a workshop. This rule is described in logical statements in formula 5.3.

$$\forall d \exists o [hasDomainTag(o, d) \Rightarrow DomainTag(d)] \quad (5.3)$$

The following way to define domain tags is through inheritance. The assumption is that if a concept is connected to a specific domain, its children are also connected. Our knowledge base has a hierarchical structure, so the domain tags should be passed along over multiple hierarchy levels. Therefore, we must define our *subclassOf* relation as transitive. The transitivity definition in logical statements is shown in formula 5.4. Formula 5.5 shows the logical statement for deriving the domain tag of a concept from its super concept.

$$\forall a, b, c [(subclassOf(a, b) \wedge subclassOf(b, c)) \Rightarrow subclassOf(a, c)] \quad (5.4)$$

$$\forall o \exists p, d [(subclassOf(o, p) \wedge hasDomainTag(p, d)) \Rightarrow hasDomainTag(o, d)] \quad (5.5)$$

The last way of assigning domain tags, which will be discussed in this transformation algorithm, is through the context in which a concept is used. The context here refers to the workshop or project in which this concept was added. Here, the application for managing the concept should add all concepts created for the same purpose in the same group, and then the domain label can be added to the group and subsequently assigned to all its members. Therefore, we will use the *belongTo(a, b)* predicate to assign the *a* to a member of the group *b*. This reasoning rule is visualised as a logic statement in formula 5.6.

If the automated assignment to the group and the rule of formula 5.6 are in place, then the domain tag will be assigned to all concepts, and then the domain tag can be queried and used to decide which duplicate should be removed.

$$\forall o \exists g, d [(belongTo(o, g) \wedge hasDomainTag(g, d)) \Rightarrow hasDomainTag(o, d)] \quad (5.6)$$

Listing 5.3 shows an example of applying the before-introduced rules. Lines 1 to 6 show the data that must be provided. The assumption is that the application managing the knowledge base adds lines 4 to 6 because the workshop context was defined. Then, with the rules, the facts from lines 8 to 13 can be automatically derived. The facts in lines 8 to 10 are based on formula 5.3. The *smartCity* tag for *engineer* (line 11) can be derived with formula 5.5. Lines 12 to 13 can be derived based on formula 5.6.

5. The Adaptive Modelling Environment

```
1 subclassOf(engineer, person).
2 hasDomainTag(engineer, production).
3 hasDomainTag(person, smartCity).
4 hasDomainTag(workshop1, publicTransportation).
5 belongTo(engineer, workshop1).
6 belongTo(person, workshop1).
7
8 DomainTag(production).
9 DomainTag(smartCity).
10 DomainTag(publicTransportation).
11 hasDomainTag(engineer, smartCity).
12 hasDomainTag(engineer, publicTransportation).
13 hasDomainTag(person, publicTransportation).
```

Listing 5.3: Examples of assigning domain tags, based on the example concepts from section 5.4.1.1.1

If all domain tags are queried from the knowledge base in Listing 5.3, the result would be: *production*, *smartCity* and *publicTransportation*. However, none of these facts was manually defined as *DomainTag*. An example of this using RDF can be found in the appendix in section A.1.2.

In the transformation algorithm, the duplicate removal can also be done with another sub-algorithm for comparing the domain of the objects and the workshop as long as the sub-algorithm allows a comparison between the two objects. The above-introduced sub-algorithm was chosen because it provides additional semantics to the workshop while keeping the additional workload of assigning tags to concepts minimal. Additionally, it does not rely on an intricate ontology to increase the expressiveness.

In the *removeDublicates* function of Algorithm 4, the amount of tags assigned to each concept is then used to decide which duplicates should be kept and which should be removed. This is done by creating the intersection of domain tags between each object and the domain tags provided for the workshop. The intersection is the number of tags used for the object and the workshop. The object for which the intersection is higher will be kept, and the one with the lower intersection value will be removed.

If the intersection of domain tags is the same for both objects, including if no fitting domain tags were found, then the confidence of the CNN decides which object is kept. Here, the object with the higher confidence value wins.

The next step in the general transformation algorithm (depicted in Algorithm 1) is manipulating the digital picture. This includes drawing the bounding boxes and the labels of the recognised objects on the input picture. Through this, the picture with the visual information can be shown to the human users, who can use this information for controlling purposes and check if everything is correctly recognised.

The second part of this step is that the recognised objects in the pictures are cut out and attached to the object's representation within the algorithm. The bounding box of the recognised object is used to cut this part of the picture and save it to the recognised object. Therefore, each recognised object within the algorithm contains a picture of the object used in the physical model.

Cutout pictures serve two purposes. One is that this information can be sent to the

Algorithm 4: Pseudocode for the *removeDuplicates()* function, which is used in Algorithm 2

Data: *objectlist*: list with identified object

Data: *overlapDifferenceThreshold*: percentage threshold for when the overlap is interpreted as the same physical object

Data: *workshopDomains*: list with domains, which are assigned to the workshop

Result: *alteredObjectList*: list with objects, where multiple identifications for the same physical objects are removed

```

1 Function removeDuplicates(objectList):
2   foreach obj in objectList do
3     foreach compare in objectList do
4       if obj == compare then
5         | skip;
6       overlap ←
7         calculateOverlapArea(obj.boundingBox, compare.boundingBox);
8       if overlap > 0 then
9         if overlapDiff(obj, compare) > overlapDifferenceThreshold then
10          | obj.domains ← getDomains(obj);
11          | compare.domains ← getDomains(obj);
12          | domainIntersectionObj ←
13            buildIntersection(obj.domain, workshopDomains);
14          | domainIntersectionCompare ←
15            buildIntersection(compare.domain, workshopDomains);
16          | if len(domainIntersectionObj) >
17            len(domainIntersectionCompare) then
18              | objectList.remove(compare);
19          | else if len(domainIntersectionObj) <
20            len(domainIntersectionCompare) then
21              | objectList.remove(obj);
22          | else if obj.confidence > compare.confidence then
23            | objectList.remove(compare);
24          | else
25            | objectList.remove(obj);
26       |
27     |
28   |
29 return objectList

```

5. The Adaptive Modelling Environment

modelling tool. Therefore, if the modelling tool does not yet have a class for representing the recognised object, it can generate a generic object and dynamically set the sent picture as its notation. The generic object follows the instance based metamodeling approach introduced in section 5.1.2.

The second reason why the picture is cut out is so that it can be further processed to extract additional information. The cutout pictures are used for text recognition based on the loaded text recognition model. The goal is for handwritten text added to the physical object to be recognised and assigned to the object. The cutout picture is used so that parallelisation of the recognition is possible, and the text can easily be assigned to the recognised object without the need for a matching based on position.

Two steps must be fulfilled to recognise the handwritten text. The location of the handwritten text must be found, and then the text itself must be identified. The position is needed so that the text can be cut out, and only the part of the picture with the text is used for the recognition. Similar to object detection, different approaches can be used to understand handwritten text.

The difference to object detection in the context of this thesis is that the recognition of handwriting must not be tailored to the domain of the workshop, as recognising handwriting is a generic task. Therefore, if a working predefined model for text recognition is found, it can be used for multiple workshops. This means that the adaptation and usage must not be adapted for this algorithm.

The next step in the overall transformation algorithm, as visualised in Algorithm 1, is identifying relations between the used physical objects. Therefore, two approaches will be used, which are introduced in section 5.4.1.1.2. First, relations will be identified based on dedicated physical objects, which the workshop participants place. This is done with the *identifyRelationsDedicatedRelations*, used in Algorithm 1 and visualised in pseudocode in Algorithm 5. These dedicated physical objects must also be identified by the CNN and then processed to check if they can be used as relations in the concrete model. This algorithm will be discussed in the following.

The function defining this part of the algorithm needs four parameters: the list of recognised objects, a list of objects which can be interpreted as relations and the two parameters *minDegree* and *minRatio*. The last two are configuration parameters, which define the positional restrictions for the relations. The second input list is also a configuration parameter, containing the list of objects that can be interpreted as relations. This means that if the physical space's positional restrictions are met, they will be interpreted as relations. Otherwise, they will be used as regular objects and presented as modelling objects in the modelling tool.

The first step is to split the provided list with the recognised objects into those objects that can be relations and those that can not. This separation is needed to compare the relation candidates to the other recognised objects. The separation is done by creating a new list containing all recognised objects minus those which can be relations. The remaining objects could be relations in the specific case and are saved in the *relationCandidates* list.

Then, each object within the *relationCandidates* list is compared with the objects

without relation candidates. The comparison consists of three objects: the relation candidate and two non-relation candidates as the endpoints.

A relation candidate will be used as a relation if it lays generally in the middle between two non-relation objects on an approximately straight line. The parameters *minDegree* and *minRatio* define how far the constellation of the three objects can differ from the exact middle and an absolute straight line, allowing flexibility in identifying relations.

First, the angle between the assumed straight lines between the relation candidate and the other two objects is calculated and compared to the *minDegree* parameter to evaluate these relations requirements. If the angle fits, the ratio between two assumed lines is calculated and compared to *minRatio*. This ratio defines if the two lines are approximately the same length, corresponding to the relation candidate being approximately in the middle. The nearest pair is chosen if multiple pairs of non-relation candidates fulfil these requirements for one relation candidate. If the relation candidate fulfils the requirements, then the target and source for the relation will be set. The flag at the recognised object will also be set, identifying this object as a relation.

If the relation and the two fitting endpoints are identified, then the direction of the relation must be determined. The pseudocode for this decision is shown in Algorithm 6. To define the direction of a relation, the relation candidates must also possess information about their direction. For the physical objects, four directions are possible, which must be saved to them:

- up
- down
- right
- left

The name of the direction always points in the direction of the relation. If no direction is provided, then the target and source of the relation will be set randomly. The algorithm itself checks based on the direction of the relation candidate and the position of the two related objects, which object should be set as a target and which as a source of the relation. For the *right* and *left* relation candidates, the *x* coordinate of the objects is used. For the *up* and *down* relation candidates, the *y* coordinate of the objects is used.

As the next step, additional information for the recognised object (including the identified relations) is loaded from the knowledge base, if available. Here, the focus is set on attributes, which are then used in the modelling tool. The attributes can either be on the class or instance level. On the class level, the attributes will be added to the classes in the modelling tool, but the values in the created model objects are left empty. The users can then fill in the information in the workshop or afterwards. If instance information is provided, the attributes will also be filled out in the created modelling objects. If no additional information is provided in the knowledge base, the modelling objects can still be created, but the users must manually fill in all the information.

5. The Adaptive Modelling Environment

Algorithm 5: Pseudocode for identifying relations from physical objects and their position

Data: *objectList*: list of objects, which should be checked for possible relations
Data: *possibleRelationCandidates*: list of all possible objects, which can be transformed into relations
Data: *minDegree*: configuration parameter for limiting the degree the vectors between the relation candidate and two objects (between a relation should be established) can take
Data: *minRatio*: configuration parameter for defining the minimal ratio which must exist between the distance from the two objects to the relation candidate

Result: *relationList*: list of the identified relations

```

1 Function identifyRelations(objectList):
2   notRelCandidates  $\leftarrow$  (objectList - possibleRelationCandidates);
3   relationCandidates  $\leftarrow$  (objectList - notRelCandidates);
4   foreach RelCan in relationCandidates do
5     foreach obj1 in notRelCandidates do
6       foreach obj2 in notRelCandidates do
7         uv1  $\leftarrow$  calculateUnitVector(obj1, RelCan);
8         uv2  $\leftarrow$  calculateUnitVector(obj2, RelCan);
9         angle  $\leftarrow$  calculateAngleBetween(uv1, uv2);
10        if angle > minDegree then
11          distance1  $\leftarrow$  distanceBetween(RelCan.center, obj1.center);
12          distance2  $\leftarrow$  distanceBetween(RelCan.center, obj2.center);
13          distanceRatio  $\leftarrow$  calculateRatio(distance1, distance2);
14          if distanceRatio > minRatio then
15            averageDistance  $\leftarrow$  avg(distance1, distance2);
16            if averageDistance < RelCan.nearestFigureDistance
17              then
18                RelCan.nearestFigureDistance  $\leftarrow$  averageDistance;
19                setRelationEndpoints(RelCan, obj1, obj2);
20                RelCan.isRel = true;

```

Algorithm 6: Pseudocode for setting the endpoints of an identified relation from Algorithm 5

Data: *relationObject*: physical object, which was mapped to a relation and for which the endpoints should be set

Data: *object1*: physical object, which was identified as a relation to *relationObject*

Data: *object2*: physical object, which was identified as a relation to *relationObject*

Result: *relationObject*: with set endpoints

```

1 Function setRelationEndpoints(relationObject,object1,object2):
2   if relationObject.direction == "right" then
3     if obj1.x > obj2.x then
4       relationObject.target  $\leftarrow$  obj1;
5       relationObject.source  $\leftarrow$  obj2;
6     else
7       relationObject.target  $\leftarrow$  obj2;
8       relationObject.source  $\leftarrow$  obj1;
9   else if relationObject.direction == "left" then
10    if obj1.x > obj2.x then
11      relationObject.target  $\leftarrow$  obj2;
12      relationObject.source  $\leftarrow$  obj1;
13    else
14      relationObject.target  $\leftarrow$  obj1;
15      relationObject.source  $\leftarrow$  obj2;
16  else if relationObject.direction == "up" then
17    if obj1.y > obj2.y then
18      relationObject.target  $\leftarrow$  obj1;
19      relationObject.source  $\leftarrow$  obj2;
20    else
21      relationObject.target  $\leftarrow$  obj2;
22      relationObject.source  $\leftarrow$  obj1;
23  else if relationObject.direction == "down" then
24    if obj1.y > obj2.y then
25      relationObject.target  $\leftarrow$  obj2;
26      relationObject.source  $\leftarrow$  obj1;
27    else
28      relationObject.target  $\leftarrow$  obj1;
29      relationObject.source  $\leftarrow$  obj2;
30  else
31    relationObject.target  $\leftarrow$  obj1;
32    relationObject.source  $\leftarrow$  obj2;

```

5. The Adaptive Modelling Environment

The next step for the list of identified objects is identifying relations based on their proximity. This is shown in the overall algorithm of Algorithm 1 with the function *identifyRelationsProximity* and is visualised as pseudocode in Algorithm 7.

The function consumes a list of recognised objects and compares them to each other if the visualisation of the objects overlaps. If an object should be compared with itself, then this comparison will be skipped. Otherwise, the overlap between the objects is calculated using the *calculateOverlapArea* function, defined in Algorithm 3. If there is an overlap between the objects, then a generic relation will be created between them. The knowledge base is also searched if additional information about the relation between these two objects is available.

If additional information is available for the new relation, one crucial information is the direction of the relation. Another essential aspect of the relation is its semantics. This can be encoded in its notation, type or label, which the knowledge base can provide. Additionally, parameters can also be provided and added to the relation. If no additional information is available in the knowledge base, then at least a generic relation will be provided, but without additional semantics.

The relations in the knowledge base must not be defined directly between the recognised object. The hierarchical structure within the knowledge base can also be used to retrieve relations defined between the superclasses of the recognised objects.

Finally, the relation is added to the list of the identified relations, which is then returned and added to the list of identified objects.

The last step of the overall transformation algorithm introduced in Algorithm 1 is the formatting of the result. This is achieved by the function *formatResult()*, which consumes the list of identified objects and represents them in a way which can be consumed by the modelling tool. This representation should be in a computer consumable format, e.g., JSON or XML and then offered to the modelling tool. The modelling tool will consume the information, create the models, and adapt the metamodel where necessary. For the algorithm itself, it is not essential how this formatted result looks precisely, but that it can be understood by the modelling tool.

5.4.1.1.4. Preparing the Machine Learning Based Object Detection To enable the modelling environment to automatically transform the physical objects into their digital counterpart, the used physical figures must be identified. The transformation algorithm in this section uses CNN based object detection (see section 2.4.2 for background). This subsection now discusses how this kind of recognition can be prepared for use in a workshop.

Different artefacts must be created and made available for the modelling environment. We will call the collection of these artefacts *icon library*, and its conceptual structure is shown in Figure 5.18. The icon library will be created for a specific workshop and will contain all the information needed to hold this workshop. It will contain the objects the workshop organiser prepares, but not all the prepared objects must be used in the workshop models.

The icon library contains the class hierarchy of the concepts defined in the knowledge

Algorithm 7: Pseudocode for identifying relation based on proximity and pre-defined knowledge

Data: *identifiedObjects*: list of the identified objects and their position in the picture

Data: *knowledgebase*: knowledgebase with the additional knowledge, which is needed to add semantics to the relation between two objects

Result: *identifiedRelations* list of the identified relations, with the added semantic

```

1 Function identifyRelations(objectList):
2   foreach obj1 in identifiedObjects do
3     foreach obj2 in identifiedObjects do
4       if obj1 == obj2 then
5         | skip;
6       overlap ←
          calculateOverlapArea(obj1.boundingBox, obj2.boundingBox);
7       if overlap >= 0 then
8         newRel ← createRelation(obj1, obj2);
9         relationInfo ← loadRelationInfo(obj1, obj2);
10        if newRel != null then
11          newRel.setDirection(relationInfo.direction);
12          newRel.setRelationInformation(relationInfo);
13        identifiedRelations.append(newRel);
14  return identifiedRelations

```

5. The Adaptive Modelling Environment

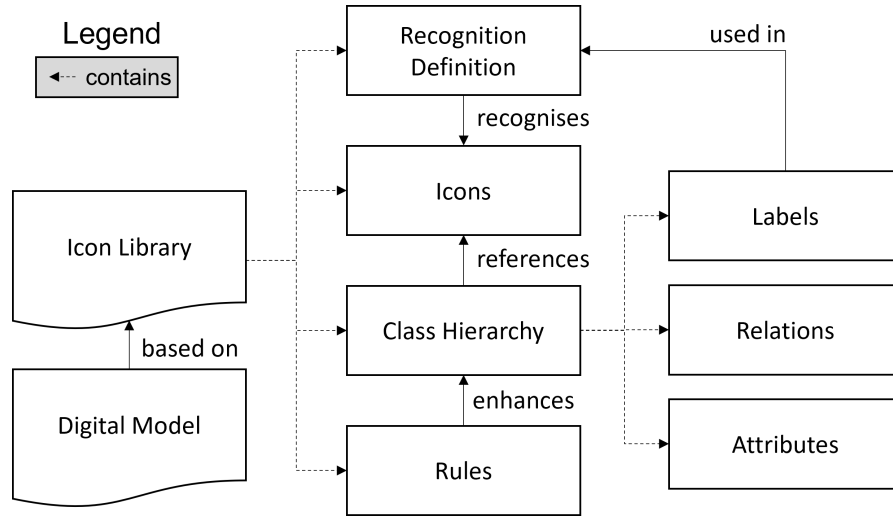


Figure 5.18.: Conceptual representation of the artefacts which are needed as input for the modelling environment to use the machine learning based object identification approach

base and, therefore, contains the basic conceptual structure of the prepared objects. This hierarchy contains the abstraction layers for the recognition as introduced in section 5.4.1.1.1. The classes defined in the hierarchy can also hold additional information like relations and attributes, which can be utilised during the transformation process. Additionally, rules can be defined, which allow the automatic reasoning of additional information.

The class hierarchy and the rules represent the knowledge base, which was discussed in the transformation algorithm (see section 5.4.1.1). The minimal information that must be provided is the labels, defining the recognised objects' names. They map to the metamodel so the information can be used in the modelling tool. However, the additional information which can be provided allows an enhancement of the automatically created semantics in the models.

The class hierarchy can also reference icons, which should be used in the modelling tool. This can be pictures of the object or figure they should graphically represent. These are usually not just simple geometry forms, like rectangles or circles, but complex figures that can convey semantics to humans. The icons will be prepared physically and used in the workshop, and then they should be recognisable by the trained neural network. In this way, not only the humans understand the figures, but also the machine.

For the workshop environment, other needed artefacts are the trained neural networks. The *Trained CNN* concept in Figure 5.18 stands for the already trained networks. These must be provided as input to the transformation algorithm. The labels of the CNNs must fit into the class hierarchy and subsequently to the metamodel of the modelling tool.

The combination of these artefacts will be called the icon library. The icon library can then also be imported into other modelling tools and, in this way, be used by different

5.4. Design Thinking Digital Twin Capabilities for Using the Adaptive Modelling Environment in Workshops

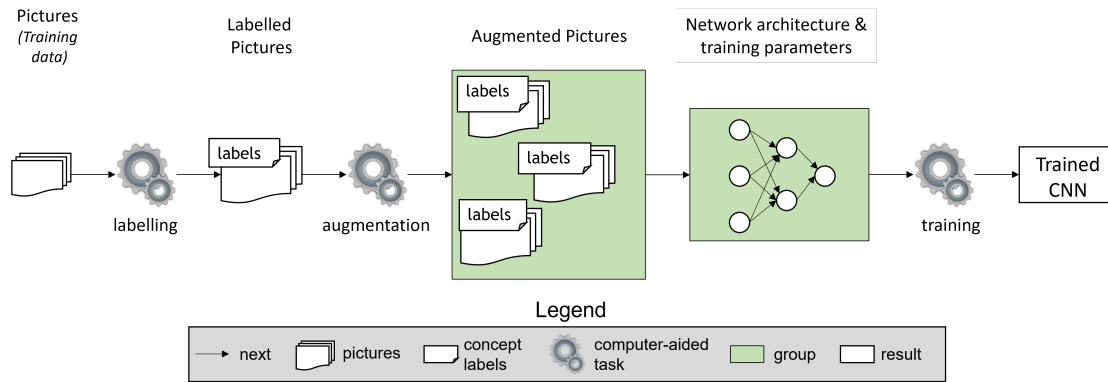


Figure 5.19.: Graphical representation of the training process to create a trained CNN

users. If the digital models should be shared with involved stakeholders, then the icon library must also be sent with it. Therefore, the icon library and the digital model must be shared.

The icon library is the artefact that the workshop organiser must prepare in advance. Already existing icon libraries can be reused for new workshops. For the sake of this introduction, we assume that the organiser creates an icon library from scratch.

The first task that must be done is for the workshop organiser to understand the domain in which the workshop is held. Here, no deep understanding is necessary, but general knowledge about what should be discussed in the workshop is needed. This understanding is important so the workshop organiser can identify the needed concepts and find icons for them. After this step, the organiser should have a set of icons that can be used for the workshop.

These icons must be printed and cut out for the workshop. Additionally, the cutout icons will be used as input for training the CNN. Therefore, a training set must be created by taking pictures of the cutout icons. To improve the later recognition, the pictures of the figures should be made in a similar environment, as they are later used in the workshop. A similar background improves the recognition results if only a few pictures are made. Each icon should be photographed individually and in combination with other icons.

The general training procedure proposed in this thesis is visualised in Figure 5.19. After pictures are made, they must be prepared for the training. This means that first, they must be labelled. Labelling means an application is used to draw bounding boxes around the concepts and add the label to identify the concept. Each occurrence of a concept in the pictures must be labelled. Additionally, some empty backgrounds should also be added to the training data. The result of this step is a set of pictures, with the labels and bounding boxes as training data.

Data augmentation will be used to reduce the number of pictures which must be taken for the training and subsequently reduce the workload of the workshop organiser (see section 2.4.2 for details). This means that the available training data is multiplied, but

5. The Adaptive Modelling Environment

the pictures are copied and automatically adapted to create new pictures. The already available bounding boxes are also transformed to fit the adapted pictures. Examples of augmentation are making the picture blurry, cutting out parts of them or changing their perspective.

After the inflated training data is created, the architecture and configuration of the CNN must be defined (see section 2.4.2 for more information). Afterwards, the training must be started, and at the end, the trained net is available, which can be used in the workshop.

As mentioned in section 5.4.1.1.1 CNNs on different abstraction layers can be used within the introduced transformation algorithm. If the labelling of the pictures is done on the lowest abstraction layer (concrete object layer), then the training data can be reused to train the neural networks on the higher abstraction layers.

This can be automated through a script. Here, the workshop organiser must define a mapping between the labels, from the concrete objects to the class objects. For each label of concrete objects, a label for the class objects must be defined, with which it is replaced. The same must be repeated for a mapping to the generic object layer. This new training data can be created and used in the procedure introduced in Figure 5.19. If this is done, the workshop organiser has three trained CNN which can be used in a workshop.

The labels defined in the training represent the available concepts for which additional information can be provided in the knowledge base. The used label should be added to the class hierarchy. Then, references to default icons should be set, which are used as a notation of this concept in the modelling tool. Therefore, the class hierarchy information must be made available for the modelling environment.

Using trained networks on different abstraction layers allows for reusing the rules defined in section 5.4.1.1.3. Then, the different abstraction layers of recognised objects can be automatically defined. After class hierarchy is defined in the knowledge base, additional information can be added as more rules, attributes or relations. If this is defined, they can be automatically used in the workshop.

The here introduced information contains the concept of the transformation from physical to digital models in the context of the *Adaptive Modelling Environment*. However, further improvements to support users can be made in the future. For example, as discussed in section 5.1.3.2, the workshop organiser can use a short domain description of the workshop participants to prepare the workshop. Semantic-rich technologies can further support the workshop organiser by suggesting concepts, their attributes and their relation. Such a suggestion system can be built upon Natural Language Processing (NLP), which can identify concepts in a text and provide additional information about them. Even though the automatic recognition of relations in the text is still a challenging task [191], there is work done in this direction [190], which makes a reasonable possibility for the future. However, even though it is not easy to automatically extract relations, one can extract the used concepts and create a suggestion for classes automatically.

5.4.1.2. Identifying Tangible Objects with Pre-Defined Tags

An alternative to identifying tangible objects with machine learning is using pre-defined tags. In this alternative, the system for identifying the physical objects is not adapted to enable a finding of these objects in a picture like it is done by training and using a CNN, as introduced in section 5.4.1.1. Here, pre-defined tags will be attached to the physical objects, and the system can identify the tags in a picture. Using such an approach eliminates the need in the preparation to adapt the system (e.g., the training of CNN). However, a mapping of the predefined tags to the used objects must be established within the system.

The mapping of the objects to the tags must be established for the object identification component of the *Adaptive Modelling Environment* and the physical objects used in a workshop. Therefore, the physical attached tags must be the same as the one defined in the mapping. As such tags do not contain semantics visually interpretable by humans, physical objects in the workshops cannot be replaced with the tags but must be enhanced with them. This is important to keep the physical understandable by humans to facilitate their working with them. Then, the humans can interpret the visual figure, and the computer can understand the attached tag.

By using attached tags, the workshop participants must consider the visual representation and the visibility of the tags for the system. Using CNN, a system is enabled to visually interpret the same pictures as the humans, and therefore, if the participants create the visual interpretation so that they can understand it, the system can use the same view. However, if pre-defined tags are used, they are not part of the visualisation for human users, and therefore, they are not automatically considered. This poses additional effort for the participants.

For example, let us assume we use a paper figure-based storyboarding approach (as introduced in section 2.1.2.3) in a workshop. Then, the pre-defined tags must be printed and attached to the paper figures. Then, when placing the paper figures, the participants must keep in mind the position of the tags, or the transformation cannot be correctly executed.

The approach introduced in this section is not limited to one specific kind of tag. As long as the system can recognise the tags without training, their position in the picture can be calculated, and they contain information that can be mapped to objects in the modelling tool. Examples would be bar codes, QR codes or fiducial markers. In this thesis, we will focus on fiducial makers as computer systems can quickly identify them, and they are often used for positioning in computer vision [8]. One example of such markers is introduced by [8] and was implemented in the *ArUco* library. A visual example of such tags can be found in Figure 5.20.

The general procedure is similar to the one introduced for the machine learning based algorithm (see section 5.4.1.1). However, with tag recognition, the further processing of singular pictures of identified figures is not possible straightforwardly, as only the tag and its position are recognised and not directly the figure. Therefore, text on the physical figures cannot be identified without additional recognition steps. Additionally, the definition for the identification of the relations may be influenced, as tags sometimes

5. The Adaptive Modelling Environment



Figure 5.20.: Example of tags created with ArUco [8]

only provide an integer (like the ArUco fiducial markers [8]). Then, the numbers of the tags, which do not hold semantics of the domain, must be used as input for relation identification.

In Algorithm 8, the pseudocode for the tag recognition is shown. Here, how the tags are found in pictures is not described, as this depends on which kind of tags are used concretely. The assumption in this case is that existing libraries will be used to find the tags and their positions in pictures.

The algorithm needs two kinds of inputs: the pictures of the physical model and the implementation of the tag recognition based on the types of tags used. First, the taken picture must be processed in a format that can be used for tag recognition, which is carried out subsequently. This recognition returns the list of identifier and their coordinates. Depending on how the coordinates are identified, they may need reformatting to fit the rest of the algorithm. Here, it is assumed that the *tagRecognitionImpl.identifyTags()* function returns the coordinates in the needed format.

Afterwards, additional information for the objects can be loaded, and relations can be identified. In this variant, relations can only be identified based on defining dedicated objects, which should be interpreted as relations. Identifying relations based on overlap is not feasible, as the overlap between the tags and the used physical figures will not be the same. The loading of additional information and the relation identification are similar to the ones discussed for the machine learning based approach (see section 5.4.1.1). However, the used identifier is not a label returned by a CNN, but the tag's identifier. As it is similar to the one introduced for the machine learning based approach, these parts will not be repeated here.

Last but not least, the information gathered from the physical model must be formatted so that it can be interpreted by the modelling tool. The modelling tool interprets this information and maps it to its metamodel to create different objects.

Preparing the usage of the tag recognition approach is also similar to the machine learning based approach (see section 5.4.1.1). No neural network must be trained, but an assignment from the pre-defined tags to the concepts used in the workshop must be established. The concepts must be provided physically and digitally, and the digital and the physical must be enhanced with the tag. For the digital representation of the modelling tool, a mapping must be saved so that the modelling tool knows which objects must be created if a particular tag is used. For the physical representation, the tag must also be printed, cut out and then attached to the physical object so that a camera can see it.

Additional information can be saved in a knowledge base, which can be utilised to create models with more information. Such information must be provided during the

Algorithm 8: Pseudocode for the tag-based identification of physical objects

Data: *pic*: picture of physical objects

Data: *tagRecognitionImpl*: the implementation of the tag recognition, based on the type of tags which are used

Result: *identifiedObjects*: list with the identified objects

```

1 preprocessPicture(pic);
2 identifiedObjects ← tagRecognitionImpl.identifyTags(pic);
3 foreach obj in identifiedObjects do
4   | obj.loadAdditionalInformation();
5 identifyRelations(identifiedObjects);
6 formatResult(identifiedObjects);

```

preparation. However, here again, it is helpful to allow the transformation if only the tag identifier and the coordinates are provided, and all information that can be provided additionally can be seen as a bonus. Allowing the usage with a minimal need for provided information eases the preparation phase and simplifies the usage.

5.4.2. Analyse Feedback of Conceptual Models

A comprehensible visualisation of the idea can be an excellent way to get feedback from all stakeholders, even those who cannot participate in the workshop designing the idea. Their input can be valuable, especially at the beginning of the project, where changes can be applied the easiest.

In this section, we discuss an algorithm with which textual feedback from people can be gathered and processed to create an overview of the general acceptance of the created idea. The modelling tool supports the workshop organiser and the participants by sending pictures of the models to a platform where the involved stakeholders can access and add their textual feedback to the picture. A question or statement should be added to the picture to guide the direction of the feedback. The platform where the picture is posted and the feedback is gathered is independent of the modelling environment. It could be a public or organisation's internal social media platform, where data can be sent and retrieved using an API.

The quick feedback overview will be provided by grouping the provided feedback statements according to their positive, negative or neutral sentiment. The grouping will be done using a sentiment analysis from the field of NLP. The pseudocode visualising this approach can be found in Algorithm 9.

The algorithm uses a dedicated modelling object called *evaluationModellingObject*, which holds all the data for one evaluation and visualises the results. This way, multiple evaluations can be done using the same conceptual model, enabling the use of the same model for different groups with different questions.

The workshop participants or organisers must also provide the statement or questions, which should be posted with the picture to steer the feedback in the right direction. This

5. The Adaptive Modelling Environment

statement is provided as a string and is represented through the *evaluationStatement* variable in the pseudocode. In future improvements, these questions can also be suggested by generative AI, based on the information provided in the model.

The *conceptualModel* variable represents the diagrammatic model for which the evaluation should be done. This algorithm assumes that the model is already finished and prepared for the evaluation.

The *platformConfiguration* parameter stands for the information needed to post or gather data from the chosen platform. Which platform it is used does not matter for the algorithm, as long as there is an API with which posts (including a picture) can be created and the answers to the post can be gathered. What the *platformConfiguration* variable contains precisely depends on the chosen platform, but it possibly must contain the API endpoints or some form of identification.

First, the *evaluationStatement* must be assigned for the specific evaluation, which is done by adding it to the evaluation modelling object. Before starting the evaluation, the model must be visually prepared by hiding all the available evaluation modelling objects. They are not deleted but are not shown in the visualisation of the model. This is because the evaluation modelling objects do not represent the system under study and, therefore, should not influence the feedback.

Then, the picture and the evaluation statement can be posted on the chosen platform by providing the model's picture and configuration information. Subsequently, an identifier for the new post must be saved so that the feedback can later be extracted. This identifier is also saved in the evaluation modelling object so that it is saved in the context of the evaluation. Afterwards, the evaluation modelling objects can be shown again. Now, the users must wait until the involved stakeholders for which the post was made have provided feedback.

When enough stakeholders have provided feedback, their input can be gathered from the platform, analysed, and rendered. The visualisation of the results should fit the evaluation that is made. For example, a pie chart can be used to visualise the shares of the three groups. Concrete feedback should also be made available to the users by saving this information in the evaluation modelling object.

The critical part of the introduced algorithm is separately visualised in pseudocode in Algorithm 10. This is how the information is gathered and processed to fit the visualisation in the model. This function is based on a predefined analysis algorithm, which is reused. In this case, this algorithm can assign sentiment to a sentence (more information on sentiment analysis can be found in section 2.4.4).

First, the answers to the post are loaded from the platform using the saved *postId* and the *platformConfiguration*. Then, for each answer, its sentiment is assigned, and based on the assigned sentiment, the answer will be added to one of the three groups. Each group represents one sentiment, which is later used for the visualisation. Before the result is returned, the size of each group will be specifically saved to the result object. Then, the information is returned, and the modelling tool can visualise the results.

Here, a standardised library or approach can be used, as the sentiment of the feedback does not depend on the domain or system under study. The assumption is that independent

Algorithm 9: Pseudocode for the automated feedback analysis

Data: *evaluationModellingObject*: modelling object which is used to store and visualise the information of the feedback

Data: *evaluationStatement*: the statement or question that should be posted with the picture of the model

Data: *conceptualModel*: the digital, conceptual model which should be analysed

Data: *platformConfiguration*: this variable contains the configuration parameters, which are needed to post the picture and statement

Result: *feedbackOverview*: graphical overview of the feedback results

```

1 init
2   evaluationObject  $\leftarrow$  newEvaluationObject(platformConfiguration);
3   evaluationModellingObject.statement  $\leftarrow$  evaluationStatement;
4   conceptualModel.hideEvaluationObjects();
5   evaluationModellingObject.posted  $\leftarrow$ 
      evaluationModellingObject.makePost(conceptualModel.getPicture(), platformConfiguration);

6   conceptualModel.showEvaluationObjects();
7   wait for feedback;
8   evaluationModellingObject.feedbackData  $\leftarrow$ 
      gatherFeedback(postId, platformConfiguration);
9   evaluationModellingObject.render();

```

5. The Adaptive Modelling Environment

from the domain, we can assign a sentiment to the feedback and use it. How well this assignment works depends on the concrete implementation. It does not matter which implementation is used for the modelling environment as long as the desired results are provided.

Algorithm 10: Pseudocode for the *gatherFeedback()* function of the automated feedback analysis

Data: *postId*: id of the post, so that it can be identified

Data: *platformConfiguration*: this variable contains the configuration parameters, which are needed to interact with the platform

Data: *analyseImplementation*: this is an implementation of the natural language based analysis, which should be applied to the feedback (e.g., sentiment analysis)

Result: *analysedFeedback*: is an object that contains the information from the analysis. It contains the number of members in each sentiment group and arrays with the statements divided per group.

```
1 Function gatherFeedback(postId,platformConfiguration):
2   positiveFeedback  $\leftarrow$  newArray();
3   negativeFeedback  $\leftarrow$  newArray();
4   neutralFeedback  $\leftarrow$  newArray();
5   answers  $\leftarrow$  getAnswersFromPost(postId,platformConfiguration);
6   foreach answer in answers do
7     helpSentiment  $\leftarrow$  analyseImplementation.getSentiment(answer);
8     if helpSentiment == positive then
9        $\lfloor$  positiveFeedback.append(answer);
10    else if helpSentiment == negative then
11       $\lfloor$  negativeFeedback.append(answer);
12    else if helpSentiment == neutral then
13       $\lfloor$  neutralFeedback.append(answer);
14  analysedFeedback.positiveFeedbackCount  $\leftarrow$  positiveFeedback.len;
15  analysedFeedback.negativeFeedbackCount  $\leftarrow$  negativeFeedback.len;
16  analysedFeedback.neutralFeedbackCount  $\leftarrow$  neutralFeedback.len;
17  analysedFeedback.positiveFeedback  $\leftarrow$  positiveFeedback;
18  analysedFeedback.negativeFeedback  $\leftarrow$  negativeFeedback;
19  analysedFeedback.neutralFeedback  $\leftarrow$  neutralFeedback;
20  return analysedFeedback
```

5.4.3. Model Transformation to Enhance captured Knowledge

Enhancement of knowledge in this section does not mean the enrichment of the available information during the transformation from physical to digital models (as introduced

in section 5.4.1), but that the ideas and designs created in workshops are enhanced so that they can be implemented in the real world. Implementation in this context means that software is created or aspects like organisational restructuring or creating work instructions for humans are included. The assumption is that conceptual models of other modelling methods are used to specify this enhanced knowledge.

As the workshop models will be on a high abstraction level to facilitate communication between experts with different expertise, such models cannot be directly implemented. However, this high-level knowledge can be seen as the starting point. Therefore, this section discusses how models from the workshop can be transformed into starting points for other modelling methods. Through this transformation, the manual task of reusing the created workshop knowledge should be reduced.

The proposed transformation will be an exogenous and vertical model transformation [192], meaning that the source and target model will not be instances of the same modelling languages, and the models will not be on the same abstraction level.

The rules will also contain a transformation from one model to another. Cases like combining multiple models into one or creating multiple models out of one, which can be cases of model transformation [192], are not covered in the here introduced rules.

Testing and validation are often seen as an essential aspect of model transformation [192]. However, as we aim not to create an immediately usable model but a starting point for users to model the context in another modelling method, the user must also ensure that the target model is feasible.

To make the transformation approach easier and accessible to the users, pre-defined transformation rules are provided, which can be parametrised by the user. They can be understood as templates, which users can fill out to transform their models of a workshop into models of another modelling language, which they use to enhance the representation of the knowledge.

Using pre-defined rules also reduces the burden for the user, as they do not have to learn separate model transformation language, as discussed in [193]. A method or infrastructure engineer can adapt the predefined set of transformation rules (information on these roles can be found in section 5.1.3.1).

5.4.3.1. Transformation Procedure

This section focuses on the general procedure of the model transformation, which uses rules to specify how the machine creates one model out of the other. The rules are defined in section 5.4.3.2, which provide the type of rules that users can parametrise to create the rule sets applied during the transformation.

This section assumes that the rules are defined before the transformation is executed and that one set of rules is available for each type of rule introduced in section 5.4.3.2.

The first set of rules applied is mapping the models (cf. Listing 5.4). This checks for each model defined as input, aka the source models, if it should be transformed into a model from another modelling method. If such a transformation is executed, the source model's objects, relations and attributes are checked to see if they should be transformed by applying the other rule sets. Thereby, the rule sets for the attribute mapping are only

5. The Adaptive Modelling Environment

applied if a new object or relation is created in the target model. In this way, the context of the objects can be provided to the attribute rules.

Because the rules for mapping the objects and results are applied to the models, the context of the models is not explicitly defined in the rules. If a rule creates a relation, the rule contains the creation of the needed objects. First, the target model is searched to see if a fitting object (based on the name and class) is already available. If this is the case, then the existing object is taken. Otherwise, the needed object is created. This is true for all creation functions. If a similar object or relation (based on name and object class/relation class) already exists, no new object or relation will be created. This can be the case if rules in different rules create a similar object. For example, a similar object is created from the conditional object mapping (cf. Listing 5.10) and object mapping (cf. Listing 5.5).

The more specific transformation rules can contain additional information, so they are executed first. For example, the conditional ones can provide additional attributes and values. Afterwards, the simpler transformation rules will be executed. This also applies to the attributes, where the conditional attribute mapping is first applied and, afterwards, the simpler ones.

The following list contains the specific transformation rules that will be applied first:

- Conditional Object Mapping (see Listing 5.10)
- Conditional Attribute Mapping (see Listing 5.11)
- Conditional Relation Mapping (see Listing 5.12)
- Relation to Object Mapping (see Listing 5.8)
- Object to Relation Mapping (see Listing 5.9)

The next list contains the simpler rules that will be applied last:

- Object Mapping (see Listing 5.5)
- Relation Mapping (see Listing 5.6)

5.4.3.2. Set of Transformation Rules

In this subsection, the rule templates will be discussed. They were chosen as they are understandable and provide the foundation for quickly transforming workshop models to other modelling languages. Additionally, the process for executing the transformation will be discussed.

The rules will be resented in pseudocode. Therefore, the rules are not defined in a way that makes them easily executable by a machine but understandable by humans. The rules are structured in a *if-then-elseif-else* structure. *If* or *elseif* cover the condition which must be fulfilled so that the *then* part is executed. If the *if* part is not triggered, then the *else* part is executed, if available.

Comparison Operator	Explanation
==	true if both sides are the same
!=	true if both sides are not the same
>	true if the left side is greater than the right side
>=	true if the left side is greater or equal than the right side
<	true if the left side is smaller than the right side
<=	true if the left side is smaller or equal than the right side
contains	true if the left side is contained in the right side. This can be that the left substring is within the right string or that the element on the left is contained in the list or array of the right side.

Table 5.2.: List of the available comparison operators for the pseudo rules and their explanation

The rules aim to check for the *if* condition in the source model and then create something in the target model. Text between the < and the > symbol should be interpreted as parameters, which must be provided. Round brackets are used to group statements or visualise function parameters. The “.” operator from the object-oriented paradigm is used to denote that something is an attribute or function of an object. The = denotes the assignment of a variable.

An assumption for the transformation is that all models are provided in a format that the transformation engine can parse and that each object, independent from the modelling method, has a *name* and *position* attributes. The *name* represents an identifier of the objects, and the *position* is the object’s position on the drawing area. Here, the assumption is that the identifier and position are mapped based on the needs of the used modelling environment. Relations have, in addition to a *name*, also the attributes *source* and *target*, representing the beginning and end of a relation.

In the pseudo rules, the function *copyAttributes(<src>,<tar>,attributeMapping)* is used, which represents that all attributes of object or relation <src> are copied to the object or relation <tar>. During this copying, the rules for attribute mapping are applied. Attributes not defined in the mapping will only be copied if the target objects have an attribute with the same name and type as the source object. All other attributes will be ignored.

Table 5.2 provides an overview of operation operators which can be used for the comparison.

To make the rules more understandable, assigning elements, like objects or relations, to a specific model will not be shown. However, it is essential that if an object is mapped

5. The Adaptive Modelling Environment

from one model to another, the transformation engine assigns the new object to the new model. However, this information is not visualised to avoid overloading the pseudo rules.

The first rule (see Listing 5.4) defines a rule for mapping a model from a specific model view to one another. Here, the model view belongs to a modelling method. For example, the class diagram in UML is one model view. However, in this context, view is also used if only one model view is defined for a modelling method and, therefore, stands for the whole modelling method.

For the rule in Listing 5.4, two parameters must be provided for the two model view names. The parameter `<vn>` stands for the view name of the source model, and `<vn'>` is the view name of the model that should be created through the mapping. The name of the source model will also be assigned to the target model.

```
1 IF
2   model.viewName == <vn>
3 THEN
4   newModel = createModelofView(model.name,<vn'>)
```

Listing 5.4: Model Mapping

The rule in Listing 5.5 defines how an object is mapped from one model to another. Here, the assumption is that this rule is called within the transformation of a model so that the context of the source and target model are given.

The rule is triggered if the object belongs to the class with the class name defined by the parameter `<cn>`. If the trigger is fulfilled, then a new object of the class `<cn'>` is created. The position and name of the source object are copied to the target object. Additionally, the other attributes are mapped based on the defined mapping of attributes.

```
1 IF
2   obj.className == <cn>
3 THEN
4   newObj = createObjOfClass(obj.name,<cn'>)
5   newObj.position = obj.position
6   copyAttributes(obj,newObj,attributeMapping)
```

Listing 5.5: Object Mapping

The rule of Listing 5.6 is similar to the mapping of objects (see Listing 5.5) but for the mapping of relations from one model to another. In addition to creating a new relation and taking the name from the source relation, the source and target from the source relation are transferred to the target relation. As the objects are already transformed, when this rule is triggered, there should already be objects with the identifier of the source model in the target model. Therefore, the source and target identifiers can be taken from the source model. For this rule, the user must provide the relation class in the source model (`<rn>`) and the relation class in the target model (`<rn'>`).

5.4. Design Thinking Digital Twin Capabilities for Using the Adaptive Modelling Environment in Workshops

```
1  IF
2    rel.relationClassName == <rn>
3  THEN
4    newRel = createRelationOfRelationClass(rel.name,<rn'>)
5    newRel.source = rel.source
6    newRel.target = rel.target
7    copyAttributes(rel,newRel,attributeMapping)
```

Listing 5.6: Relation Mapping

The following rule (Listing 5.7) shows how the mapping of an attribute can be defined. The *if* part checks if the object is an instance from the class or relation defined by *<cn>*. Additionally, the object in the source model must possess an attribute with the name *<an>*. Additionally, the source attribute *<an>* must be of the same type as the target attributes *<an'>*. This is indicated by line 8 in Listing 5.7. If an object that fulfils all the requirements is found, then the value of the provided attribute of the source object will be set as the value of the target object. What the target object is is provided by triggering these rules in the transformation procedure.

The user must provide a class name of the object in the source model (*<cn>*), an attribute name for the object in the source model (*<an>*) and the attribute name for the object in the target model (*<an'>*)

```
1  IF
2    (obj.className == <cn>
3  XOR
4    obj.relationClassName == <cn>)
5  AND
6    obj.hasAttribute(<an>)
7  AND
8    obj.<an>.attributeType == targetObj.<an'>.attributeType
9  THEN
10   targetObj.<an'> = obj.<an>
```

Listing 5.7: Attribute Mapping

The following transformation rule (see Listing 5.8) provides an example of how a relation from the source model can be transformed into an object in the target model. Not only will an object be created in the target model, but the new object will be related to the two objects, which were connected by the relation in the source model.

For the trigger condition of the rule, the user must provide three parameters: the class of the relation (*<relclass>*) and the classes of the two objects which are connected by the relation (*<class1>* and *<class2>*).

If the trigger condition is met, an object with the relation's name is created from a class defined by the user with the name *<cn'>*. Then, the two objects, which are related to the source model, are created. Here, an assumption is that the *createObjOfClass* function checks if objects with these names of the provided classes are already available before they are created. If so, then a reference to these objects is returned. Here, the user must provide the class names for the two objects (*<class1'>* and *<class1'>*). Last but not least, relations between the objects are created. Here the two classes for the relations

5. The Adaptive Modelling Environment

must be provided with `<relClass1'>` and `<relClass2'>`. The `_` should signal that for this relations no name will be provided. Also, the endpoint of the new relation is set.

```
1  IF
2    rel.relationClassName == <relclass>
3  AND
4    rel.source.className == <class1>
5  AND
6    rel.target.className == <class2>
7  THEN
8    newObj = createObjOfClass(rel.name,<cn'>)
9    copyAttributes(rel,newObj,attributeMapping)
10
11   newSrc = createObjOfClass(rel.source.name,<class1'>)
12   copyAttributes(rel.source,newSrc,attributeMapping)
13
14   newTar = createObjOfClass(rel.target.name,<class2'>)
15   copyAttributes(rel.target,newTar,attributeMapping)
16
17   newRel1 = createRelationOfRelationClass(\_,<relClass1'>)
18   newRel1.source = newObj
19   newRel1.target = newSrc
20   newRel2 = createRelationOfRelationClass(\_,<relClass2'>)
21   newRel2.source = newObj
22   newRel2.target = newTar
```

Listing 5.8: Relation to Object Mapping

The following transformation rule (see Listing 5.9) takes an object connected to two other objects and transforms it into a relation. Here, the endpoints of the target relation are set. In the trigger condition, it is checked if the three objects with the corresponding classes (`<cn>`, `<cn1>` and `<cn2>`) are available and if they are related with relations of the user-defined relation types (`<rn1>` and `<rn2>`). The *existsRelation* function checks if there is a relation between the objects, no matter in which direction.

If the condition is met, then the objects of the endpoints are created by using the user-defined classes (`<cn1'>` and `<cn2'>`). The relation is created with the information from the corresponding source object and the user-defined relation class `<rn'>`.

```
1  IF
2    obj.className == <cn>
3  AND
4    obj1.className == <cn1>
5  AND
6    obj2.className == <cn2>
7  AND
8    existsRelation(r1,obj,obj1) AND instanceOf(r1,<rn1>)
9  AND
10   existsRelation(r2,obj,obj2) AND instanceOf(r2,<rn2>)
11 THEN
12   obj1' = createObjOfClass(obj1.target.name,<cn1'>)
13   copyAttributes(obj1,obj1',attributeMapping)
14   obj2' = createObjOfClass(obj2.target.name,<cn2'>)
15   copyAttributes(obj2,obj2',attributeMapping)
```

```

16
17 rel' = createRelationOfRelationClass(obj.n,<rn'>)
18 rel'.source = obj1
19 rel'.target = obj2
20 copyAttributes(obj,rel',attributeMapping)

```

Listing 5.9: Object to Relation Mapping

The following two pseudo rules will provide examples of semantic richer transformation rules. They will use structural information like objects, relations, and attributes to define the mapping and make them attribute-dependent. Therefore, the user must provide a comparison statement based on an attribute, which can be evaluated as true or false. The comparison operators introduced in Table 5.2 can be used here.

One example for mapping a class based on an attribute value and one for mapping a relation based on an attribute value are provided.

For mapping an object based on an attribute value, the user must provide the class of the object (*<cn>*) and the compare statement, containing the attribute name (*<an>*), the compare operator (*<compare>*) and the value (*<val>*) against which the attribute should be compared. Here, adding comparisons between an attribute and a predefined value would be possible by adding the lines with an *AND* operator. This transformation rule is visualised in Listing 5.10.

In the execution of the rule, it is checked if a new attribute name (*<targetAttr>*) for the target object and a new attribute (*<targetVal>*) value for the target object were created. Depending on which of the operators is available, the attribute name and value for the target object are chosen in addition to copying the other information to the target object. Therefore, a class name (*<cn'>*) from the target model must be provided, from which the object should be instantiated.

```

1  IF
2  obj.className == <cn>
3  AND
4  obj.<an> <compare> <val>
5  THEN
6  IF
7  exists(<targetVal>)
8  AND
9  exists(<targetAttr>)
10 THEN
11 newObj = createObjFromClass(obj.name,<cn'>)
12 newObj.position = obj.position
13 newObj.<targetAttr> = <targetVal>
14 copyAttributes(obj,newObj,attributeMapping)
15 ELSEIF
16 not exists(<targetVal>)
17 AND
18 exists(<targetAttr>)
19 THEN
20 newObj = createObjFromClass(obj.name,<cn'>)
21 newObj.position = obj.position
22 newObj.<targetAttr> = obj.<an>

```

5. The Adaptive Modelling Environment

```
23   copyAttributes(obj,newObj,attributeMapping)
24   ELSEIF
25     exists(<targetVal>)
26     AND
27     not exists(<targetAttr>)
28     THEN
29     newObj = createObjFromClass(obj.name,<cn'>)
30     newObj.position = obj.position
31     newObj.<an> = <targetVal>
32     copyAttributes(obj,newObj,attributeMapping)
33   ELSE
34     newObj = createObjFromClass(obj.name,<cn'>)
35     newObj.position = obj.position
36     copyAttributes(obj,newObj,attributeMapping)
```

Listing 5.10: Conditional Object Mapping

The following rule (Listing 5.11 is an extension to the attribute mapping rule (Listing 5.7) by allowing the user to provide a new attribute name for the target object (<an'>) and a specific value for the target object (<val'>), which is set if the source attribute fulfils a specific condition provided by the user. Therefore, the user must provide a comparison operator (see Table 5.2) and a value against which the attribute of the source object should be compared.

The attribute and its value, which will be set for the target object, depend on information provided by the user. If a new attribute name (<an'>) and new value (<val'>) were provided, then they will be set at the target object. If one of the parameters is missing, then the missing parameter is taken from the source object. If neither a new attribute name nor a new value is provided, this rule will do nothing, as it would have the same result as the standard attribute coping. The attribute name from the source object is used if it is available in the target object. Also, the attribute value from the source object will be used if available.

```
1   IF
2     (obj.className == <cn>
3   XOR
4     obj.relationClassName == <cn>)
5   AND
6     obj.hasAttribute(<an>)
7   AND
8     obj.<an> <compare> <val>
9   THEN
10    If
11      exists(<val'>)
12    AND
13      exists(<an'>)
14    THEN
15      targetObj.<an'> = <val>'
16    ELSEIF
17      not exists(<val'>)
18    AND
19      exists(<an'>)
20    THEN
```

```

21 targetObj.<an'> = obj.<val>
22 ELSEIF
23 exists(<val'>)
24 AND
25 not exists(<an'>)
26 THEN
27 targetObj.<an> = <val>
28 ELSE
29 targetObj.<an> = obj.<an>

```

Listing 5.11: Conditional Attribute Mapping

A pseudo rule for the conditional mapping of relation is introduced in Listing 5.12. It is similar to the mapping of an object introduced in Listing 5.10. The user must provide a relation class name (*<rn>*) for the relation which should be transformed and the comparison statement with the attribute name (*<an>*) of the source attribute, the compare operator (*<compare>*) and a value (*<val>*), to define the trigger of the rule. The compare operators are defined in Table 5.2.

For the creation of the relation, different variants are available based on whether a target attribute name (*<targetAttr>*) or target value (*<targetVal>*) is provided by the user. The possibilities differentiate in which attribute name and which value is used.

```

1  IF
2    rel.relationName == <rn>
3  AND
4    rel.<an> <compare> <val>
5  THEN
6    IF
7      exists(<targetVal>)
8    AND
9      exists(<targetAttr>)
10   THEN
11     newRel = createRelationOfRelationClass(rel.name,<rn'>)
12     newRel.target = rel.target
13     newRel.source = rel.source
14     newRel.<targetAttr> = <targetVal>
15     copyAttributes(rel,newRel,attributeMapping)
16   ELSEIF
17     not exists(<targetVal>)
18   AND
19     exists(<targetAttr>)
20   THEN
21     newRel = createRelationOfRelationClass(rel.name,<rn'>)
22     newRel.target = rel.target
23     newRel.source = rel.source
24     newRel.<targetAttr> = rel.<an>
25     copyAttributes(rel,newRel,attributeMapping)
26   ELSEIF
27     exists(<targetVal>)
28   AND
29     not exists(<targetAttr>)
30   THEN
31     newRel = createRelationOfRelationClass(rel.name,<rn'>)

```

5. The Adaptive Modelling Environment

```
32     newRel.target = rel.target
33     newRel.source = rel.source
34     newRel.<an> = <targetVal>
35     copyAttributes(rel,newRel,attributeMapping)
36     ELSE
37     newRel = createRelationOfRelationClass(rel.name,<rn'>)
38     newRel.target = rel.target
39     newRel.source = rel.source
40     newRel.<an> = rel.<an>
41     copyAttributes(rel,newRel,attributeMapping)
```

Listing 5.12: Conditional Relation Mapping

5.5. Conceptual Architecture of the *Adaptive Modelling Environment*

This section will discuss the conceptual architecture and its components of the *Adaptive Modelling Environment*, needed to support the defined capabilities, use it in workshops and apply the digital twin from design thinking. The architecture discussed here will not be focused on a specific installation but on a conceptual level. This is done to focus on the essential aspects without getting lost in implementation-dependent details. The different components were defined based on the goal and the identified capabilities. The conceptual architecture is visualised in Figure 5.21.

Before the introduction, the difference between the terms modelling tool and modelling environment should be emphasised again. The component that manages the digital models and is the foundation for the digital twin from design thinking will be called a modelling tool. This includes creating and adapting models, metamodel adaptations, providing a user interface and so on. The modelling environment contains a modelling tool and additional components that enhance its capabilities, enabling functionality that does not belong to a modelling tool's core. In the context of this thesis, these are components that enable the transformation from the physical to the digital models or support the further utilisation of the created models.

The conceptual architecture does not focus on which components are implemented in one application or different ones but on the different conceptual components needed in the proposed modelling environment. Not each instance of the modelling environment must always contain all the components. However, the modelling environment must provide the capability to exchange different instances of the components. This is needed to allow the adaptability of the modelling environment and the extension and tailoring of the available functionality.

To make the conceptual architecture shown in Figure 5.21 usable, concrete algorithms and their implementation must replace the conceptual component. For example, object detection can be achieved through object detection with convolutional neural network (CNN), and relation identification can be realised by using the proximity of the used modelling objects.

5.5. Conceptual Architecture of the Adaptive Modelling Environment

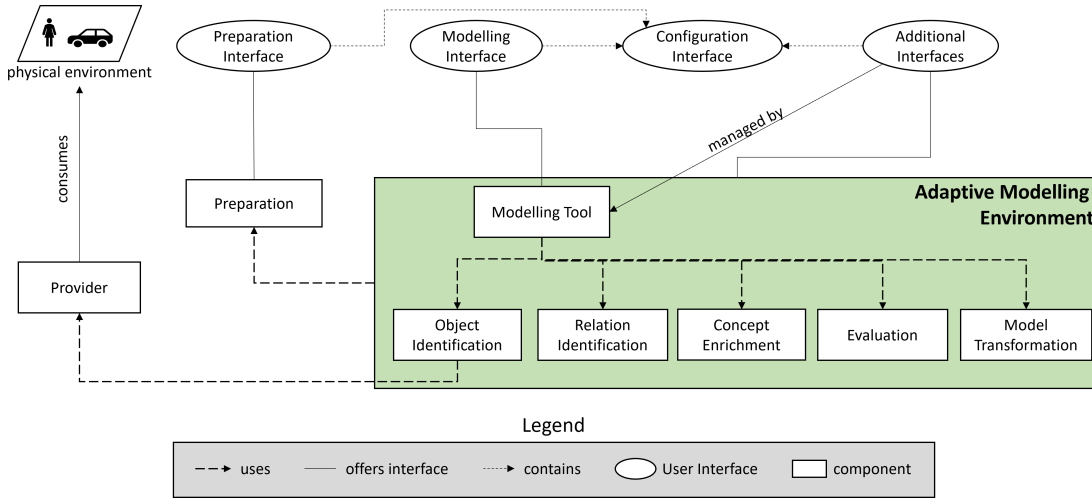


Figure 5.21.: Conceptual architecture for the *Adaptive Modelling Environment*

The *Adaptive Modelling Environment* shown in Figure 5.21 contains the components for the modelling environment, which possess links to external components. These are the *Preparation* (discussed in section 5.5.9) and the *Provider* component (discussed in section 5.5.3). Through the provider component, the modelling environment can gather information from the physical environment. The interfaces are offered by the different components, e.g., the modelling tool, but are shown to and interacted with by the users. Therefore, they are shown outside the green box. More information about the interfaces can be found in section 5.5.2.

Figure 5.21 shows that the modelling tool can communicate with the components within the modelling environment. This is visualised because one of the main goals is that information can be provided to or used from the modelling tool in the other components. Based on the implementation, the different components can also interact. These relations are not shown, as drawing the relation from each component to each other would visually clutter the figure.

The conceptual architecture does not provide information about the deployment of the components, meaning that the modelling environment must not run on a singular device but can be separated into multiple ones. Further, one modelling tool must not work with one specific instance of the modelling environment, as the user of the modelling tool may participate in multiple workshops, which can have their installation of the components.

Before the individual components are discussed in detail, an example of the conceptual architecture based on the capabilities for the modelling environment (as introduced in section 5.4) is introduced. This example is then used to support the discussion and explanation of the different components.

For this example, we assume that a physical, paper figure-based storyboarding (see section 2.1.2.3 for more information) design thinking technique is used in a workshop. The physical models should be automatically transformed into digital models using CNN

5. The Adaptive Modelling Environment

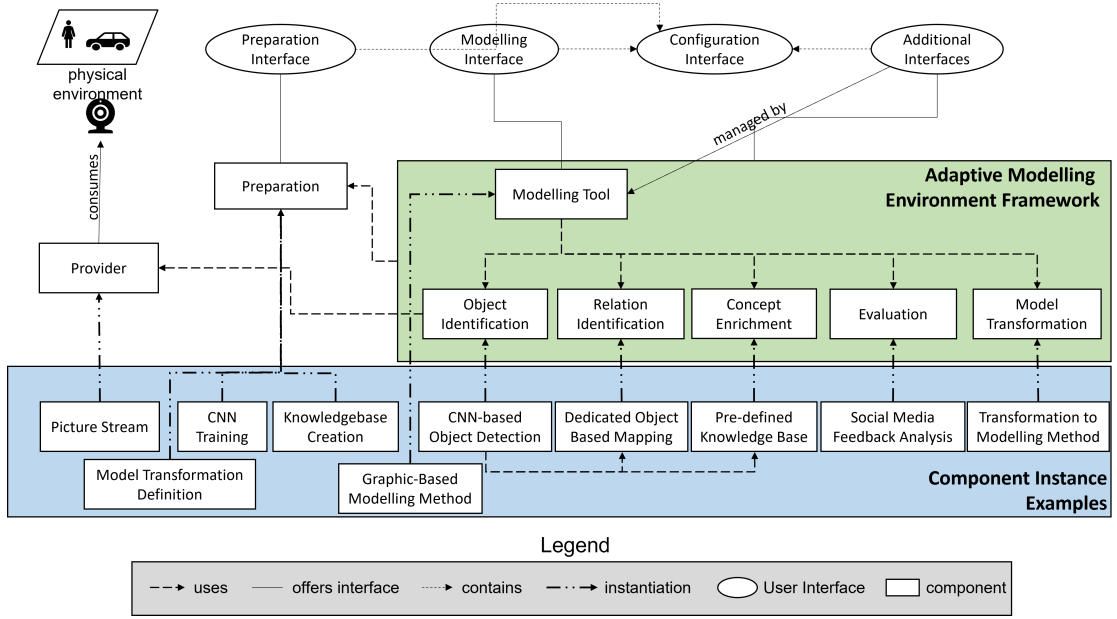


Figure 5.22.: Conceptual architecture for the *Adaptive Modelling Environment*, enhanced with instances for the components

based object detection. Specified objects should be transformed into relations within the digital model. Here, a workshop organiser has provided a starting point in advance. For more information on such a transformation, see section 5.4.1.1. After the digital model is created, the users should be able to gather feedback from involved stakeholders by posting it to social media and analysing the feedback (see section 5.4.2). Last but not least, the model from the workshop should be transformed into a model from another modelling method, as discussed in section 5.4.3.

This example is visualised in Figure 5.22 and additional information will be provided in the description of the individual components. The figures also show that the modelling tool must be instantiated to have a fitting modelling method for the example. It must be able to digitally represent the physical storyboards. Additionally, the example does not offer the functionality of the components *dedicated object mapping* and *pre-defined knowledge base* directly, but only over the *CNN-based object detection* component. This should show that the components can also interact with each other, depending on the concrete installation.

5.5.1. Modelling Tool

The modelling tool is the core component of the modelling environment. It manages the digital model and, subsequently, the design thinking digital twin. Further, it allows users to interact with the digital twin and trigger its functionalities. To use a modelling tool in a particular context, a fitting modelling method must be provided within it, which

5.5. Conceptual Architecture of the Adaptive Modelling Environment

is achieved by creating the metamodel and additional artefacts like a definition of the needed mechanisms.

In the example environment (see Figure 5.22), the modelling tool must possess a modelling method that allows the creation of graphical objects for visualising a storyboard. The needed metamodel should be based on the metamodel introduced in section 5.2.3.

To allow the users a direct interaction with the models the modelling tool must provide a graphical user interface (GUI) as *modelling interface*. This must support standard functionality like creating or changing models, adding information to the objects in the models, and visualising the model so that users can see it. Additionally, the modelling tool must be able to trigger the transformation from the physical to the digital model when the users are finished with the physical model. As the modelling tool is the core of the modelling environment, the user's primary interaction should be with the tool, which then communicates with the other components. Depending on the concrete implementation, different components may have interfaces with which the users must interact. Such *additional interfaces* may be websites or other form of GUIs. In such cases, the modelling tool should offer a way to open these interfaces to reduce the user's burden of knowing all the interfaces. For example, the modelling environment can trigger the opening of the webpage of one component.

The modelling tool itself must interact with the other components to either gather information that should be shown in a model or provide information out of models that are needed in other components. However, as the components must be configured to work with the current workshop, the modelling tool must also be configurable to work with different instances of the components for different workshops.

The modelling tool itself must also be able to dynamically adapt its metamodel so that the users can tailor the modelling method to their needs. Such adaptation must also be done through the modelling tool's mechanisms so that the users can be guided. In the example introduced in Figure 5.22, the classes representing the physical figures must be adaptable to fit the used physical figures.

5.5.2. User Interface

This component is a collection of the different interfaces needed in the modelling environment, which depends on the concrete implementation. However, every modelling environment needs the *modelling interface*, the main GUI used to interact with the modelling environment. As discussed in section 5.5.1 the modelling interface should be able to forward the users to *additional interfaces* they might need.

The modelling and the additional interfaces need to be configured to fit a concrete instantiation of the modelling environment. Different places for the workshop may provide different components for recognising and transforming the physical objects. The *configuration interface* should then be used to enter the parameters of the concrete installation of the components so that they can be used.

The *preparation* component also needs an interface which supports the user in creating the needed input for the modelling environment. An interface is needed for each preparation component instance, which may include a configuration. However, this depends on

5. The Adaptive Modelling Environment

the concrete instance.

5.5.3. Provider Component

This component focuses on preprocessing data to be input for the modelling environment. It also functions as the interface between the physical world and the digital world by using sensors. For this thesis, the provider component will focus on digitising physical models using a camera. Theoretically, creating a provider for other physical aspects like sound and digitised spoken words from the user would also be possible. However, this would go beyond the scope of this thesis. Therefore, the focus is on taking pictures and providing them to the modelling environment.

This functionality is encapsulated in its own component, as the device for taking the image may not have the processing capabilities to provide the tasks of object or relation identification components. Additionally, the physical restraints of the workshop may call for the separation of taking a picture and processing it. Too much hardware can hinder the space in the physical workshop.

The provider needs a hardware device connected to a camera, e.g., a smartphone or a laptop with a webcam. Then, a picture is provided of when the physical model should be transformed. This picture is input for the *object identification* component, which contains the first processing step.

For the example visualised in section 5.22, the provider is used to take pictures of the physical objects of the design thinking technique and forward them to the modelling environment.

5.5.4. Object Identification

The *object identification* component consumes input from the provider component and processes it so that the modelling environment can further use it. The main objective is that individual objects used in the physical model are identified and encoded so that the modelling environment can further process them. These objects are primarily aimed to be used in the modelling tool but should also be useable in other components.

Therefore, the object identification must be linked to a provider component, which offers the input data in a consumable format, like a picture. To achieve this link, the object identification and provider components must coordinate with each other through their configuration. This requires establishing a data connection between the components, for example, over a network or the internet.

How the object identification is accomplished depends on the concrete scenario and what is provided by the provider component. In the example visualised in Figure 5.22, the object identification must recognise the used objects and their position in a picture using convolutional neural networks. A trained network must be available to allow this kind of object identification (see section 5.4.1.1.4 for more information). The CNN will be trained and prepared in the *preparation* component, which will be discussed in section 5.5.9.

After the objects and their position are detected, they must be offered in a data format which can be used internally. It must contain the needed information to be mapped to

5.5. Conceptual Architecture of the Adaptive Modelling Environment

the metamodel of the modelling tool. This at least contains the position in the form of x and y coordinates and an identifier. This data must not only be provided to the modelling tool but also to the *relation identification* component (see section 5.5.6) and the *concept enrichment* component (see section 5.5.5).

As the object identification component provides the data structure later used by other components, it is the main component for transforming from physical to digital models within a workshop.

5.5.5. Relation Identification Component

The *relation identification* component is used to derive relations between concepts in models, independent of whether they are physical or digital models. Therefore, the objects and their position are interpreted, and based on it, possible relations will be derived, which will be made available for the modelling tool or other components if needed. A way to identify relations was introduced in section 5.4.1.1.

To ease the interoperability between the components, a basic data structure in the *object identification* should be used, which must be enhanced to define which concept relations should be established. Therefore, the type, source, and target must be defined for the relation. The input can either be directly from the modelling tool or from the object detection component.

This component needs configuration where the additional information, which is needed for the identification of relations, must be provided. Based on the two possibilities introduced in section 5.4.1.1, this can be the specification of the concepts representing relations or that additional information about the relation between two concepts is provided, like a type or a label. The last part belongs to the *concept enrichment* component, introduced in section 5.5.5.

5.5.6. Concept Enrichment Component

The *concept enrichment* component aims to provide additional information to the identified concepts and relations, improving users' understanding. Therefore, this component will be called after the *object identification* and the *relation identification* components have finished their task.

The enrichment must be based on available data, as enhanced information cannot come from thin air. The component can either use information that was created for the workshop with the *Preparation* component (see section 5.5.9) or the use of generative AI or open data sources to retrieve additional information. For the latter, it must be ensured that the retrieved information fits the workshop domain, which can be difficult.

The enrichment can work on the meta or the instance layer. The attributes are then assigned to the class and can be filled out by workshop participants. Alternatively, they can contain the attribute for the class and a value for the instance. For relations, they can provide the same information, and additionally, they can provide a type for the relation based on the connected objects.

5. The Adaptive Modelling Environment

5.5.7. Evaluation Component

The *evaluation* component supports users in gathering feedback from involved stakeholders and analysing them. Therefore, the component must be linked to a platform where the information on the models can be provided, and involved stakeholders can provide feedback. For example, the participants and external stakeholders already use social media platforms. This component is used after a digital model is created and probably enriched by the user. Such functionality is described in section 5.4.2.

The component should enable a triggering through the modelling tool, which allows sending a picture with questions or statements from the modelling tool to gather feedback. Therefore, this exchange of information must be enabled between the modelling tool and the evaluation component. The evaluation component should support posting to a social media platform and retrieving the information provided on the platform. This information must then be processed to be visualised in the model. The modelling tool must also be able to trigger information gathering. The mapping from the post to gathering the feedback must be enabled between the modelling tool and the evaluation component.

Here a prerequisite is that the evaluation component is connected to the API of a social media platform. This must be configured, and the evaluation component must provide a configuration interface.

5.5.8. Model Transformation

The *model transformation* component is used after the models are created and the first version is finalised. The component aims to support enhancing the knowledge created in the workshop. This should be achieved by using additional modelling methods to describe further aspects of the design created in the workshop.

The model transformation component supports the users by allowing them to transform the models from the workshop to a starting point in the other modelling methods. This approach is discussed in section 5.4.3.

The transformation is based on rules which users must define. The more rules are defined, the better the starting point for the additionally chosen modelling method will be. A modelling tool must be available for this modelling method, which allows an import of the created starting point.

5.5.9. Preparation Component

The *Preparation* component is a collection of applications and services used to prepare the workshop and the later usage of the digital model from the workshop. As discussed in the proposed usage guidelines in section 5.1.3.2, a workshop and the used modelling environment should be prepared. A set of applications is needed to support the user. Which applications are needed directly depends on the concrete components used.

Three tools are needed to prepare a workshop based on the example visualised in Figure 5.22. This is the training of the CNN for the object identification, the additional information for the objects and relations, and the rules for defining the model transform-

5.5. Conceptual Architecture of the Adaptive Modelling Environment

ation. However, this does not mean that an individual preparation application must be created for each part. For those that are related, a joined one can be created. For example, the CNN training and the knowledge base for the additional information can be combined, as they are based on the same concepts.

The results of the preparation component are files in a format that the other components can use. In this way, the Preparation can be used during the workshop. Additionally, the files can also be reused for other workshops.

6. Evaluation with the Scene2Model Prototype

This chapter discusses the evaluation of the concepts introduced in this thesis, based on the Scene2Model (s2m) proof-of-concept prototype. Scene2Model implements the core functionality introduced in chapter 5, for SAP ScenesTM (cf. [7]) as the one chosen haptic design thinking technique. This technique is further discussed in section 3.3 and section 2.1.2.3. Therefore, the haptic modelling is based on paper figures with semantic-rich visualisation placed on a table, and the *Adaptive Modelling Environment* is used to create design-thinking digital twins based on conceptual models.

This design thinking technique was chosen as its semantically rich visualisation aligns well with the goal of diagrammatic modelling, where visualisations are used to make the captured knowledge easier to understand. Additionally, created storyboards are usually on a high abstraction level, facilitating the co-creation between humans without needing a cumbersome learning process of the technique. In this way, the prototype can focus on applying and evaluating the design thinking digital twin and not explaining the design thinking technique to the users before the workshop. Last but not least, as the paper figures must be adapted to the domain of the storyboard, not only the ones from SAP ScenesTM were used, but other figures were added.

The evaluation is based on two parts: once implementing the core of the *Adaptive Modelling Environment* in a proof-of-concept, evaluating and reflecting how it can be implemented and secondly, using the prototype in workshops, where the participants were observed and afterwards discussions were held. For two workshops, which were held during two instances of the Nemo¹ Summer School, the evaluation was additionally supported with questionnaires. The results of these questionnaires are the focus of the second part of the evaluation. Observing and testing the prototype in workshops without questionnaires were used to improve the tool's implementation. For example, we organised workshops with students observing and discussing their impressions of the tool.

The created Scene2Model (cf. [194, 195, 5]) prototype implements the aspects from the *Adaptive Modelling Environment* introduced in chapter 5 which were needed for the workshops and was used as the bases for proof-of-concept implementation for analysing the needed functionality. For the proof-of-concept implementation of this thesis, we used an AI based approach for digitalising the physical figures and enriching them (as introduced in section 5.4.1.1). For the evaluation of the questionnaire, we used a tag-based approach (as introduced in section 5.4.1.2). The tag-based approach was used, as the evaluation focused on using the physical and digital components together. Therefore, we did not

¹<https://nemo.omilab.org/> (accessed: 28-5-2024)

6. Evaluation with the Scene2Model Prototype

shift the participants' focus to the machine learning approaches, which require time to understand how to use them, but instead used the easier-to-use approach and focused on the usage.

Below, first, additional technical aspects of the prototype are introduced, and afterwards, the questionnaire and the results are discussed.

6.1. The Scene2Model Proof-of-Concept Prototype

This section will introduce technical aspects which were used to implement the *Adaptive Modelling Environment* (introduced in chapter 5) for a haptic storyboarding approach into the Scene2Model prototype. Information on how the prototype can be used in a workshop environment will also be provided.

The Scene2Model prototype consists of a modelling environment to support tangible and digital modelling for storyboarding. It contains a modelling tool for managing conceptual models, which are the foundation for designing a digital twin. It also contains the environment to photograph the tangible design artefacts and analyse them to create the digital model automatically. Based on the digital twin from design thinking, the models in the modelling tool are then used as bases for functionality.

First, we introduce the technologies used to implement the Scene2Model prototype. The concepts behind the implementation and how it works are described in chapter 5, and here, only the technologies are introduced. Afterwards, the usage of the prototype and its implementation will be discussed.

For managing the code of the proof of concept prototype, the OMiLAB GitLab was used: <https://code.omilab.org/muckc/prototype-dissertation>. Here, one can request access to the code. The repository contains the code which can be reliably started and used.

6.1.1. Used Technologies for Implementing the Scene2Model Prototype

For creating the prototype, the components introduced in the architecture (see section 5.5) were implemented. The different components were implemented in individual applications, only connecting them through REST based interfaces or by exporting and importing files.

Each functionality not directly used in the modelling tool was implemented in its own service and is deployed independently from the modelling tool. This increases the flexibility of the environment, as the components can be deployed for different workshops, and the modelling tool can be connected to different scenarios by using configuration capabilities. For example, the pictures can be taken with different implementations, or the recognition can be changed between tag-based and AI-based with the same modelling tool. This means that the different functionalities are offered as their services, which can be combined in different ways to fit specific scenarios.

This also means the implementation is not limited to a specific programming language or framework. The best fitting can be used if it supports the creation of REST interfaces.

6.1. The Scene2Model Proof-of-Concept Prototype

The modelling tool is the heart of the environment, where all the information comes together or can be provided as input for other components. The tool was implemented using the ADOxx metamodeling platform (see appendix A.4.1 for more information). The platform offers to make HTTP and process the results. This functionality was used to connect it to the other components.

Different prototypes were implemented for the image provider component. Here, a camera was opened within some application, and the taken picture was offered to the recognition component. As USB camera often the *Logitech HD Pro Webcam C920*² were used. For example, the *Python opencv* library (see appendix A.4.11) or *ffmpeg* were used.

When the object identification component is triggered, it fetches the image from the provider component and identifies the used paper figures. Two approaches were implemented for this identification. One is based on predefined tags, as described in section 5.4.1.2, and one is AI based and uses CNN, as described in section 5.4.1.1.

For the tag-based recognition, we used the *ArUco* (see appendix A.4.11) library, which is also integrated into Python *opencv*. This allows us to create tags that can be attached to the paper figures after printing them. Then, the offered recognition functionality is used to identify the tags in the picture.

For the CNN based object identification prototype, we used the *Yolov4-tiny* (see appendix A.4.12) architecture, trained with Darknet (see appendix A.4.4) application. The trained net was then used in a Python server, using the *Python opencv* (see appendix A.4.11) library. This library offers functionality to open the files created after the training of the CNN. The recognition can then be applied to a picture loaded in the Python code.

The relation identification was implemented in the same service as the CNN based object identification to better use its results. For identifying the text written on the identified objects, we used the *EasyOCR* (see appendix A.4.5) library. An algorithm was written to identify the relations.

For testing the RDF based ontology functionality for deriving additional information, we used *GraphDB* (see appendix A.4.7). To avoid putting additional strain on the users, the prototype works without the GraphDB connection, as the database must be set up and maintained separately.

The evaluation component was implemented in the *Olive* microservice framework (see appendix A.4.10) to mention the connection to the X/Twitter platform. Additionally, it uses the *CoreNLP* (see appendix A.4.2) library implemented by Stanford NLP group to apply sentiment analysis on the comments made to the posts.

Two prototypes were implemented for the model transformation. For one, we used *lxml* Python library. We defined the rules for transforming XML, which can be exported and imported out of the box with ADOxx-based tools to exchange information between different modelling methods. For the second, we used the ADOxx (see appendix A.4.1) build-in *AdoScript* functionality to write an export and import of JSON files, which can be shared between specific ADOxx-based modelling tools.

²<https://www.logitech.com/de-at/products/webcams/c920-pro-hd-webcam.960-001055.html> (accessed: 01.06.2024)

6. Evaluation with the Scene2Model Prototype

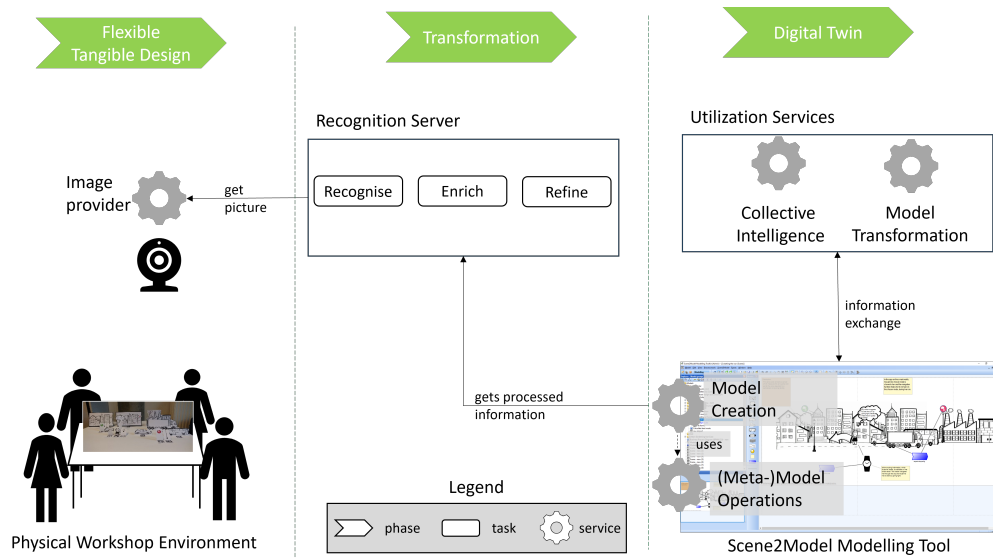


Figure 6.1.: Visual Representation of usage phases of the prototype (the figure contains pictures from SAP ScenesTM [7])

6.1.2. Usage Phases of the Scene2Model Prototype

The usage of the Scene2Model prototype during a workshop can be separated into three phases, which will be discussed in this section and are visualised in Figure 6.1. The phases discussed here focus primarily on the usage in the workshop and processing of the created models, which are the design steps of the usage guideline introduced in section 5.1.3.2.

A starting set of figures and their visual representation must be established before the environment can be used, including the tangible and digital aspects. These figures must then be prepared to be used as tangible paper figures and as classes in the modelling tool. Only a starting set of possible figures should be prepared to ease the working. Later, needed figures can be added on the fly during usage.

Here, the domain the figures should represent must be considered, and the model usage after the workshop should be kept in mind. For example, other modelling methods used later to implement the idea should be considered. For example, value networks could be used after the workshop to analyse the idea. Such considerations could influence available figures or relations prepared for the modelling environment, including the paper figures.

For the introduction of the prototype, we assume that we want to use Value Networks, as implemented in the EcoViz (cf. [188]) after the workshops, and, therefore, a model transformation prototype from the Scene2Model models to EcoViz models is also considered. Here, the tangible and digital models enabled a way to define the different value streams between actors.

The usage starts with the flexible, tangible design phase, which focuses on using paper figures in a workshop. Flexibility in this context means that the paper figures can be placed in any order, any semantics can be assigned to them based on the discussion, and

they can be enhanced by adding new printed or hand-drawn pictures or even handwritten nodes to create the physical model. Afterwards, a picture of the figures is taken and analysed in the transformation phase.

The transformation phase gets the picture of the figures as input. It processes them so that the information encoded in the pictures can be captured in a digital twin from design thinking. For this, the objects must first be identified and afterwards enriched by deriving additional information or using the information defined in the preparation for the workshop. Last but not least, the information must be refined so that it can be offered in a format understandable by the modelling tool. More information on the concrete process and prototype implementation is discussed in section 6.1.2.2.

The last phase is called digital twin, which takes the formatted input from the transformation phase and creates the model. The automated creation of the models uses operations offered by the modelling environment. The digital twin structures the knowledge in models corresponding to the metamodel. This information can then be processed to gather additional information or transform the knowledge into other models to be further used.

6.1.2.1. **Tangible Design**

The prepared and flexibly added paper figures are used in a physical workshop for the tangible design. Therefore, the physical environment must be prepared. This includes a table with enough space to put up the paper figures. Here, a space of about 80 cm times 80 cm should be sufficient. Additionally, the table should allow the participant to quickly set up and rearrange the paper figures to represent their case.

To support the participants, the available paper figures should be positioned on a separate table nearby so the workshop participants can get an overview of the available figures and easily pick the ones needed for their scenario.

Apart from the paper figures and physical location, the image provider component and the camera must also be set up to use the automated transformation and create the design thinking digital twin.

The camera must be adjusted to the used recognition component. The camera must be placed above the paper figure for tag recognition to capture the tags. However, suppose an AI based solution is used. In that case, the camera must take a picture of the visual representation of the paper figures and, therefore, capture it from the front.

By setting up the camera, one must consider not to put it in the way of the workshop participants and hinder their work. If a fixed-mounted camera is used, it should not hinder the participants from viewing the paper figures.

It is essential to consider that the picture is offered in a way so that it can be received from the recognition server. Moreover, the quality must be good enough to make the processing possible.

6. Evaluation with the Scene2Model Prototype

6.1.2.2. Transforming Haptic Design Artefacts Into Digital Twins of Design Thinking

The second phase focuses on transforming from the tangible design artefact to the digital twin of design thinking to reduce the additional workload for the users. The objective is to have the model instantly available during the workshop for use within it. Either by adding information, executing functionality on the model or evaluating the story at the end of the workshop.

This section focuses on how the transformation was implemented with AI-based object detection. This is done because this version enables us to derive additional information from the tangible models. The design of these approaches is discussed in section 5.4.1.1, and here we will talk about the implementation. This section will not discuss code or implemented algorithms. They are only instantiations and adaptations of the designs in section 5. Adaptations were sometimes necessary based on the used technologies, but they were done to have the same behaviour as wanted in the design.

First, the preparation of the AI-based approach for the recognition is described, and afterwards, the recognition itself is introduced.

6.1.2.2.1. Preparing for the Object Identification for a Workshop Before object identification based on convolutional neural network (CNN) can be used, it must be prepared by training it. First, the training data must be created. Therefore, pictures of the cutout paper figures must be taken, and then they must be labelled. During the labelling, the paper figures in the pictures must be marked by drawing a bounding box around them, and a label must be provided. The training data then consists of the pictures and, for each picture, the objects which can be found in it based on the label and the coordinates of the bounding box. For our prototype, we used the *DarkMark* (see appendix A.4.3 for more details) tool, which allows us to provide the names of the labels and then offers a GUI where all the pictures can be marked.

To reduce the time needed to create the training set, we also used data augmentation, which can be applied to an arbitrary set of pictures. For the prototype, a Python script was created which applies image augmentation based on the open-source *imgaug* library (see appendix A.4.8 for more information). The augmentation takes pictures and visually transforms them to inflate the training data artificially. This improves the overall result of the trained CNN. The bounding boxes applied to the original pictures must be transformed as well. Otherwise, the augmented pictures must be manually labelled, which would increase the users' workload.

Examples of image augmentation can be seen in Figure 6.2. Here, algorithms were applied that shift the perspective or zoom of the picture, blur it or cut parts of random parts of the picture.

For training itself, we used the open-source *Darknet* (see appendix A.4.4 for more information) tool with the *Yolov4-tiny* convolutional neural network (CNN) architecture (see appendix A.4.12 for more information). To execute training in a reasonable time, hardware with a potent GPU is needed. This, together with the training configuration, influences the training time and the results. Here, one crucial configuration parameter that directly influences the time and results and depends on the GPU is the number of

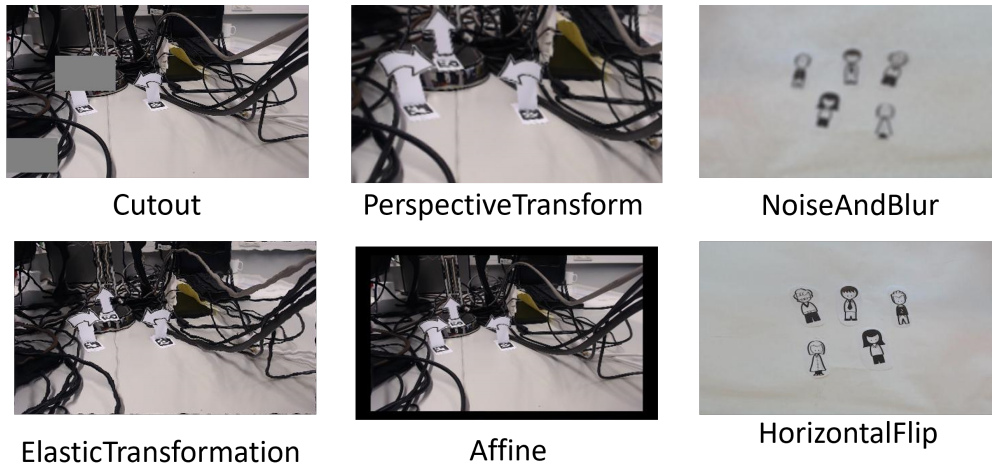


Figure 6.2.: Example of automated image augmentation with the *imgaug* library (the figure contains pictures from SAP ScenesTM [7])

pictures trained as one batch, meaning how many pictures are used before the weights in the neural network are updated. This can heavily influence the training time of an unchanged training set. For example, this change could change the time between one hour and eight hours for our training data set. The longer trained net is better abstracted and better recognises the figures from different angles, in different lighting conditions or with different backgrounds.

After the training, three files are needed to use the CNN: the file with the labels, the file with the configuration of the neural network and the encoded CNN itself. These three files can then be used in the recognition server.

It cannot be precisely specified how many pictures are needed to get fitting results, as it depends on the concrete set of used figures. If the figures are different from each other, only a few of them are needed. For example, for the person figures shown in Figure 6.2, three pictures of the individual figure were used and some pictures with the different figures together. However, more pictures must be taken for pictures that look similar, like the same arrow, but in a different direction. If figures should be recognised with text, then the training set should also contain pictures with text written on them. Lastly, empty pictures (without any figures) should also be added.

To improve the recognition of the figures, the same camera and a similar background should be used in the training. For example, it helps if a white background is used during the training and the recognition. Pictures of the figures with different backgrounds should be used in the training set to improve the trained network.

To allow the identification of relation between the figures, based on the pre-defined objects (as introduced in section 5.4.1.1) also, those objects, which should be interpreted as relations, must be defined. Therefore, the identifier of the figure must be defined and in which direction the relation should be interpreted. This is not part of the training and will be based on the labels used for the training. This information will be provided when

6. Evaluation with the Scene2Model Prototype

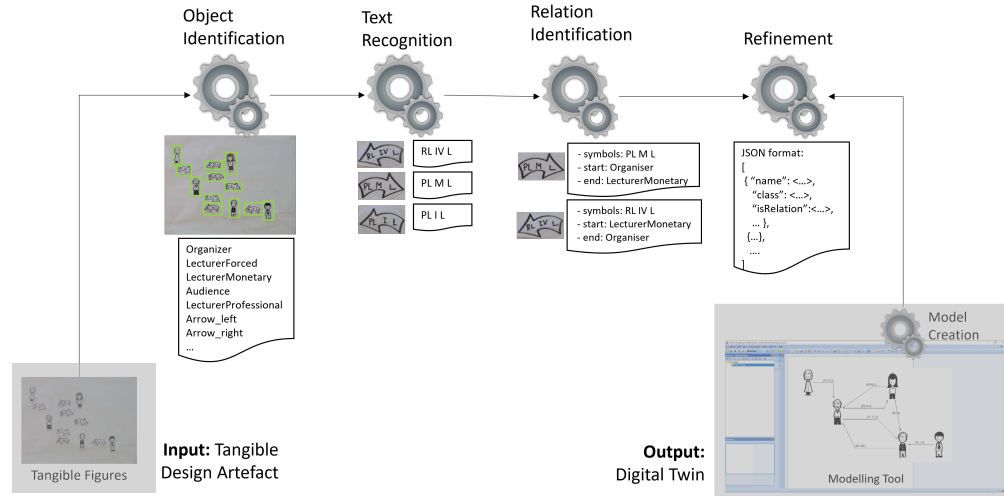


Figure 6.3.: Visualisation of the recognition workflow using pre-defined objects for relation identification (the figure contains pictures from SAP ScenesTM [7])

the recognition server software is configured.

Suppose a later transformation of the digital design thinking models to other modelling methods should be used. In that case, the rules for the transformation must also be defined, including the concepts from both modelling methods.

6.1.2.2.2. Recognition Workflow In this section, the recognition will be described, containing identifying the objects and proposing relations between them using pre-defined objects. The overall procedure is visualised in Figure 6.3.

Before the recognition component can be used, it must be configured. This contains general parameters of the server, like the IP address and the ports, but also parameters which influence the usage and functionality of the server. Having the configuration should allow the recognition to be adapted to the current needs. For example, the object's relation identification and text recognition can be turned off or on. Additionally, the trained networks for recognising the figures can be uploaded. Last but not least, the confidence threshold, which defines at which threshold an identified figure is considered or dismissed, can also be adapted. Last but not least, the modelling tool must be configured to reach the endpoint of the recognition component, which offers the recognised data.

Without text recognition and relation identification, the recognition component only returns the label of the identified paper figure and its position. For the description below, it is assumed that text recognition and relation identification are used.

The recognition procedure, visualised in Figure 6.3, is started by triggering the process and getting the picture of the image provider. Then, the uploaded trained convolutional neural network is used to identify the used paper figures.

A label and the coordinates of the bounding boxes in the pictures are then available for

6.1. The Scene2Model Proof-of-Concept Prototype

recognised paper figures. For each figure, one label and one bounding box are determined. If one figure is recognised by multiple CNN, then the recognition components decide which to keep using the decision process introduced in section 5.4.1.1.3. The bounding box is a rectangular frame drawn around the recognised figure, which is used to cut the picture of the recognised paper figures and have them available as individual pictures. These cutout pictures are used for two purposes. Firstly, they are provided as part of the result so that the picture can be used in the Scene2Model modelling tool. For example, it can be visualised if the recognised label is not yet defined in the metamodel. Secondly, each cutout picture is individually used as the basis for text recognition. In this way, the recognised text can directly be mapped to the figure on which it occurred.

Additionally, the coordinates of the bounding boxes are used to identify possible relations based on the position of the figures. Some figures can be defined to represent relation if placed in the middle between other figures, like the arrows in Figure 6.3. More information on how the concepts representing relations can be defined is discussed in section 5.4.1.1.3.

In the refinement step, the gathered information is processed and put in a format that can be easily transmitted between the recognition component and the modelling tool. A JSON format was chosen, containing the recognised label, the x and y coordinates, the picture in *base64* format and the recognised text. A flag indicates whether the identified object should be interpreted as a relation. If it is a relation, then the labels of the start and endpoint are provided so that the relation can be drawn in the modelling tool. The created JSON string is then made available over a REST endpoint. This endpoint must then be configured in the modelling tool to retrieve the information and create the model.

6.1.2.3. Using the Digital Twin of Design Thinking

The transformation results in the design thinking digital twin, which represents the haptic model digitally in a model. The individually used paper figures are represented as their objects and mapped to the domain metamodel available in the modelling tool. Therefore, it is not a mere representation, and the transformation algorithm enhances the information in the digital twin.

The automatically created model can be manually enhanced with information through the user and used to automatically process it during and after the workshops.

In Scene2Model, two prototypes for processing the modelled information were implemented. One sent a picture of the model with a corresponding question or statement to X (former Twitter) to collect further input. This is based on the description in section 5.4.2. The second was that the resulting models were transformed to create models in another modelling language. This functionality is based on the description in section 5.4.3.

For gathering information over social media, the metamodel was enhanced through a supportive concept called *Social Media Evaluation*, allowing us to define the question that should be asked and then save the results from the social media platform and visualise it. Here, X/Twitter was used, as at the time of the creation, the API was free and easy to use.

6. Evaluation with the Scene2Model Prototype

This functionality could make a post retrieve the comments and apply sentiment analysis to categorise the answers into positive, neutral or negative. The overview of these groups could be visualised in a pie chart within in *Social Media Evaluation* modelling objects.

A service in the Olive Framework (see append A.4.10 for more details) was created to handle the communication with the social network platform and the sentiment analysis. The modelling tool then called the Olive microservice, processed the results to save and show them in the model.

The functionality has two steps that must be triggered individually. First, send the picture and the question or statement to the platform. After some comments are made, the results must be collected and integrated into the model. For sending, all not-needed objects in the model should be hidden, like the *Social Media Evaluation* objects. Then, the picture must be created and sent with the provided text. The ID of the created post on the social media platform must be saved to the corresponding *Social Media Evaluation* so that the comments can later be retrieved.

When retrieving the comments, a call is triggered in the modelling tool, which collects all the comments based on the saved ID. Then, for each comment, a sentiment value is assigned. For this, we used the *CoreNLP* (see appendix A.4.2 for more information) library created by the Stanford NLP group. The comments and their assigned sentiment are then saved in the object, which takes the visualisation of a pie chart.

The other prototype to process the digital, conceptual models based on the design thinking technique was to transform them into models or starting points for models in other modelling languages.

For Scene2Model, two prototypes for such transformations were developed. One was to define the actors and their relations to create value nets in the *Ecoviz* tool (cf. [188]) and one to create rule models to be used in Cyber-Physical system (CPS) testing environment to create experiments for evaluating the designs [196].

For the transformation into the EcoViz value nets, the focus was on identifying different relations types between actors. Therefore, the transformation algorithm with relation identification was implemented, based on the AI-based recognition. This allowed us to analyse the arrows further, which were used to represent the relations and identify the text written on them. The text contained different abbreviations, consisting of one or two letters, which were also saved in the relations in the conceptual model. A trained CNN was used to recognise the images. Based on this recognition, the different figures were cut out of the picture and used as input for the text recognition. Text recognition based on the EasyOCR library (see appendix A.4.5 for more information) was used for the prototype.

The Scene2Model prototype and the EcoViz modelling method are implemented on the ADOxx metamodeling platform (see appendix A.4.1). Therefore, the ADOxx standard XML export and import of models were used as bases for the transformation. The rules for the transformation were defined and implemented using the Python lxml library (see appendix A.4.9 for more information) to transform the Scene2Model XML to one that can be imported into the EcoViz tool.

The second prototype for transforming Scene2Model models into the one used for the CPS experiment, an own algorithm was implemented, using ADOxx's AdoScript. This was used as no relations were transformed here. However, the identified sensors, actuators and other devices needed for the experiment were transformed to be imported into the modelling method for controlling the CPS. Therefore, the algorithm searched for the defined classes created in the Scene2Model library, created for this prototype and saved them in a JSON file that can be imported into the other tool.

6.2. Questionnaire Based Evaluation

For the evaluation with questionnaires, workshops with guided cases were prepared, and participants from different backgrounds were invited. Two phases of this evaluation were done. One was done early during the EU-funded *DIGITRANS*³ project and with students from the University of Vienna to evaluate if the idea is accepted and in the second phase, more elaborated workshops were held, where the participants also prepared their own workshops and used the created ideas and models also in other tasks.

The first phase focused on combining physical innovation workshops with digital modelling and an automatic transformation from the physical to the digital. At that stage, the modelling was based on a static, pre-defined modelling method. This prototype did not possess any dynamic adaptation capabilities.

The results of this first questionnaire were published in our paper [4]. For this evaluation, partners from an EU-funded *DIGITRANS* project and students from different courses at the University of Vienna were questioned. The second phase of the questionnaires evaluated the prototype and its usage. Therefore, it was used in teaching, where guided cases were provided to groups, and they had to create innovative ideas using the prototype and physical workshops. These two workshops were held in two consecutive years of the Nemo Summer School (<https://nemo.omilab.org/>).

The main focus of the evaluation as a whole was to get insights into whether it is possible to support creative workshops in the context of finding innovative ideas with model-based digital twins from design thinking and if it is possible how the users can be supported in their working in a natural and supportive way and not to force a new way of thinking and working onto them. Additionally, it was analysed how conceptual modelling can be used in this context. Using it in such a workshop does not follow the requirement of formally and precisely describing systems under study, as it is often needed for modelling for code generation and enterprise modelling. Here, the focus is on conceptual modelling to explore, understand and design the system under study. The goal is not to replace the other modelling approaches but to offer it as a compatible enhancement.

The evaluations of this section were done by applying the prototype in guided workshops and using tag-based recognition. Mock-up workshops in this context mean that the workshops were not used in a concrete real-world innovation project but with a fictional case. This case was introduced to the participants, who then worked with the

³<https://www.interreg-danube.eu/approved-projects/digitrans> (last-visited: 03-01-2024)

6. Evaluation with the Scene2Model Prototype

Scene2Model prototype. Afterwards, we discussed the workshops, used the prototype with the participants, and let them complete the questionnaires.

This section will focus on the last two evaluations and briefly mention the first evaluation, which was already published. For the first evaluation, where the focus was set on getting a first insight into the overall idea is feasible, the whole workshop environment was set up, and the participants had a guided workshop for a provided innovation scenario or one of their own if they had one [4]. The text used for the provided scenario can be found in the appendix under section A.5.1. The results of this first evaluation were positive and indicated that the idea and the first implementation could be helpful in such workshops.

The other two evaluations were made during the Nemo Summer School, where PhD and master students participated. For this evaluation, the participants had to install the prototype on their own devices for each group and then design and present an innovative idea during the summer school. They had to use the prototype independently with technical support and after an introduction. The different approaches for these evaluations will be described in more detail below. The two evaluations were made in two different instalments of the Summer School, with one year between them. Therefore, the two procedures for the evaluation are in more detail below.

6.2.1. Second Workshop Evaluation

This evaluation was conducted during the Nemo Summer School of 2022 and was related to the self-working sessions of the participants. All fifty people participated in the summer school and the working sessions, from which 34 filled out the questionnaire at the end. The overall task of all the participants was to use the Scene2Model environment to create an initial design of an innovative idea in the context of a fictional smart city. Therefore, the participants were separated into groups of 4-6 people, and they could choose their own idea for which an innovative idea should be created, as long as it was in the context of a smart city.

They received an introduction to smart cities through a lecture and to the Scene2Model prototype with a guided workshop as input for their task. The participants received a pre-defined scenario and the overall Scene2Model environment, including the technical infrastructure and a set of paper figures they could use. They had to create an innovative idea for the provided scenario, guided by a workshop leader with experience with such workshops and the Scene2Model environment. However, the focus here was on giving the summer school participants a hands-on introduction to the workshops so they could later use it for their ideas.

For the main task, they had to use the Scene2Model tool for designing their own idea. This included the installation of the tool, preparing the needed paper figures, and adding them to the modelling tool. The participants started with zero figures and had to find them on their own so that they could use them in their own workshops. Therefore, materials for creating the paper figures, like paper, scissors or a printer, were provided. Additionally, the participants got time slots to work, and the technical infrastructure, like the camera and the recognition component, was provided. Even though they had to work alone, experienced people were available to answer questions and help if a group got

stuck. As additional help for the groups, video tutorials were provided to the participants. One part of the questionnaire also focused on evaluating the provided video tutorials.

At the end of the summer school, the participants were asked to fill out a questionnaire voluntarily. They filled out anonymously, and filling one out or not did not influence their grade at all. The questionnaire itself was held short to motivate participation. It first started with asking some information about the participants, then proceeded with questions about the prototype, where the answers were quantified on a scale between one (strongly disagree) and five (strongly agree). Open questions were asked to gather additional information openly. The questionnaire's PDF can be found in the appendix in section A.5.2.

6.2.1.1. Questions of the Questionnaire

This section will introduce the questions used in the questionnaire.

The questions about the personal information were:

- Age range
 - here the choices were between 18 to 24, 25 to 34, 35 to 44 and higher than 45
- Academic field
- occupation
- How knowledgeable are you in the topic of design thinking?
- How knowledgeable are you in the topic of conceptual modelling?

The next part of the questionnaire was separated into two sections. The first focused on questions about the prototype in general, like combining physical and digital modelling and adapting the available figures (digitally and physically) to fit the needs of the concrete scenario. The second section focused on comparing starting the workshop with an initial set of figures and starting without any figures. The first approach was used in the introduction workshop, and the second was used when the participants worked alone. These questions could be answered by choosing between 1 (strongly disagree) to 5 (strongly agree).

The general questions about the Scene2Model prototype were:

- Reflecting on the concepts that were needed during the working session had a positive influence on the created idea/design.
- Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.
- The added value from adapting the figure outweighs the additional effort.
- The digital model outperforms a photograph in capturing the knowledge collected in the workshop.

6. Evaluation with the Scene2Model Prototype

- Combining paper figures and digital models improves the overall experience.
- The tangible paper figures foster creativity and cooperation in the workshops.

The question comparing the two approaches was asked twice, once for the approach with provided figures and once for the approach where the students started without figures.

- I was able to represent all the notational concepts that I wanted to represent.
- The existing concepts in the tool were not sufficient for modelling the desired scenario(s).
- It was easily possible to add additional concepts in the tool according to my needs.
- Based on the time I spent using the tool, I would require more time to feel familiar with its use.

The next part of the questionnaire related to the provided tutorial videos, with the following questions:

- These instructional videos helped me when using Scene2Model.
 - Here one could choose a number from 1 (strongly disagree) to 5 (strongly agree)
- How many times did you watch the videos?
 - Here one could choose 0, 1, 2 or 3+
- Overall, I feel I was able to learn the information from these instructional videos as well as I would have in a face-to-face class presentation.
 - Here one could choose between *Yes* or *No*
- Do you have any feedback regarding the tutorial videos?
 - This was an open question, where one could answer in free text.

The last part of the questionnaire had general open questions to gather additional input from the participants. The questions were:

- What did you like about the Scene2Model tool?
- What did you not like about the Scene2Model tool?
- Do you have any other comments about the Scene2Model tool?

6.2.1.2. Results of the Questionnaire

Here, the results gathered from the answers to the questionnaire and some general information about the people who filled it out will be provided.

Around 70 % of the participants were between 25 and 34 years old, and around 18 % were between 35 and 44 years old. Almost half of them were PhDs, and around 18 % are Maser students. The exact numbers and occupations can be found in the list below:

- Assistant professor: 1
- Data engineer: 1
- IT Project Manager: 1
- Lecturer: 1
- Master student: 6
- PhD: 17
- Postdoc researcher: 2
- Professor: 1
- Teaching assistant: 1
- Technical Project Manager: 1
- University Project Assistant: 1

The most significant part (32,35 %) of the questionnaire participants came from the academic field of computer science. Followed by business informatics at about 14,71 %, information systems at 8,82 %, software engineering at 5,88 % and enterprise modelling at 5,88 %. The rest was split equally, and the whole list of provided academic fields can be found in the list below:

- Business
- Business Information Systems
- Business Information Technology
- Computer science
- Conceptual Modeling for Simulation
- Data science
- Enterprise modelling
- Industrial Engineering

6. Evaluation with the Scene2Model Prototype

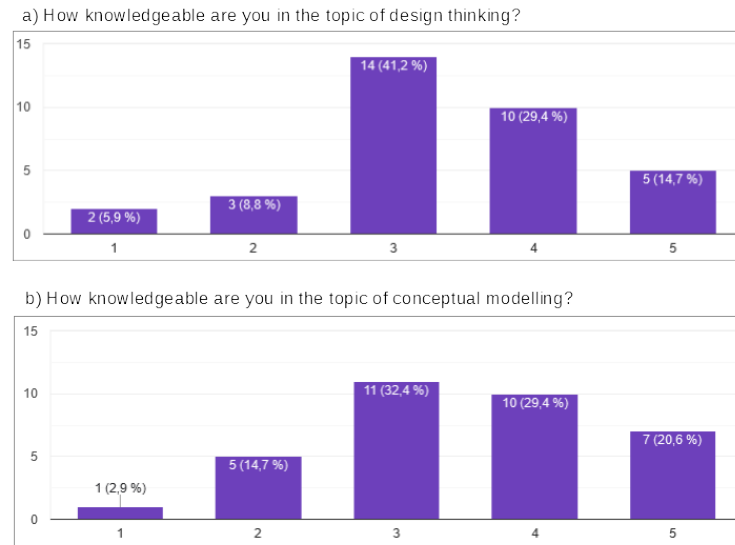


Figure 6.4.: Bar charts on how knowledgeable the participants were about design thinking and conceptual modelling

- Information systems
- IT
- Mechanical Engineering
- Model-Driven Engineering
- Process mining
- Software engineering
- Systems engineering

Most participants answered how knowledgeable they are in design thinking and conceptual modelling, with three (the average) followed by four and then five. Therefore, one can assume that most participants were averagely knowledgeable till not knowledgeable and were no experts. This must be considered when evaluating how well the prototype can help. The participants' overview is visualised in the Figure 6.4.

Because we did not assume that the participants were deeply knowledgeable about modelling and design thinking, we used slots during the summer school to provide the background knowledge needed for the tasks.

Next, the results of the general questions regarding the Scene2Model tool will be introduced and discussed. The graphics and discussion will use a number scale, as used in the questionnaire, from 1 (strongly disagree) to 5 (strongly agree).

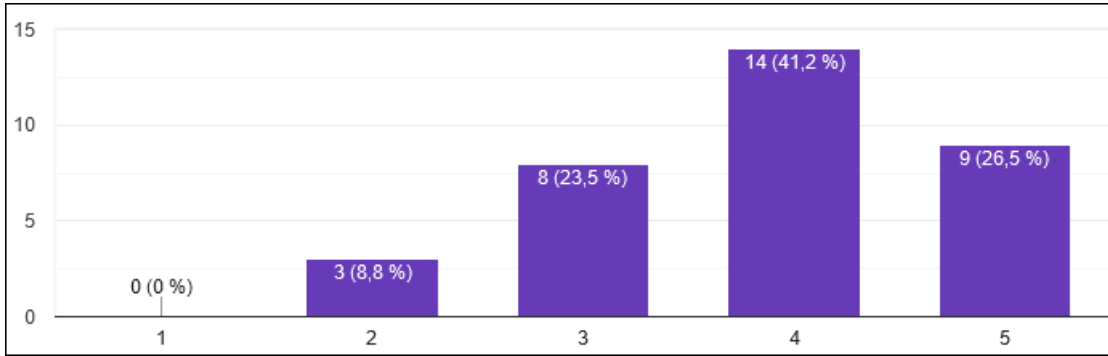


Figure 6.5.: Bar charts of the answer to the question *Reflecting on the concepts that were needed during the working session had a positive influence on the created idea/design.* in the second evaluation

The first question in this section was *Reflecting on the concepts that were needed during the working session had a positive influence on the created idea/design.* Almost half of the participants (41,2 %) chose the value 4, which can be interpreted as their agreement. The following two most chosen values are five and four, with nearly the same amount of about a quarter of the participants, and no one has chosen value 1. The results as a bar chart can be seen in Figure 6.5.

For the question *Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.*, also half of the participants chose 4 (agree) followed by 5 and 3, and no one chose 1. The bar chart with the answers can be seen in Figure 6.6.

From these two questions, we conclude that the participants deemed it important to reflect on the concepts used to describe and design their scenarios and ideas. Additionally, they seem to think adapting the concepts to their specific domain is important. This leads to better capturing and understanding of the created models and overall support for their design process.

For the next question, *The added value from adapting the figure outweighs the additional effort.*, the answers were more evenly distributed over the scale. The most chosen answer is still 4 (with about 32 %), but the following two chosen answers are 2 and 3 with the same amount. The exact answers can be seen in Figure 6.7. Therefore, the trend of answers has shifted towards *disagree*, but most of the participants still chose between 3 (neutral) and 5 (strongly agree).

This leads to the conclusion that even though reflecting on the concepts and adapting to the concrete domain was considered valuable, the effort needed to do so was considered cumbersome. Moreover, although the answer to the question suggests that it is not the biggest problem, it identified room for improvement. This was also the reason to introduce and improve the instance base metamodeling idea (see section 5.1.2 for more details). Additionally, it was concluded that a set of pre-defined figures should be available within a workshop to ease the start.

6. Evaluation with the Scene2Model Prototype

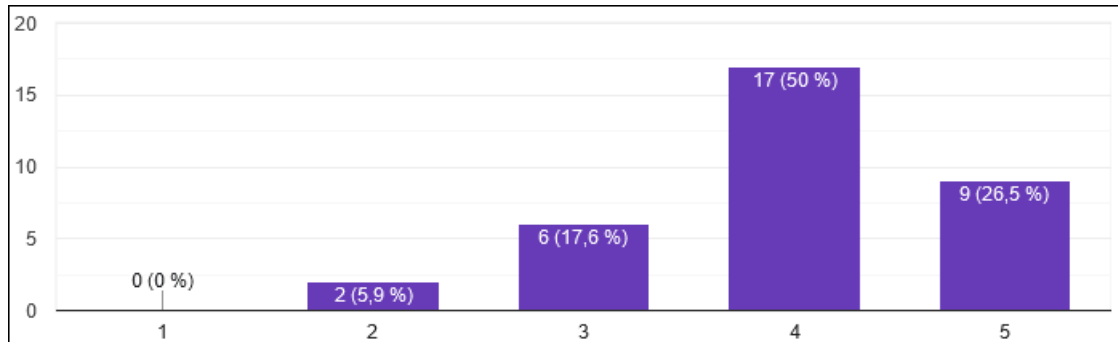


Figure 6.6.: Bar charts of the answer to the question *Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.* in the second evaluation

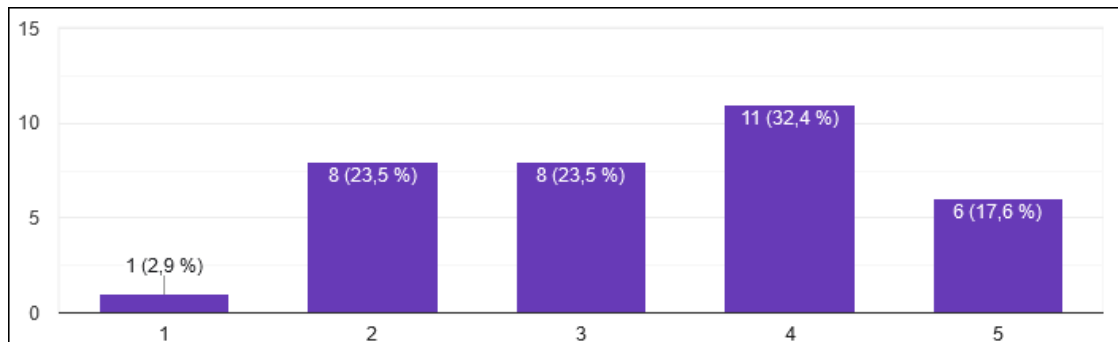


Figure 6.7.: Bar charts of the answer to the question *The added value from adopting the figure outweighs the additional effort.* in the second evaluation

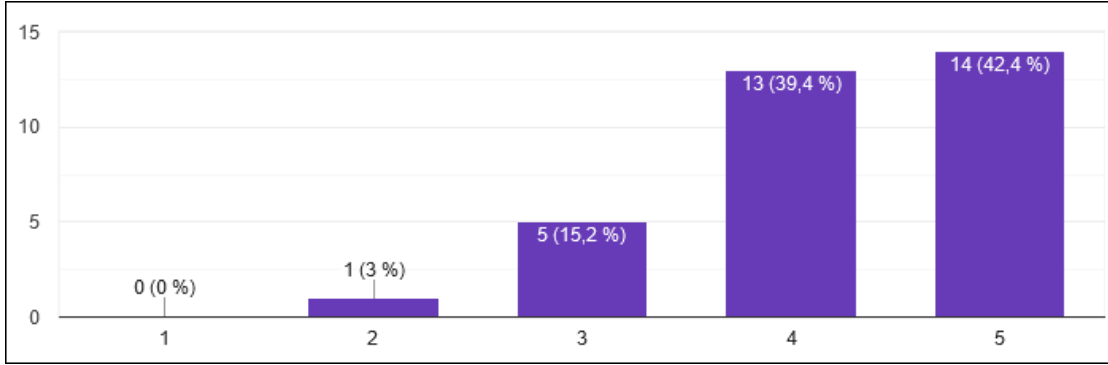


Figure 6.8.: Bar charts of the answer to the question *The digital model outperforms a photograph in capturing the knowledge collected in the workshop.* in the second evaluation

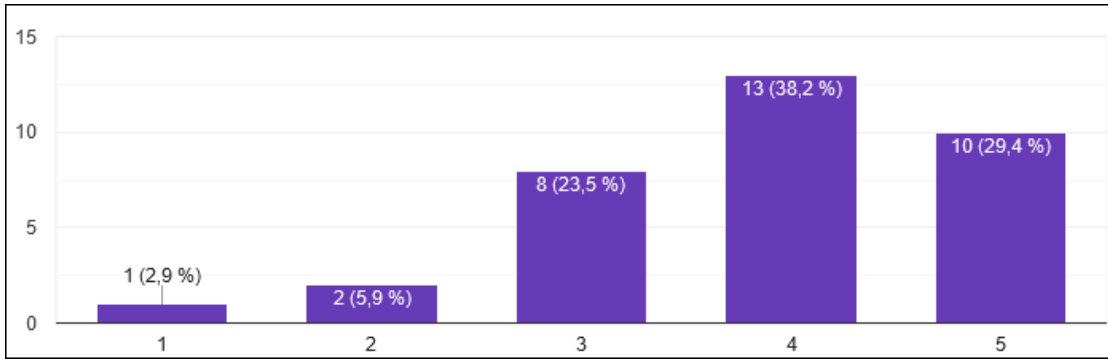


Figure 6.9.: Bar charts of the answer to the question *Combining paper figures and digital models improves the overall experience.* in the second evaluation

The next question was focused on the difference between using digital, conceptual models or pictures to capture the results of a workshop. The concrete question was *The digital model outperforms a photograph in capturing the knowledge collected in the workshop.* The answers are visualised in Figure 6.8, and 33 of 34 participants answered this question. From them, most (about 80 %) chose 5 (strongly agree) and 4 (agree), and about 15 % chose 3.

The next question was *Combining paper figures and digital models improves the overall experience.*, focuses on using paper figures and digital models together. The detailed results are visualised in Figure 6.9, and in short, the overall feedback was positive, where the most chosen answer was 4, followed by 5 and 3.

The results of the last two introduced questions were interpreted as the participants seeing the digital models as beneficial to mere pictures, leading to the hypothesis that the content of this thesis is also beneficial. Especially considering that most participants also liked combining the physical means, aka the paper figures and the digital models. This result assured us that we should not focus on just one means, digital or physical,

6. Evaluation with the Scene2Model Prototype

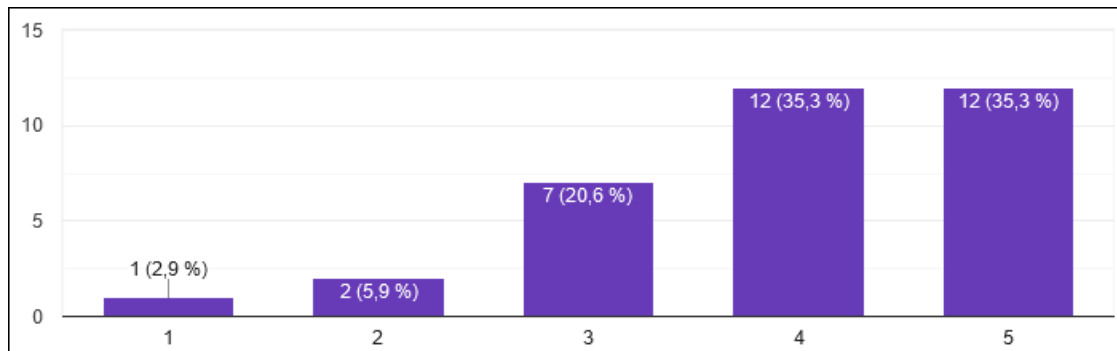


Figure 6.10.: Bar charts of the answer to the question *The tangible paper figures foster creativity and cooperation in the workshops.* in the second evaluation

in designing innovative ideas but support both and let the users choose their preferred method.

The last question for the general part was *The tangible paper figures foster creativity and cooperation in the workshops.* and focused, together with the previous question before, on how working physically together improves the experience. The detailed results are shown in Figure 6.10, but are interpreted as positive, as most chosen answers were 5 and 4, followed by three. This is additionally interpreted as using both the physical and digital parts of the prototype, which provides value. Not replacing one of them but combining them can generate benefits for users.

The following questions of the questionnaire were separated into two blocks, containing the same questions but for different tasks of the summer school. This was done to compare the two different approaches the students used. One workshop where the case and figures were provided, and the case where they had to find their own figures to use for their design. This insight into how people work and which approach is easier for them. Therefore, the results of the questions are shown next to each other so that they can be compared. On top, marked with (A) are the answers for the workshop where figures were provided and on the bottom, marked with (B) are the answers for the workshops where the participants had to find their own figures.

The results of the first comparing question are visualised in Figure 6.11. In both cases, nearly half (44,1 %) of the participants chose 4 (agree). By both, 5 was chosen next the most frequently, but for (B), more people chose 5. Overall, for (A), the results were more scattered, and more results in the direction of one 1 were chosen. Therefore, we concluded that even though people could represent their ideas in both cases, it would be better if people searched for and chose their own notations for the figures used.

The second question in this comparison section was focused on evaluating if the concepts in the modelling tool were sufficient to represent their idea. It was stated negatively as a counterpoint to the first comparison question, as it has a similar goal. Therefore, the results must be interpreted differently, where 1 means that the Scene2Model supported the user and 5 means that it did not. The results can be seen in Figure 6.12.

I was able to represent all the notational concepts that I wanted to represent.

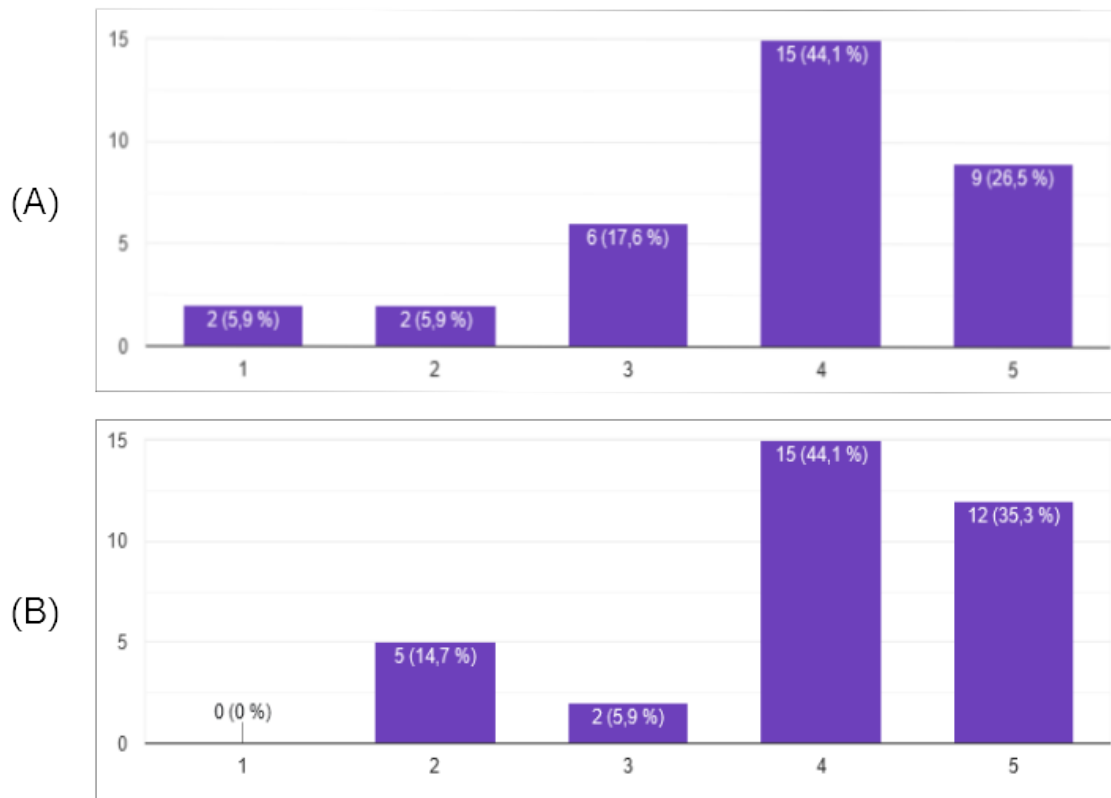


Figure 6.11.: Bar charts of the question *I was able to represent all the notational concepts that I wanted to represent.* compared between workshops with and without pre-defined figures

6. Evaluation with the Scene2Model Prototype

The existing concepts in the tool were not sufficient for modeling the desired scenario(s).

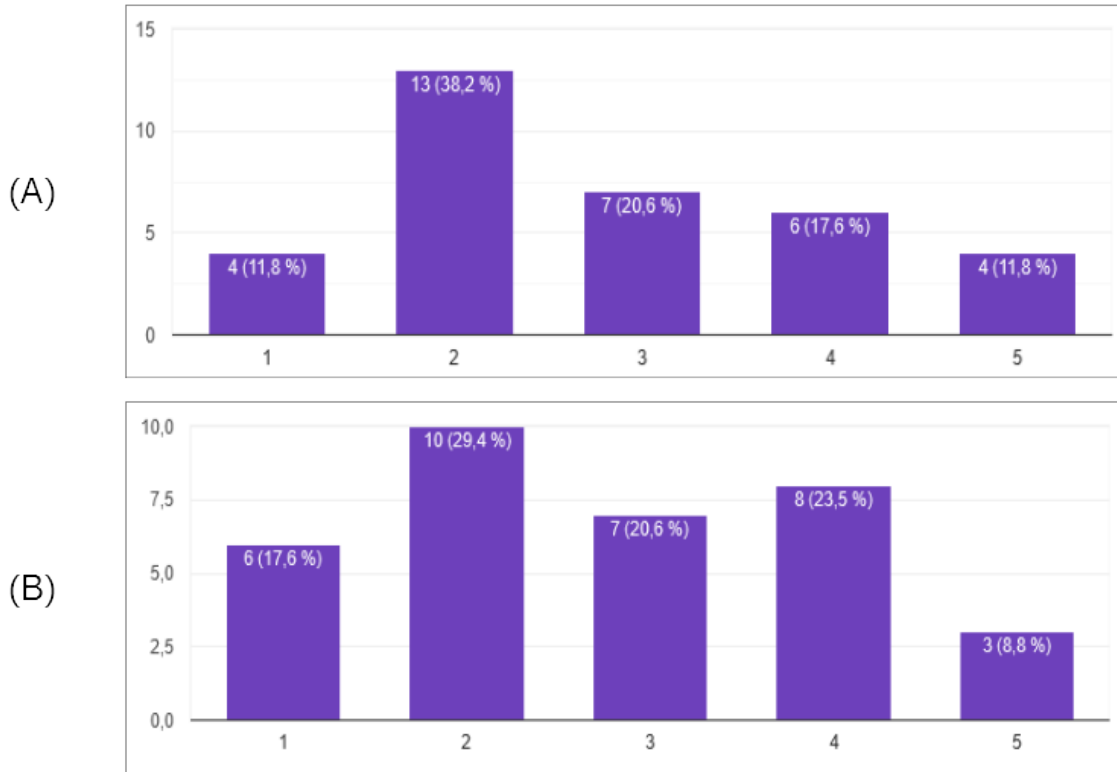


Figure 6.12.: Bar charts of the question *The existing concepts in the tool were not sufficient for modelling the desired scenario(s).* compared between workshops with and without pre-defined figures

The trend is the same as for the first comparison question, but the results for the second one were more distributed. Therefore, the conclusion made for the first question is not invalidated, and the participants represent their ideas approximately equally well.

The next question focused on the adaptability of the two approaches. The concrete question was *It was easily possible to add additional concepts in the tool according to my needs.* and the results are visualized in Figure 6.13.

The results for both questions are nearly identical, with 4 (agree) being chosen the most. Overall, the answers to these questions tended to be positive (in the direction of 5 - strongly agree). This means that for the participants, it was for both variants to add the figures they needed by choosing them from the overall set of provided ones or finding, defining and adding the figures on their own.

Last but not least, the last question asked for both usage variants of the tools was *Based on the time I spent using the tool, I would require more time to feel familiar with its use.*, where the results are visualised in Figure 6.14. The results show that the participants

It was easily possible to add additional concepts in the tool according to my needs.

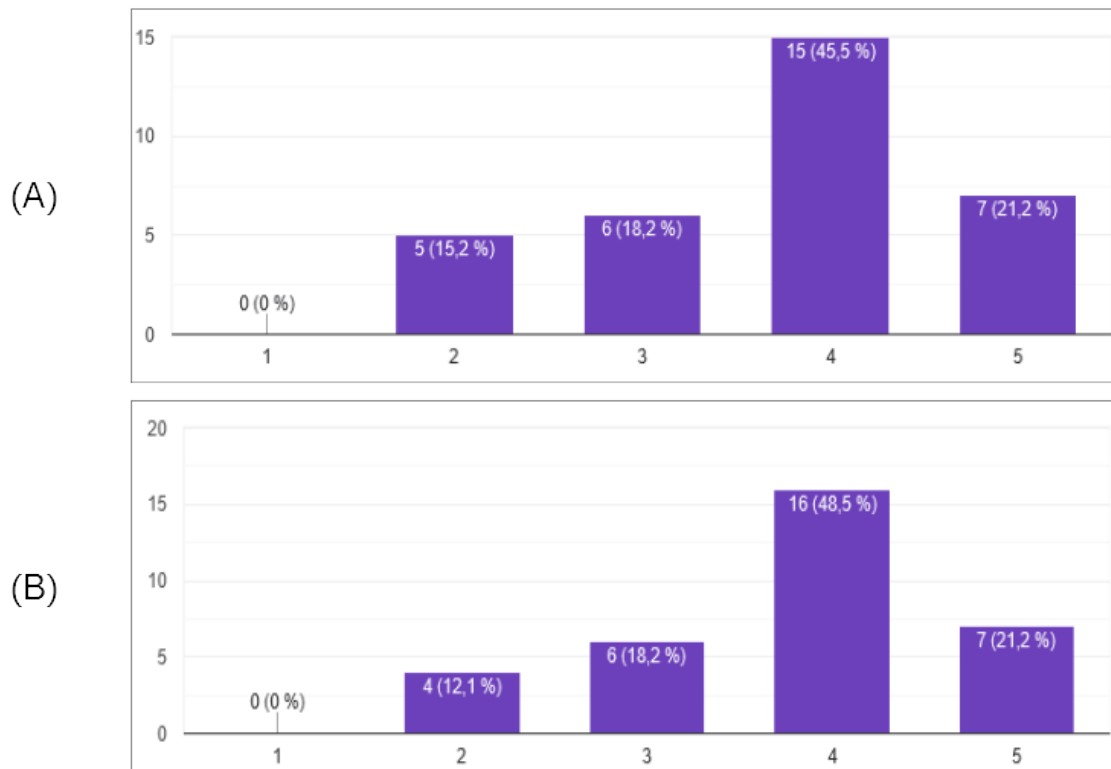


Figure 6.13.: Bar charts of the question *It was easily possible to add additional concepts in the tool according to my needs.* compared between workshops with and without pre-defined figures

6. Evaluation with the Scene2Model Prototype

Based on the time I spent using the tool, I would require more time to feel familiar with its use.

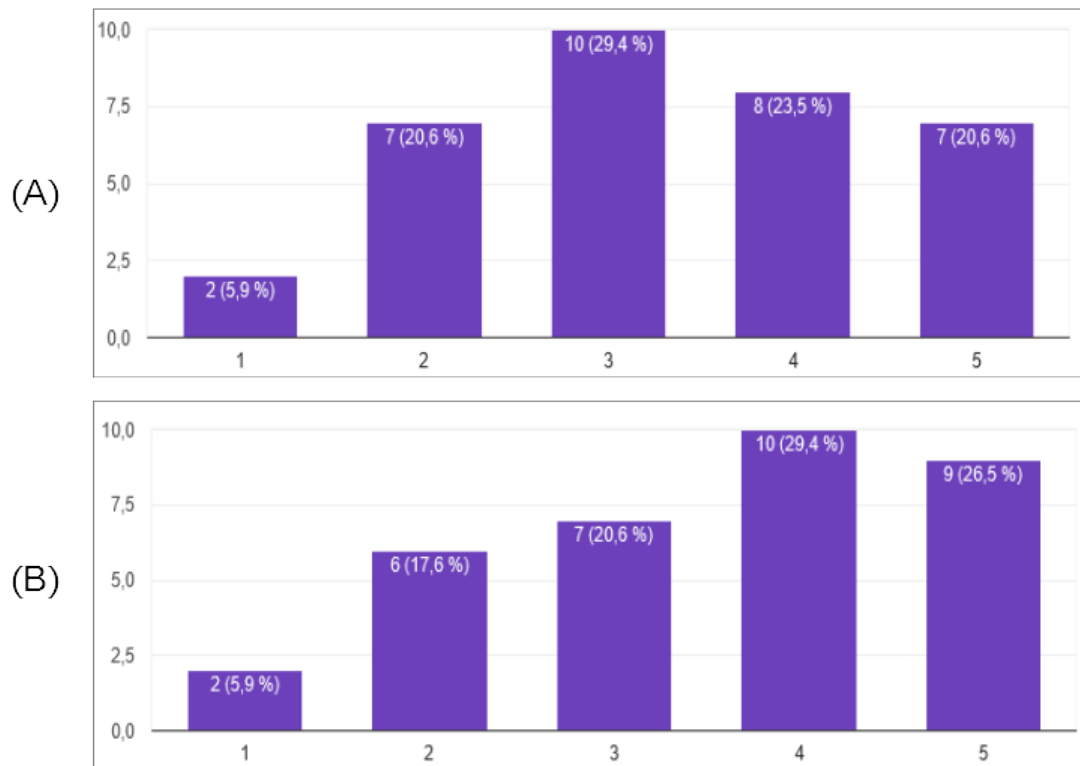


Figure 6.14.: Bar charts of the question *Based on the time I spent using the tool, I would require more time to feel familiar with its use.* compared between workshops with and without pre-defined figures

need more time to familiarise themselves with the prototype, especially with the variant where they created their own. This part can be improved by improving the introduction, supporting materials, or prototype.

The following four questions of the questionnaire focused on the tutorial videos, which were provided by the participants, whereby the first question was *These instructional videos helped me when using Scene2Model.* and the 32 answers are visualised in bar charts of Figure 6.15. The most chosen answer was 5 (strongly agree), by 34,4 % of the answers. The trend is generally going towards 5, so the participants agree that the videos helped them use the tool.

The following two questions are concerned with how often the participants watched the videos (see (A) of Figure 6.16) and how good they learned from the videos compared to the face-to-face presentation (see (B) of Figure 6.16).

There was no identifiable relation between how often a participant watched a video and how useful the videos were perceived. However, the better part of the participants (41,2 %)

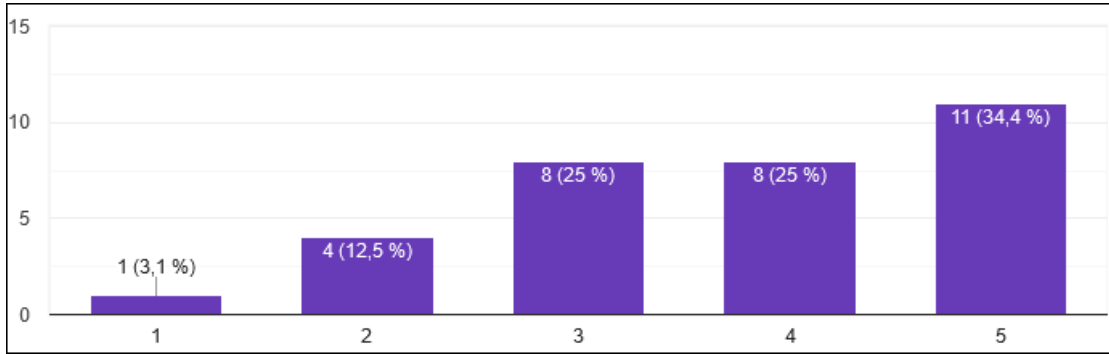


Figure 6.15.: Bar charts of the question *These instructional videos helped me when using Scene2Model.*, evaluating the teaching videos provided to the participants.

watched the video once, and nearly a third never watched the videos. However, most participants (71 %) found the videos as understandable as a face-to-face introduction.

The last question regarding the tutorial videos was the open question: *Do you have any feedback regarding the tutorial videos?* For this, we received the following seven answers, listed below. We gathered no other input for the future other than providing more videos. Therefore, the tutorial videos need to be improved and planned more granularly if the prototype is to be further developed.

- no
- Much knowledge
- Tutorial videos were fine.
- For me, the tutorials were detailed, yet clear enough, and really helpful in settling down the information.
- I did not watch the videos :(
- Please do another videos for using this tool.

The end of the questionnaire consisted of three open questions, allowing the questionnaire participants to provide some general feedback. Not all the provided answers will be listed here, but an aggregation. The list with all the information is provided in appendix A.5.3.

The first question was *What did you like about the Scene2Model tool?*, and what the participants seemed to like was that the method was intuitive and easy to use. It helped their process that they could start by interacting with tangible objects, supporting the structuring of their thoughts. Afterwards, the digitalisation of the tangible object seemed well received, so they could save it and improve the scenes in parallel.

6. Evaluation with the Scene2Model Prototype

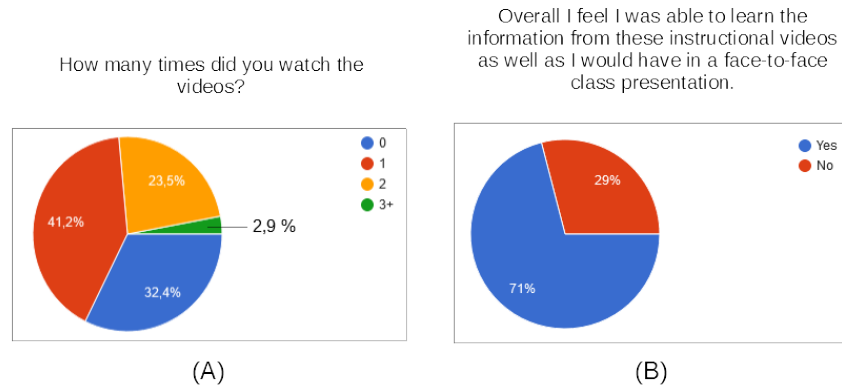


Figure 6.16.: Pie charts for the answers to two questions regarding the video

The second question was *What did you not like about the Scene2Model tool?* and one problem mentioned multiple times is that the user-friendliness and the user interface could be improved to ease working with the tool. Additionally, the robustness of the overall technical components can be improved so that the work is not startled by technical problems. One participant mentioned that the metamodel was too constricting, which hindered their design, and another participant stated that this was too much overhead for this task. Also, one participant wanted to have the additional functionality of rotating the modelling objects in the tool.

The last question was *Do you have any other comments about the Scene2Model tool?* and this got two answers with new content. One was that even though the idea seems promising, more added value can be generated when the technology further supports the physical and digital parts and their combination. The other input was regarding the improvement of the working sessions during the summer school, which is not part of this thesis.

Summarising, using physical workshops and digital models and supporting their combination through an automated transformation is beneficial and should be further pursued. Staring workshops with no figures at all can hinder the overall process. Therefore, starting with a pre-selection and then adapting it during the workshop to tailor it to the domain is better. Additionally, the tool could be improved through stability, user interface, and user guidance during usage, which are essential aspects for a finished product but are not the main focus of a scientific prototype.

6.2.2. Third Workshop Evaluation

The third evaluation was conducted during the Nemo Summer School 2023 and was applied during self-working and guided working sessions. The Scene2Model prototype was used in two ways, collectively evaluated with questionnaires. 36 students participated in the sessions, of which 34 answered the questionnaire.

The first part was similar to the above evaluation (see section 6.2.1). The participants used the Scene2Model prototype to create an innovative idea in the smart cities domain within self-working sessions. They were introduced to innovation, smart cities, design thinking, and the prototype as preparation. One guided introduction workshop was held for a pre-defined use case. Then, the students had time to use the tool independently to create innovative ideas presented at the end of summer school.

Independent of this task, the participants had another assignment, where they used the Scene2Model prototype. In this assignment, it was used within OMILAB's *Digital Innovation Environment (DIEn)* [54]. Here, the participants were separated into three groups, and each group had its own topic, where they had to create an innovative idea and a high-level prototype using IoT and conceptual modelling. The Scene2Model prototype was used in a workshop, guided by an experienced summer school staff, to let the participants explore their problem domain and ideate an innovative idea. Afterwards, they were introduced to technical equipment, consisting of a Raspberry Pi, an Arduino, and fitting sensors and actuators to create a physical environment to test their idea. The technical equipment included software to connect and use the sensors and actuators and a conceptual modelling tool to control the created environment. The description here will not detail the physical environment and the models for controlling them, as this would go beyond the scope of this thesis. More information on this and parts of the evaluation were published in [196]. Here, we will focus on using the Scene2Model prototype in this context and the part of the evaluation corresponding to that.

The third evaluation used one questionnaire containing questions about all the tasks. However, this thesis will only discuss the questions regarding the Scene2Model prototype. The questionnaire with all questions can be seen in appendix A.5.4. Below, we will discuss the questions for this thesis.

6.2.2.1. Questions of the Questionnaire

This section introduces the questions used in the third workshop evaluation, whereby questions not related to the Scene2Model prototype are left out in this section.

The questionnaire started with questions regarding the participants, where the following were used:

- Age range
 - here the choices were between 18 to 24, 25 to 34, 35 to 44 and higher than 45
- Academic field
- occupation

6. Evaluation with the Scene2Model Prototype

- How knowledgeable are you in the topic of design thinking?
- How knowledgeable are you in the topic of conceptual modelling?

The questions regarding the Scene2Model prototype were based on a scale from 1 (strongly disagree) to 5 (strongly agree). The first questions focus on getting insights into the tool that could be used to improve the physical workshops. In the end, the questions consider if the tangible aspects held added value, how well the participants could represent their ideas with the semantic-rich figures and if they would have needed more time to better familiarise themselves with the tool.

The questions regarding Scene2Model were the following:

- The haptic design thinking environment was beneficial in generating innovative scenarios.
- The Scene2Model tool effectively supported the digitalisation of the created scenarios in the haptic design thinking environment.
- Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.
- The added value from adapting the figure outweighs the additional effort.
- The digital model outperforms a photograph in capturing the knowledge collected in the workshop.
- The tangible paper figures foster creativity and cooperation in the workshops.
- Within the workshop and the Scene2Model tool, I was able to represent all the notational concepts that I wanted to represent.
- Based on the time I spent using the tool, I would require more time to feel familiar with its use.

6.2.2.2. Results of the Questionnaire

In this section, we will present and discuss the results gathered through the questionnaire.

Most participants (about 62 %) were between 25 and 34 years old, followed by about 24 % who were between 18 and 24. About 9 % were between 35 and 44 years old, and about 5 % were over 45.

Regarding their academic fields, most participants (about 20 %) have a background in *Computer Science*, followed by about 18 % with a background in *Business Informatics* and about 10 % in *Information System*. The rest are mentioned nearly the same amount of times. It should be said that participants were able to provide more than one academic field. Below, we present the collected academic field alphabetically:

- Business Administration

6.2. Questionnaire Based Evaluation

- Business Informatics
- Business Intelligence
- Business Process management
- Computer Science
- Conceptual Modelling
- Cultural Heritage
- Digital Twins
- Economics
- Electrical Engineering
- Engineering
- Enterprise Architecture
- Industrial Engineering
- Information management
- Information Systems
- Information Technology
- Management
- Requirements engineering
- Software Engineering

About 35 % of the participants were PhD students, followed by about 14 % who were active in teaching at a university level. About 9 % stated that they were students, and about 5 % were research assistants. Here again, participants could state more than one occupation. Below, the stated occupations are listed, together with how often they were mentioned. The list is sorted after the amount of how often they were mentioned:

- PhD Student: 12
- Teaching: 5
- Student: 3
- Research Assistant: 2
- Analyst: 1

6. Evaluation with the Scene2Model Prototype

- Bachelor Student: 1
- Business Intelligence: 1
- Cio: 1
- Consultant: 1
- Data Advisor: 1
- Digital Twins: 1
- E-Commerce: 1
- Machine Learning Contractor: 1
- None: 1
- Left Empty: 1
- Requirements Manager: 1
- Robotic Process Automation Developer: 1
- Technical Support: 1

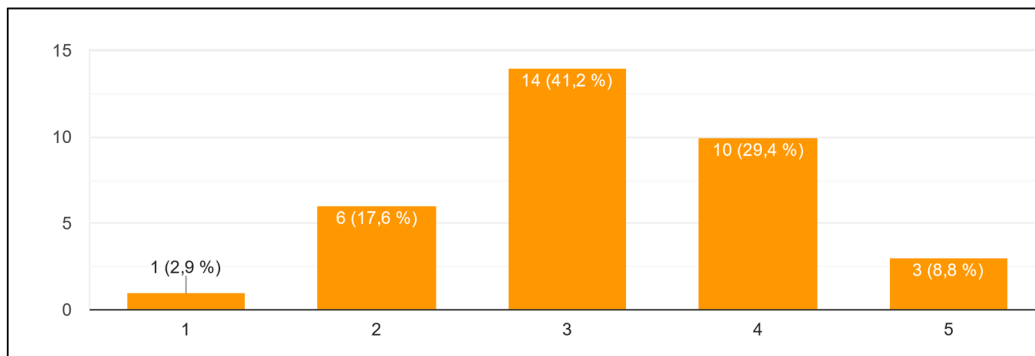
The last information about the participants, which is interesting for the discussion here, is how knowledgeable the participants are in design thinking and conceptual modelling, shown in the bar charts of Figure 6.17. More participants were knowledgeable in design thinking than conceptual modelling. However, in both cases, they tended to be average to knowledgeable in these categories. This fact has to be considered when interpreting the results of the questionnaire.

During the preparation for the working session in the Nemo summer school, missing knowledge was identified as a possible problem; therefore, the introductions to the topics were prepared.

The first question regarding evaluating the Scene2Model prototype asked how helpful the haptic design thinking environment was for creating the innovative idea. The results are visualised in Figure 6.18. 76,5 % of participants stated that they *agree* or *strongly agree* with this statement, which is more than three-quarters. This led to the conclusion that most participants found haptic design thinking helpful for creating their ideas. This strengthens our thesis that having physical workshops helps create innovative ideas and should not completely be replaced through digital means.

The next question focused on the support the Scene2Model tool provided in digitalising the haptic model, with the question *The Scene2Model tool effectively supported the digitalisation of the created scenarios in the haptic design thinking environment..* The answers are visualised in Figure 6.19. 65,7 % stated that they *strongly agree* or *agree*. Leaving 20,6 % with three points and 14,7 % saying that they *disagree* or *strongly disagree*.

a) How knowledgeable are you in the topic of design thinking?



b) How knowledgeable are you in the topic of conceptual modelling?

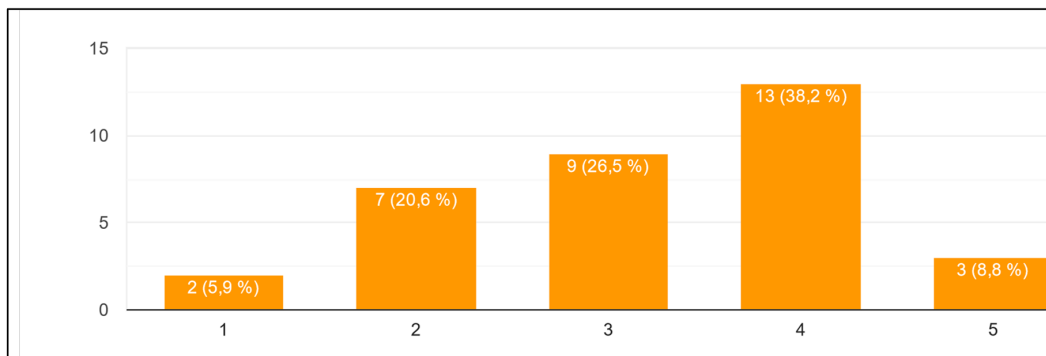


Figure 6.17.: Bar charts of the answers to the questions *How knowledgeable are you in the topic of design thinking?* and *How knowledgeable are you in the topic of conceptual modelling?* in the third evaluation.

6. Evaluation with the Scene2Model Prototype

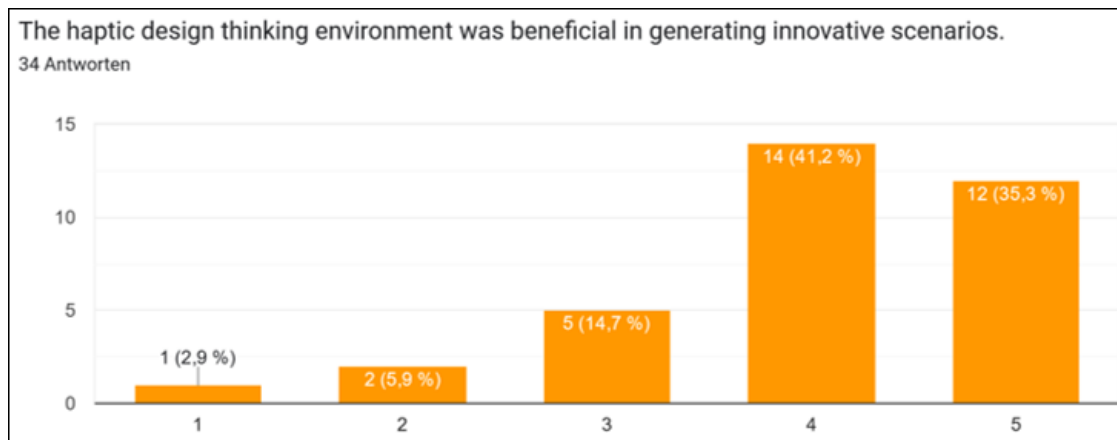


Figure 6.18.: Bar charts of the answer to the question *The haptic design thinking environment was beneficial in generating innovative scenarios.* in the third evaluation.

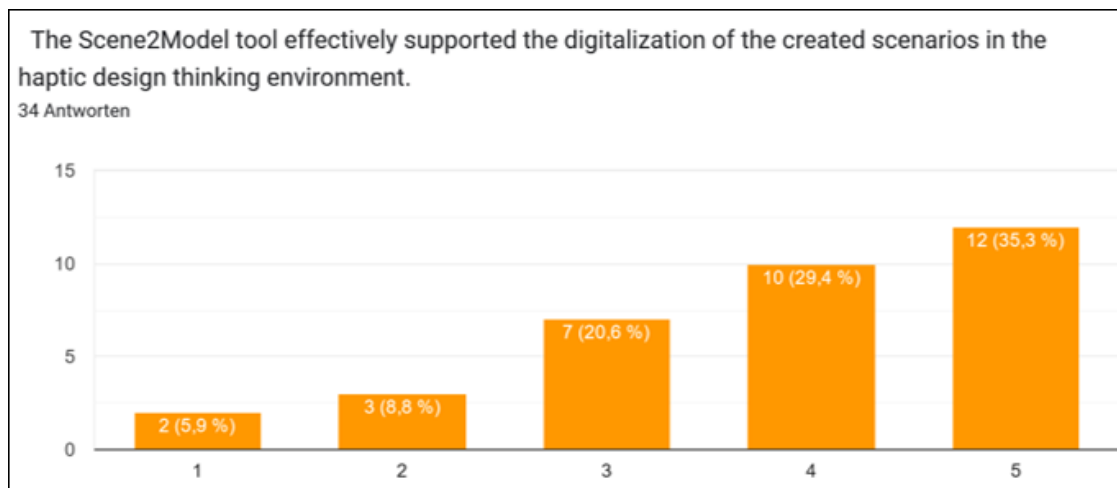


Figure 6.19.: Bar charts of the answer to the question *The Scene2Model tool effectively supported the digitalisation of the created scenarios in the haptic design thinking environment.* in the third evaluation.

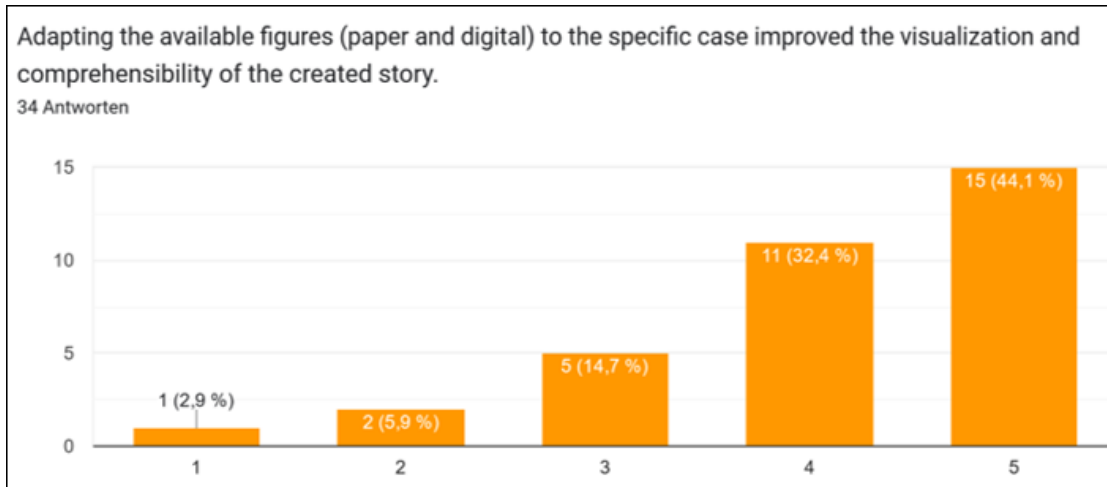


Figure 6.20.: Bar charts of the answer to the question *Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.* in the third evaluation.

Therefore, the prototype helped to digitally represent the tangible model, easing future utilisation. However, even though it is helpful, there is room for improvement if it should be used outside scientific prototypes.

The next question focused on the need for adaptability of what can be represented in the physical and digital models. The concrete questions were: *Adapting the available figures (paper and digital) to the specific case improved the visualisation and comprehensibility of the created story.* This was asked to get insight into how functional the adaptation can be to better represent the users' ideas. The results are visualised in Figure 6.20.

The response to the benefits of the adaptations was very positive, as more than three-quarters (76,5 %) of the participants chose *strongly agree* or *agree*. Therefore, we conclude that it is essential for the participants to have semantic-rich icons which possess meaning in their domain. This must be supported by the modelling environment by quickly adapting the notation of the models, both in the physical and digital environment.

The next question compared the additional effort needed to adapt the figures to the perceived added value. The concrete question was: *The added value from adapting the figure outweighs the additional effort.* The question was added, as interacting with the modelling environment to adapt the visualisation poses additional effort to users. It was about the added value of an easier interpretable visual representation. The results of these questions are visualised in Figure 6.21.

69,7 % of the participants *agreed* and *strongly agreed* that the added value outweighs the additional effort. This was interpreted as a beneficial technique for tailoring the visualisation to the concrete workshop participants and should remain an important capability of the modelling environment.

Next, we compared the model-based design thinking digital twin with just taking a photograph of the physical model by asking *The digital model outperforms a photograph*

6. Evaluation with the Scene2Model Prototype

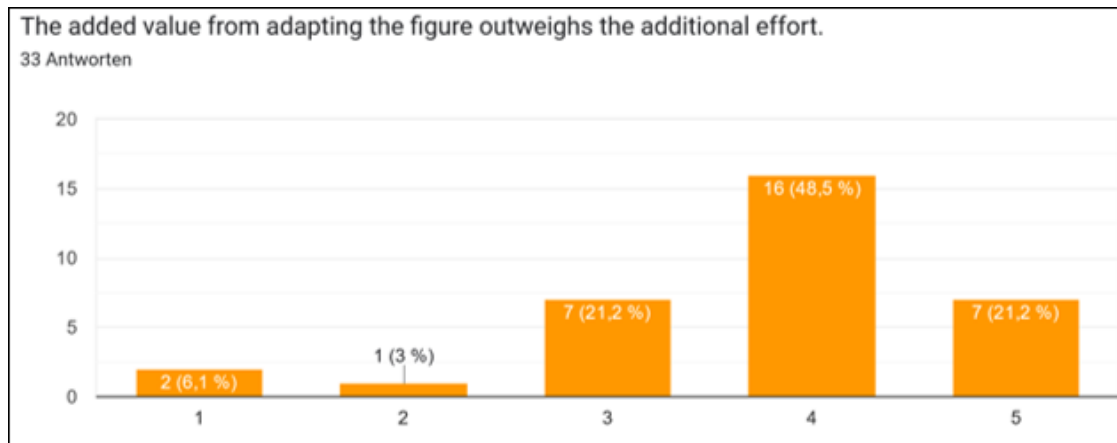


Figure 6.21.: Bar charts of the answer to the question *The added value from adapting the figure outweighs the additional effort.* in the third evaluation.

in capturing the knowledge collected in the workshop. The motivation behind this question was to evaluate if the additional effort by setting up and using the modelling environment does not outperform the additional needed effort. The results are visualised in Figure 6.22.

We interpreted the results as positive, as 67,7 % of the participants *agreed* or *strongly agreed*, that the digital twin outperforms a mere photograph. However, 28,5 % were in the middle by voting three. This leads to the conclusion that they do not prefer the photograph but could be more motivated to use the digital twin. Here, future work can focus on providing more functionality based on the digital model to improve its usefulness further.

With the next question, we wanted to evaluate further if using physical workshops with tangible figures is perceived as beneficial. Therefore, the question *The tangible paper figures foster creativity and cooperation in the workshops.* was used. Here, insights on using both the physical and digital models should be gathered to offer every aspect purely digital. Therefore, we enquired if the physical means fostered the creativity and cooperation of the participants during the workshop. The results are shown in Figure 6.23.

Over half of the received answered (51,5 %) *strongly agreed* with this statement and reinforced the goal of using the physical and digital aspects together. Additionally, 21,2 % agreed with the statement, and 12,1 % voted with the number 3. We interpreted the results as the physical workshops being beneficial and using tangible figures to improve working in the workshops. Therefore, the physical workshops should be further supported.

Above, we positively evaluated whether the adaptation of the notation was important for the participants. With the next question, we evaluated if they could represent everything they needed physically and digitally. The results are shown in Figure 6.24.

Only 11,7 % stated that they were not able to represent everything they needed. However, 61,8 % *strongly agreed* and *agreed*, that they could do so. We concluded that the capabilities offered enabled the users to represent what they needed. For future work in this context, we could further analyse what the participants could not represent and if

6.2. Questionnaire Based Evaluation

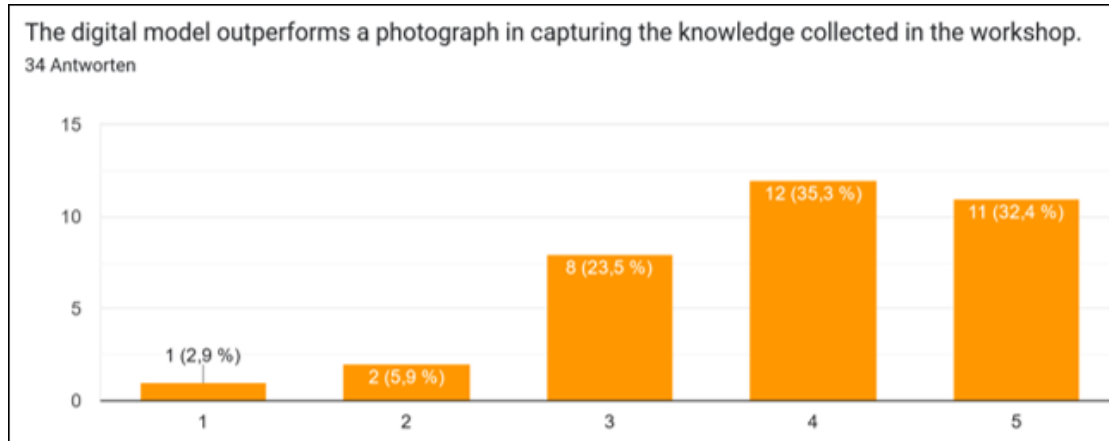


Figure 6.22.: Bar charts of the answer to the question *The digital model outperforms a photograph in capturing the knowledge collected in the workshop.* in the third evaluation.

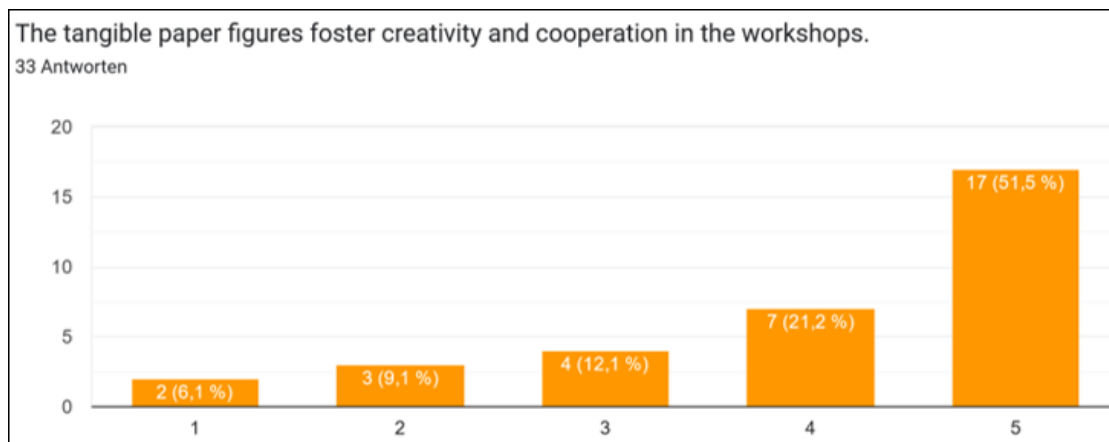


Figure 6.23.: Bar charts of the answer to the question *The tangible paper figures foster creativity and cooperation in the workshops.* in the third evaluation.

6. Evaluation with the Scene2Model Prototype

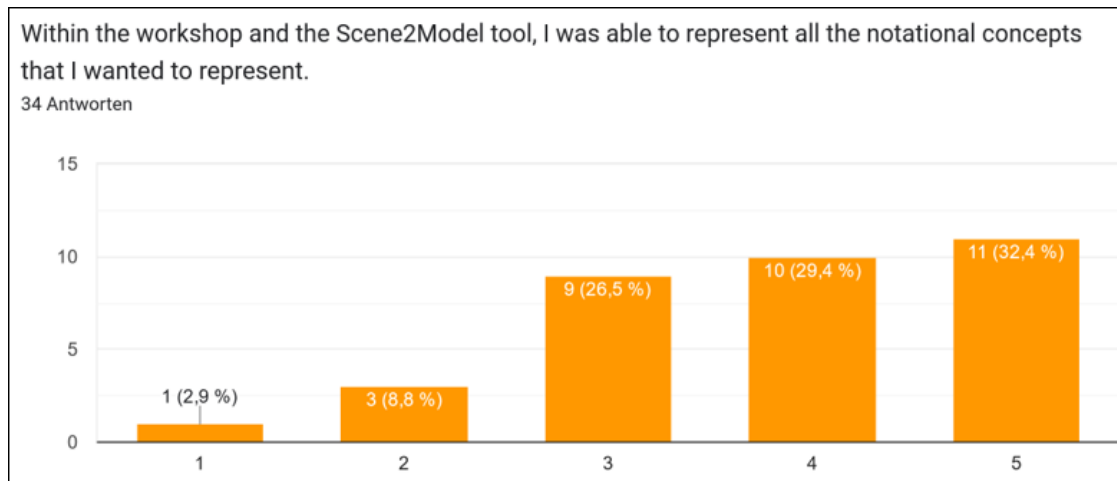


Figure 6.24.: Bar charts of the answer to the question *Within the workshop and the Scene2Model tool, I was able to represent all the notational concepts that I wanted to represent.* in the third evaluation.

these shortcomings were based on the offered capabilities or missing explanations of how they could be achieved.

The last question regarding the Scene2Model prototype was added to evaluate whether the time was sufficient to learn the tool. The concrete question was: *Based on the time I spent using the tool, I would require more time to feel familiar with its use.* The results are shown in Figure 6.25.

50 % of the participants *strongly agreed* or *agreed* that they require more time to better familiarise themselves with the tool. Additionally, 29,4 % answered this question by voting in the middle. This was interpreted that the participants generally would have needed more time to get to know the tool better. This can be improved by either planning the workshops differently or making the tool more intuitive. Another way to tackle this problem is to suggest to possible users that the organiser should be familiar with the tool and provide technical support to improve the overall results by reducing the need for each participant to learn the tool in depth.

In summary, having both the physical and digital modelling, together with the automated processing, was beneficial for the participants. Additionally, adapting the figures (physically and digitally) is useful in representing innovative ideas. Room for improvement could be identified by the additional effort and the added value. Here, future work could be done to ease the adaptation or to improve the added value. Additionally, new ways to support the visualisation of the concepts could be researched.

6.2. Questionnaire Based Evaluation

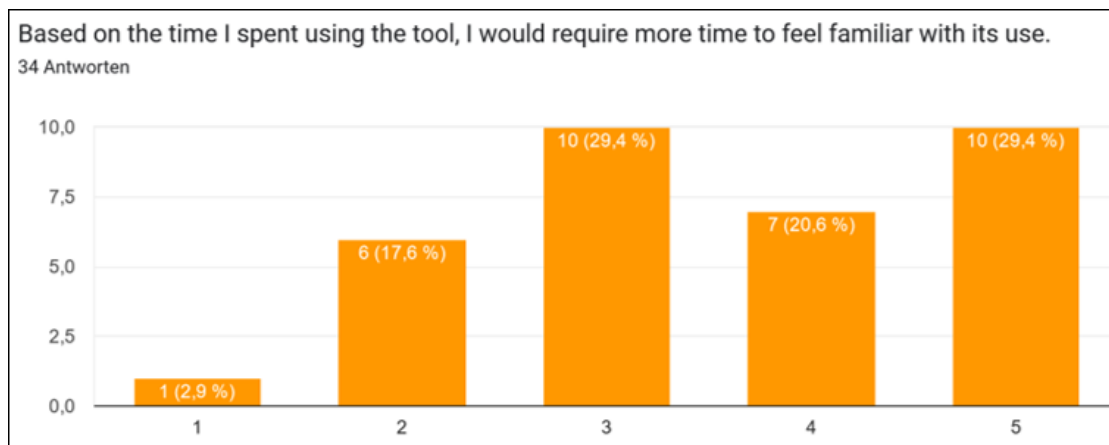


Figure 6.25.: Bar charts of the answer to the question *Based on the time I spent using the tool, I would require more time to feel familiar with its use.* in the third evaluation.

7. Conclusion and Outlook

The final chapter of the thesis contains the conclusion and outlooks regarding exciting aspects of the future. Therefore, the important findings and the research approach will be summarised shortly, and then the thesis result will be reflected in the context of the research goal. Finally, the outlook with topics for future work will be presented.

7.1. Conclusion

The research conducted for this thesis focused on designing an application to support the creation of innovative ideas through physical design thinking workshops. This application should not replace the physical workshop but support the users in their way of work. Therefore, the notion of digital twins was repurposed for what we call digital twins from design thinking, allowing an automated transformation from tangible models created in physical design thinking workshops into digital, conceptual models. These digital models are then instantly available and can be used within and after the workshop, enabling better utilisation of the created knowledge and the designed innovative idea.

Physical design thinking workshops emphasize creativity, freedom, and efficient communication among the participants in designing innovative ideas. This is supported by using tangible means, like paper figures or sticky notes, to create the design with a corresponding visualisation. However, using conceptual modelling in this context poses new challenges, as typical modelling methods are not designed to deal with the needed flexibility to capture the domain, which is not fully known.

This thesis designed the *Adaptive Modelling Environment* as the environment for applying the digital twin from the design thinking concept. For this modelling environment, it is essential that not only the models can be processed but that the metamodel of the modelling method can be dynamically adapted to fit the users. The modelling method must be changed, and the modelling environment and its functionality must be adapted to fit the user's needs.

To enable the adaptation of the modelling environment, we differ between the concept of the modelling environment and the modelling tool. The modelling environment is a term used for all the digital components, and the modelling tool manages the models and the metamodel. The additional components in the modelling environment offer functionality to the user but should be changeable and implemented outside the modelling tool. One example is the recognition component, which processes pictures of the tangible model and provides the modelling tool with the information needed to create the digital model. Therefore, the modelling tool is connected to the additional components and creates the modelling environment.

7. Conclusion and Outlook

In the context of the *Adaptive Modelling Environment*, the application was designed and prototypically implemented. However, it also analysed how it can be used during a workshop and how the digital models can be utilised afterwards.

To achieve the goal of this thesis, a design science approach was used, where the problem was first identified, and the objectives were defined by researching the literature. The design and the corresponding prototype implementation were based on a literature review, the defined objective and discussions within the OMiLAB community. Afterwards, it was demonstrated and evaluated in teaching and scientific communities, like the *OMiLAB Community of Practice*. The prototype was used in different contexts, where the evaluation was based on a discussion with tool users. Within two instantiations of the Nemo Summer School, a structured evaluation through a questionnaire was done.

The objective of this thesis was defined using the template introduced by [48] and the filled-out template defined in the introduction (see section 1.3.1) is copied here to be then used for the discussion:

- Improve **knowledge acquisition, enrichment and processing in physical workshops analysing fast evolving domains**
(problem context)
- by applying **digital twins from design thinking based on the adaptive modelling environment**
(artefact)
- that **satisfies the usage of haptic modelling in physical workshops and the automated creation of digital twins enabling the capturing and processing of the created designs**
(requirement)
- in order to **externalise, enrich, process and communicate the knowledge within the workshop and among all involved stakeholders.**
(goal)

The objective of the thesis was achieved by designing and prototypically implementing the *Adaptive Modelling Environment*, allowing us to create digital twins from design thinking. This environment supports the users in using a haptic modelling approach in workshops together with digital modelling. One key component is the automated transformation from the haptic model to the digital tone by taking and processing a picture, allowing the machine to interpret the visually rich pictures created for the human participants and transform them into a representation that can be understood by the modelling tool. The transformation is essential to ease the workload on human users and reduce the time until the digital twin is available. Recreating the models manually would reduce the readiness of the participants to use the tool, and the digital models could not be used during the workshop.

Using the *Adaptive Modelling Environment* not only influences the working during and after the workshop but also the preparation, as the haptic and the digital environment must be prepared. Based on the evaluation, combining physical and digital modelling

improves the participants' overall experience. Further, the digital model was deemed beneficial over just having a static picture.

Regarding the preparation and using it in a workshop, it was concluded that having a starting set of concepts and figures prepared for the workshop eases the workshop's beginning and flow. Starting from scratch and defining all the concepts during the workshop would increase the alignment of the models with the workshop topic. However, the practical decision to accept some restrictions based on the concepts that were prepared for the sake of the flow of the workshop seems the right way to go.

Offering users who are usually not modelling method engineers a straightforward and understandable way to adapt the metamodel increases their expressibility. This is further supported by enabling them to use and tailor pictures with visual semantics as the notation for their models, further helping them express their ideas naturally.

To answer the first research question regarding the requirements for dynamic adaptation of modelling methods and the tool during its usage, the literature on creating modelling methods was analysed, and then the prototype was used to analyse and discuss how participants use it.

It is important that one does not try to teach metamodeling to the users quickly but to translate the adaptations into the world of the users. This is not to overwhelm them, as designing sophisticated modelling methods is complex. This was done by providing them with a basic structure that they could fill out with a wizard or derive the metamodel from the instance models they made. Therefore, the resulting metamodels will not be on the same level as a sophisticated and rigorous modelling method, but they will fit into the context of a workshop. The users add information to the models, which can be reused in other models, rather than in advance thinking about all the possible concepts that may be needed.

The paragraph above also answers the second research question, which focuses on how the adaptation mechanisms can be used during the usage. There is not one answer to satisfy all users, but what is offered as a first instance is what we call *Instance Based Metamodeling* (see section 5.1.2). Here, instances of what should be available in the model can be created. Then, the metainformation can be derived by providing information to the instance model and its objects. Here, the modelling tool must not ignore concepts provided by the *Object Identification* and not yet available in the metamodel. Such concepts should be mapped to generic objects, which can then be enriched and added to the metamodel.

The third research question focuses on the internal structure of the modelling tool and the modelling methods and which parts should stay fixed or be cautiously changed so as not to create unwanted behaviour. This was tackled by creating an abstract metamodel for design thinking methods, which stays unchanged but can be used to derive different metamodels for different design thinking methods. This abstract metamodel was created by analysing various design thinking methods, creating a metamodel for them and then consolidating them to create a common structure. The functionalities of the modelling environment can then be implemented based on the abstract model.

The following research question focused on the architecture of the modelling environment

7. Conclusion and Outlook

to enable its use in and around the workshop. The *Adaptive Modelling Environment* was designed to separate the architecture into the core, which is the modelling tool and additional components which take over specific tasks that must be fulfilled. For example, one component takes a picture of the haptic model and provides it to the rest of the environment. Another component then processes the picture and identifies the object. The next component then derives relations from the picture.

The components in the architecture are not focused on a specific implementation but on the tasks defined within the thesis. This is because components can use different implementations based on the user's needs. For example, object identification can be implemented by identifying tags attached to the haptic model or using AI approaches based on CNN.

The different components are connected to the modelling tool through web services. This decoupling is needed to adapt the modelling environment to the users' needs and to support the utilisation of the best tools for the goal. Therefore, the functionality can be implemented with the programming language and libraries which are best fitting and are not limited to the technology offered by the modelling tool.

The last research question focused on how the models can be exchanged with other stakeholders if the metamodel adapts. Here, the metamodel and the models must be exchanged to allow the reuse of the model in another instance of the modelling tool. For the metamodel, this includes the available classes, characteristics, relations and other aspects if they are added. These could be constraints. For the usage in the context of the workshops, the exchange was done by allowing the modelling tool to have only loaded one metamodel, and then the models can be used with it. However, the users can easily change between the metamodels. This is done as models created in this context will be based on a project of an innovative idea, and therefore, the amount of models is limited.

Concluding, it can be said that this thesis contributes by introducing a novel approach to support haptic and digital models together, with an *Adaptive Modelling Environment* enabling the creation of synergies between them. The design of the environment and how it can be used provides a scientific contribution. The evaluation is promising that tools like this provide added value to the users. During the creation and evaluation of this thesis, topics for future work were identified. These are introduced in the next section.

7.2. Outlook

This section will provide an outlook, introducing aspects of the thesis that are suitable for future work.

One aspect for future work would be to improve the support for preparing the *Adaptive Modelling Environment* for workshops. For example, preparing the starting set of concepts, both physically and digitally, is currently done manually by the workshop organiser. Technical support will ease the workload. For example, based on a short description of the workshop's domain, an AI based approach, e.g., using large language models, can help identify important concepts that should be prepared.

Another aspect of future work could be researching new ways to utilise the digital

models during and after the workshop. During the workshop, ways for enriching the models with proposals to the participants may trigger new discussions. In this context, an interesting research topic would be how semantic constraints can be better integrated and applied during the workshops. Either provided during the preparation or generated or derived with AI. After the workshop, the transformation and linkage to other models could be further researched. Having more ways to use these models would increase the model value.

Further, another interesting future work would be implementing other design thinking methods and researching how different implementations of the design thinking methods can be linked and used together. Conceptual models have the power to make such links explicit. For example, persons used in storyboards could be further described using personas.

The ability to create digital twins from design thinking lays the foundation for novel ways to support users while creating innovative ideas. As this topic becomes increasingly essential, supporting them through physical and digital modelling can add value.

Bibliography

- [1] Design Council. The double diamond: A universally accepted depiction of the design process. <https://www.designcouncil.org.uk/our-resources/the-double-diamond/>. last-visited: 11-06-2023.
- [2] Dimitris Karagiannis and Harald Kühn. Metamodelling platforms. In *EC-Web*, volume 2455, page 182, 2002.
- [3] Jeremy Howard. Lesson 4: Practical deep learning for coders, 2016. online lecture which was last-visited: 6-7-2021.
- [4] Elena-Teodora Miron, Christian Muck, and Dimitris Karagiannis. Transforming haptic storyboards into diagrammatic models: The scene2model tool. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.
- [5] Christian Muck and Silke Palkovits-Rauter. *Conceptualizing Design Thinking Artefacts: The Scene2Model Storyboard Approach*, pages 567–587. Springer International Publishing, Cham, 2022.
- [6] Dimitris Karagiannis, Patrik Burzynski, Wilfrid Utz, and Robert Andrei Buchmann. A metamodeling approach to support the engineering of modeling method requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 199–210. IEEE, 2019.
- [7] SAP-AppHaus. Every great experience starts with a great story - scenes. <https://apphaus.sap.com/resource/scenes>. last-accessed: 1-5-2023.
- [8] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.
- [9] Tim Brown. Design thinking. *Harvard Business Review*, pages 84–92, jun 2008.
- [10] Horst WJ Rittel and Melvin M Webber. Dilemmas in a general theory of planning. *Policy sciences*, 4(2):155–169, 1973.
- [11] Richard Buchanan. Wicked problems in design thinking. *Design issues*, 8(2):5–21, 1992.
- [12] Brian W Head and John Alford. Wicked problems: Implications for public policy and management. *Administration & society*, 47(6):711–739, 2015.

Bibliography

- [13] Johann Köppel and Juliane Biehl. *Wicked Problems und Innovationen. Wie heterogene Akteure das verzwickte Feld der Energiewende bestellen*, pages 123–167. Springer Fachmedien Wiesbaden, Wiesbaden, 2023.
- [14] Alexander Grots and Margarete Pratschke. Design thinking—kreativität als methode. *Marketing Review St. Gallen*, 26(2):18–23, 2009.
- [15] Amy C Edmondson. Wicked problem solvers. *Harvard business review*, 94(6):52–9, 2016.
- [16] Simon Bell and Stephen Morse. How people use rich pictures to help them think and act. *Systemic practice and action research*, 26:331–348, 2013.
- [17] Alexander Brem. Creativity on demand: How to plan and execute successful innovation workshops. *IEEE Engineering Management Review*, 47(1):94–98, 2019.
- [18] Brainzooming. Creative Thinking: What Is the Right Size for a Brainstorming Group? <https://brainzooming.com/creative-thinking-what-is-the-right-size-for-a-brainstorming-group/21410/>, March 2014. Last-visted: 08-05-2023.
- [19] Dustin Wüest, Norbert Seyff, and Martin Glinz. Flexisketch: a lightweight sketching and metamodeling approach for end-users. *Software & Systems Modeling*, 18(2):1513–1541, 2019.
- [20] Richard F. Paige, Jonathan S. Ostroff, and Phillip J Brooke. Principles for modeling language design. *Information and Software Technology*, 42(10):665–675, 2000.
- [21] Heinrich C Mayr and Bernhard Thalheim. The triptych of conceptual modeling. *Software and Systems Modeling*, pages 1–18, 2020.
- [22] Sebastian Baltes and Stephan Diehl. Sketches and diagrams in practice. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 530–541, 2014.
- [23] François Vernadat. Enterprise modelling: Research review and outlook. *Computers in Industry*, 122:103265, 2020.
- [24] Victoria Döller. Provis - probability visualized: Ein modellierungswerkzeug für den stochastikunterricht. In Judith Michael and Dominik Bork, editors, *Modellierung 2020 Tools & Demos*, Modellierung 2020 Short, Workshop and Tools & Demo Papers, pages 222–226, February 2020.
- [25] Dominik Bork, Dimitris Karagiannis, and Igor Titus Hawryszkiewicz. Supporting customized design thinking using a metamodel-based approach. In *20th Australasian Conference on Information Systems (ACIS) 2017*, pages 1–11, December 2017.
- [26] Emanuele Laurenzi, Knut Hinkelmann, Devid Montecchiari, and Mini Goel. *Agile Visualization in Design Thinking*, pages 31–47. Springer International Publishing, Cham, 2021.

- [27] John Mylopoulos. Conceptual modelling and telos. *Conceptual modelling, databases, and CASE: An integrated view of information system development*, pages 49–68, 1992.
- [28] Dimitris Karagiannis. Agile modeling method engineering. In *Proceedings of the 19th Panhellenic Conference on Informatics*, pages 5–10. ACM, 2015.
- [29] Ulrich Frank. *Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines*, pages 133–157. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [30] Colin Atkinson, Matthias Gutheil, and Bastian Kennel. A flexible infrastructure for multilevel language engineering. *IEEE Transactions on Software Engineering*, 35(6):742–755, 2009.
- [31] Susanne Strahringer et al. *Zum Begriff des Metamodells*. Techn. Hochsch., Inst. für Betriebswirtschaftslehre, 1995.
- [32] Dimitris Karagiannis, Robert Andrei Buchmann, Patrik Burzynski, Ulrich Reimer, and Michael Walch. *Fundamental Conceptual Modeling Languages in OMiLAB*, pages 3–30. Springer International Publishing, Cham, 2016.
- [33] Thomas Kühne. Matters of (meta-) modeling. *Software & Systems Modeling*, 5(4):369–385, 2006.
- [34] Dominik Bork, Robert Buchmann, Dimitris Karagiannis, Moonkun Lee, and Elena-Teodora Miron. An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem. *Communications of the Association for Information Systems*, 34:555–579, 2018.
- [35] Ulrich Frank, Stefan Strecker, Peter Fettke, Jan Vom Brocke, Jörg Becker, and Elmar Sinz. The research field “modeling business information systems”. *Business & Information Systems Engineering*, 6(1):39–43, 2014.
- [36] Aditya Ghose, George Koliadis, and Arthur Chueng. Rapid business process discovery (r-bpd). In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *Conceptual Modeling - ER 2007*, pages 391–406, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [37] Dominik Bork and Steven Alter. Satisfying four requirements for more flexible modeling methods: Theory and test case. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 15:3–1, 2020.
- [38] Dirk Draheim, Melanie Himsl, Daniel Jabornig, Josef Küng, Werner Leithner, Peter Regner, and Thomas Wiesinger. Concept and pragmatics of an intuitive visualization-oriented metamodeling tool. *Journal of Visual Languages & Computing*, 21(3):157–170, 2010.

- [39] Katrin Hölldobler, Alexander Roth, Bernhard Rumpe, and Andreas Wortmann. Advances in modeling language engineering. In Yassine Ouhammou, Mirjana Ivanovic, Alberto Abelló, and Ladjel Bellatreche, editors, *Model and Data Engineering*, pages 3–17, Cham, 2017. Springer International Publishing.
- [40] 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.
- [41] Gabor Karsai, Holger Krah, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378*, 2014.
- [42] Igor Titus Hawryszkiewicz and Christoph Prackwieser. *MELCA—Customizing Visualizations for Design Thinking*, pages 383–396. Springer International Publishing, Cham, 2016.
- [43] Emanuele Laurenzi, Knut Hinkelmann, Stefano Izzo, Ulrich Reimer, and Alta Van der Merwe. Towards an agile and ontology-aided modeling environment for dsml adaptation. In *International Conference on Advanced Information Systems Engineering*, pages 222–234. Springer, 2018.
- [44] Raja Gumienny, Lutz Gericke, Matthias Quasthoff, Christian Willems, and Christoph Meinel. Tele-board: Enabling efficient collaboration in digital design spaces. In *Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 47–54, 2011.
- [45] Jesús J López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan De Lara. Example-driven meta-model development. *Software & Systems Modeling*, 14(4):1323–1347, 2015.
- [46] Colin Atkinson and Ralph Gerbig. Melanie: multi-level modeling and ontology engineering environment. In *Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards, MW ’12*, New York, NY, USA, 2012. Association for Computing Machinery.
- [47] Tony Clark and Ulrich Frank. Multi-level modelling with the fmmlx and the xmodelerml. *Modellierung 2020*, 2020.
- [48] Roel J Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.
- [49] Brian Henderson-Sellers and Jolita Ralyté. Situational method engineering: state-of-the-art review. *Journal of Universal Computer Science*, 2010.
- [50] Kurt Sandkuhl, Hans-Georg Fill, Stijn Hoppenbrouwers, John Krogstie, Florian Matthes, Andreas Opdahl, Gerhard Schwabe, Ömer Uludag, and Robert Winter.

- From expert discipline to common practice: a vision and research agenda for extending the reach of enterprise modeling. *Business & Information Systems Engineering*, 60(1):69–80, 2018.
- [51] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.
 - [52] Ken Peppers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.
 - [53] Iulia Vaidian, Arkadiusz Jurczuk, Zbigniew Misiak, Michael Neidow, Martin Petry, and Martin Nemetz. *Challenging Digital Innovation Through the OMiLAB Community of Practice*, pages 41–64. Springer International Publishing, Cham, 2022.
 - [54] Dimitris Karagiannis, Robert Andrei Buchmann, and Wilfrid Utz. The omilab digital innovation environment: Agile conceptual models to bridge business value with digital and physical twins for product-service systems development. *Computers in Industry*, 138:103631, 2022.
 - [55] Jan vom Brocke, Alexander Simons, Bjoern Niehaves, Bjorn Niehaves, Kai Reimer, Ralf Plattfaut, and AnneCleven. Reconstructing the giant: On the importance of rigour in documenting the literature search process. In *ECIS 2009 Proceedings*, 2009.
 - [56] Hannah Snyder. Literature review as a research methodology: An overview and guidelines. *Journal of business research*, 104:333–339, 2019.
 - [57] Tim Brown and Barry Katz. Change by design. *Journal of product innovation management*, 28(3):381–383, 2011.
 - [58] C West Churchman. Guest editorial: Wicked problems, 1967.
 - [59] Amin Mojtahedi. Running the Perfect Design Thinking Workshop: 15 Best Practices. <https://uxplanet.org/running-the-perfect-design-thinking-workshop-15-best-practices-28e97d8fb49c>, September 2019. Last visited: 14-2-2020.
 - [60] Katja Tschimmel. Design thinking as an effective toolkit for innovation. In *ISPIM Conference Proceedings*, page 1. The International Society for Professional Innovation Management (ISPIM), 2012.
 - [61] Rim Razzouk and Valerie Shute. What is design thinking and why is it important? *Review of educational research*, 82(3):330–348, 2012.
 - [62] Jeanne Liedtka. Learning to use design thinking tools for successful innovation. *Strategy & Leadership*, 39(5):13–19, 2011.
 - [63] Florian Hawlitschek. Interview with samuel tschepe on “quo vadis design thinking?”. *Business & Information Systems Engineering*, 65(2):223–228, 2023.

Bibliography

- [64] Hasso-Plattner-Institut. The six phases of the design thinking process. <https://hpi.de/en/school-of-design-thinking/design-thinking/background/design-thinking-process.html>. last-visited: 11-06-2023.
- [65] Hasso Plattner. An introduction to design thinking process guide. <https://dschool-old.stanford.edu/sandbox/groups/designresources/wiki/36873/attachments/74b3d/ModeGuideB00TCAMP2010L.pdf>, 2010. last-visited: 19-11-2018.
- [66] Alexander Osterwalder and Yves Pigneur. *Business model generation: a handbook for visionaries, game changers, and challengers*. John Wiley & Sons, 2010.
- [67] G. Lynn Shostack. How to design a service. *European journal of Marketing*, 16(1):49–63, 1993.
- [68] G. Lynn Shostack. Designing Services That Deliver. *Harvard Business Review*, January 1984.
- [69] Terry Dixon. The Service Blueprint: A powerful design tool to stay relevant with today’s fast-moving customer. <https://medium.com/sparck-io/the-service-blueprint-a-powerful-design-tool-to-stay-relevant-with-todays-fast-moving-customer-ef7d9bd89e71>, January 2018. last-visited: 17-3-2020.
- [70] Sarah Gibbons. Service Blueprints: Definition. <https://www.nngroup.com/articles/service-blueprints-definition/>, August 2017. last-visited: 17-3-2020.
- [71] Izac Ross, Lauren Chapman Ruiz, and Shahrzad Samadzadeh. Service blueprints: laying the foundation. <https://www.cooper.com/journal/2014/08/service-blueprints-laying-the-foundation/>. last-visited: 17-3-2020.
- [72] Interaction Design Foundation. Service Blueprints - Communicating the Design of Services. <https://www.interaction-design.org/literature/article/service-blueprints-communicating-the-design-of-services>, March 2020. last-visited: 17-3-2020.
- [73] Judit Kertesz. Creating a service blueprint: levels of digital and human interactions. <https://miro.com/blog/service-blueprint/>. last-visited: 17-3-2020.
- [74] Katrin Mathis. Service Blueprints – Blaupause für gute Erfahrungen. <https://katrin-mathis.de/blog/blogposts/service-blueprints-blaupause-fuer-gute-erfahrungen/>, November 2016. last-visited: 17-3-2020.
- [75] Laura Blumenthal. Your New Go-To Design Method: The Service Blueprint. <https://www.careinnovations.org/resources/your-new-go-to-design-method-the-service-blueprint/>, September 2018. last-visited: 17-3-2020.
- [76] Sara Beckman and Michael Barry. Design and innovation through storytelling. *International journal of innovation science*, 2010.

- [77] Corrie Van der Lelie. The value of storyboards in the product design process. *Personal and ubiquitous computing*, 10(2-3):159–162, 2006.
- [78] Michael Grieves and John Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. *Transdisciplinary perspectives on complex systems: New findings and approaches*, pages 85–113, 2017.
- [79] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihm. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018.
- [80] Jörg Christian Kirchhof, Judith Michael, Bernhard Rumpe, Simon Varga, and Andreas Wortmann. Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems. In *Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems*, pages 90–101, 2020.
- [81] Maulshree Singh, Evert Fuenmayor, Eoin P. Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. Digital twin: Origin to future. *Applied System Innovation*, 4(2), 2021.
- [82] Barbara Rita Barricelli, Elena Casiraghi, and Daniela Fogli. A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE Access*, 7:167653–167671, 2019.
- [83] CN Verdouw and Jan Willem Kruize. Digital twins in farm management: illustrations from the fiware accelerators smartagrifood and fractals. In Warrick Nelson, editor, *Proceedings of the 7th Asian-Australasian Conference on Precision Agriculture Digital*, Hamilton, New Zealand, pages 16–18, 2017.
- [84] Bastian Wurm, Markus Becker, Brian T Pentland, Kalle Lyytinen, Barbara Weber, Thomas Grisold, Jan Mendling, and Waldemar Kremser. Digital twins of organizations: A socio-technical view on challenges and opportunities for future research. *Communications of the Association for Information Systems*, 52:552–565, 2023.
- [85] Rashik Parmar, Aija Leiponen, and Llewellyn D.W. Thomas. Building an organizational digital twin. *Business Horizons*, 63(6):725–736, 2020.
- [86] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer Verlag, Wien, New York, 1973.
- [87] Bernhard Thalheim and Ivor Nissen. *Wissenschaft und Kunst der Modellierung: Kieler Zugang zur Definition, Nutzung und Zukunft*, volume 64. Walter de Gruyter GmbH & Co KG, 2015.
- [88] Brian Henderson-Sellers, Jolita Ralyté, Pär J Ågerfalk, and Matti Rossi. *Situational method engineering*. Springer, 2014.

Bibliography

- [89] Bernhard Thalheim. Conceptual model notions—a matter of controversy: Conceptual modelling and its lacunas. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 13:9–27, 2018.
- [90] Nicola Guarino, Giancarlo Guizzardi, and John Mylopoulos. On the philosophical foundations of conceptual models. In *Information modelling and knowledge bases XXXI*, pages 1–15. IOS Press, 2020.
- [91] Merriam-Webster Dictionary. Concept. <https://www.merriam-webster.com/dictionary/concept>. last-visited: 3-8-2021.
- [92] Stewart Robinson, Gilbert Arbez, Louis G. Birta, Andreas Tolk, and Gerd Wagner. Conceptual modeling: Definition, purpose and benefits. In *Proceedings of the 2015 Winter Simulation Conference, WSC '15*, pages 2812–2826, Piscataway, NJ, USA, 2015. IEEE Press.
- [93] Mark S. Fox and Michael Grüninger. *Ontologies for Enterprise Modelling*, pages 190–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.
- [94] Dimitrios S Kolovos, Richard F Paige, Tim Kelly, and Fiona AC Polack. Requirements for domain-specific languages. In *Proc. of ECOOP Workshop on Domain-Specific Program Development (DSPD)*, volume 2006, 2006.
- [95] Domenik Bork and Hans-Georg Fill. Formal aspects of enterprise modeling methods: a comparison framework. In *2014 47th Hawaii international conference on system sciences*, pages 3400–3409. IEEE, 2014.
- [96] Jill H Larkin and Herbert A Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive science*, 11(1):65–100, 1987.
- [97] David Harel and Bernhard Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. Technical report, RWTH Aachen University, ISR, 2000.
- [98] Bran Selic. A systematic approach to domain-specific language design using uml. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 2–9. IEEE, 2007.
- [99] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1):9–36, 1976.
- [100] Il-Yeol Song, Mary Evans, and Eun K Park. A comparative analysis of entity-relationship diagrams. *Journal of Computer and Software Engineering*, 3(4):427–459, 1995.
- [101] OMG. OMG[®] Unified Modeling Language[®]. <https://www.omg.org/spec/UML/2.5.1/PDF>, December 2017. last-visited: 25-6-2021.

- [102] OMG. Business Process Model and Notation (BPMN), Version 2.0.2, December 2013.
- [103] Dimitris Karagiannis, Heinrich C Mayr, and John Mylopoulos. *Domain-specific conceptual modeling*. Springer, 2016.
- [104] Dimitris Karagiannis, Moonkun Lee, Knut Hinkelmann, and Wilfrid Utz, editors. *Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools*. Springer Cham, 1 edition, 2022.
- [105] Ulrich Frank. Multilevel modeling. *Business & Information Systems Engineering*, 6(6):319–337, 2014.
- [106] The Open Group. Archimate® 3.1 specification. <https://pubs.opengroup.org/architecture/archimate3-doc/>. last-vistied: 02-11-2021.
- [107] Anneke Kleppe. A language description is more than a metamodel. In *Fourth international workshop on software language engineering*, volume 1. megaplanet.org, 2007.
- [108] Robert Geisler, Marcus Klar, and Claudia Pons. Dimensions and dichotomy in metamodeling. In *3rd BCS-FACS Northern Formal Methods Workshop 3*, pages 1–20, 1998.
- [109] Stefan Junginger, Harald Kühn, Robert Strobl, and Dimitris Karagiannis. Ein geschaeftsprozessmanagement-werkzeug der naechsten generation - adonis: Konzeption und anwendungen. *Wirtschaftsinformatik*, 42(5):392–401, 2000.
- [110] Ulrich Frank. Multi-perspective enterprise modeling (memo) conceptual framework and modeling languages. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 1258–1267. IEEE, 2002.
- [111] Jonathan Sprinkle, Bernhard Rumpe, Hans Vangheluwe, and Gabor Karsai. *3 Metamodelling*, pages 57–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [112] Susanne Leist, Dimitris Karagiannis, Florian Johannsen, and Hans-Gert Penzel. *From Business Engineering to Digital Engineering: The Role of Metamodeling in Digital Transformation*, pages 51–71. Springer International Publishing, Cham, 2021.
- [113] Dimitris Karagiannis and Peter Höfferer. Metamodels in action: An overview. *ICSOF*, 1:11–14, 2006.
- [114] Jean-Marc Jézéquel, Olivier Barais, and Franck Fleurey. Model driven language engineering with kermeta. In *International Summer School on Generative and Transformational Techniques in Software Engineering*, pages 201–221. Springer, 2009.

Bibliography

- [115] Srđan Živković and Dimitris Karagiannis. Towards metamodeling-in-the-large: Interface-based composition for modular metamodel development. In *Enterprise, Business-Process and Information Systems Modeling*, pages 413–428. Springer, 2015.
- [116] David Götzinger, Elena-Teodora Miron, and Franz Staffel. Omilab: An open collaborative environment for modeling method engineering. In *Domain-Specific Conceptual Modeling*, pages 55–76. Springer, 2016.
- [117] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.
- [118] Juha-Pekka Tolvanen. Metaedit+ integrated modeling and metamodeling environment for domain-specific languages. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 690–691, 2006.
- [119] Steven Kelly, Kalle Lyytinen, and Matti Rossi. Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In *International Conference on Advanced Information Systems Engineering*, pages 1–21. Springer, 1996.
- [120] Hans-Georg Fill and Dimitris Karagiannis. On the conceptualisation of modelling methods using the adoxx meta modelling platform. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 8(1):4–25, 2013.
- [121] BOC AssetManagement GmbH. The ADOxx Metamodeling Platform. <https://www.adoxx.org/>. last-visited 29-6-2021.
- [122] Andreas Dengel. *Semantische Technologien: Grundlagen. Konzepte. Anwendungen*. Springer-Verlag, 2011.
- [123] RDF Working Group. Resource Description Framework (RDF). <https://www.w3.org/rdf/>, February 2014. last-visited: 13-5-2021.
- [124] OWL Working Group. Web Ontology Language (OWL). <https://www.w3.org/OWL/>, December 2012. last-visited: 13-5-2021.
- [125] Michael Kifer, Jos de Bruijn, Harold Boley, and Dieter Fensel. A realistic architecture for the semantic web. In Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors, *Rules and Rule Markup Languages for the Semantic Web*, pages 17–29, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [126] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.
- [127] Kristen Grauman and Bastian Leibe. Visual object recognition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(2):1–181, 2011.

- [128] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International journal of computer vision*, 128(2):261–318, 2020.
- [129] Monika Bansal, Munish Kumar, and Manish Kumar. 2d object recognition techniques: State-of-the-art work. *Archives of Computational Methods in Engineering*, 28(3), 2021.
- [130] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [131] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [132] Keiron O’Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.
- [133] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Communications of the ACM*, 54(10):95–103, 2011.
- [134] Junwei Han, Dingwen Zhang, Gong Cheng, Nian Liu, and Dong Xu. Advanced deep-learning techniques for salient and category-specific object detection: a survey. *IEEE Signal Processing Magazine*, 35(1):84–100, 2018.
- [135] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. Mixed pooling for convolutional neural networks. In *International conference on rough sets and knowledge technology*, pages 364–375. Springer, 2014.
- [136] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.
- [137] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.
- [138] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [139] Jonathan Hui. Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359, March 2018. last-visted: 17-7-2020.

Bibliography

- [140] Ankit Sachan. Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN, YOLO, SSD. <https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>, N.A. last-visted: 17-7-2020.
- [141] Wei Fang, Lin Wang, and Peiming Ren. Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, 8:1935–1944, 2019.
- [142] Ibrahim Kandel and Mauro Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT express*, 6(4):312–315, 2020.
- [143] Pranoy Radhakrishnan. What are hyperparameters ? and how to tune the hyperparameters in a deep neural network? <https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a>, August 2017. last-visited: 09-08-2021.
- [144] Suki Lau. A walkthrough of convolutional neural network — hyperparameter tuning. <https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd>, July 2017. last-visited: 09-08-2021.
- [145] Alexey Bochkovskiy. CFG Parameters in the [net] section. <https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-%5Bnet%5D-section>, 2020. last-visted: 09-08-2021.
- [146] Luke Taylor and Geoff Nitschke. Improving deep learning with generic data augmentation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1542–1547. IEEE, 2018.
- [147] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. Learning data augmentation strategies for object detection. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 566–583, Cham, 2020. Springer International Publishing.
- [148] R. Plamondon and S.N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.
- [149] Yintong Wang, Wenjie Xiao, and Shuo Li. Offline handwritten text recognition using deep learning: A review. *Journal of Physics: Conference Series*, 1848(1):012015, apr 2021.
- [150] Harald Scheidl, Stefan Fiel, and Robert Sablatnig. Word beam search: A connectionist temporal classification decoding algorithm. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 253–258, 2018.
- [151] Michael Gamon. Sentiment Classification on Customer Feedback Data: Noisy Data, Large Feature Vectors, and the Role of Linguistic Analysis. In *Proceedings*

- of the 20th International Conference on Computational Linguistics, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.
- [152] Dongjoo Lee, Ok-Ran Jeong, and Sang-goo Lee. Opinion mining of customer feedback data on the web. In *Proceedings of the 2Nd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '08, pages 230–235, New York, NY, USA, 2008. ACM.
 - [153] Lin Yue, Weitong Chen, Xue Li, Wanli Zuo, and Minghao Yin. A survey of sentiment analysis in social media. *Knowledge and Information Systems*, 60(2):617–663, 2019.
 - [154] Raja Gumienny, Christoph Meinel, Lutz Gericke, Matthias Quasthoff, Peter LoBue, and Christian Willems. Tele-board: enabling efficient collaboration in digital design spaces across time and distance. *Design Thinking: Understand–Improve–Apply*, pages 147–164, 2011.
 - [155] Markus Happe, Friedhelm Meyer auf der Heide, Peter Kling, Marco Platzner, and Christian Plessl. On-the-fly computing: A novel paradigm for individualized it services. In *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, pages 1–10, 2013.
 - [156] Holger Karl, Dennis Kundisch, Friedhelm Meyer auf der Heide, and Heike Wehrheim. A case for a new it ecosystem: On-the-fly computing. *Business & Information Systems Engineering*, 62(6):467–481, 2020.
 - [157] Robert A Buchmann and Dimitris Karagiannis. Enriching linked data with semantics from domain-specific diagrammatic models. *Business & Information Systems Engineering*, 58(5):341–353, 2016.
 - [158] Chris Rupp, Matthias Simon, and Florian Hocker. Requirements engineering und management. *HMD Praxis der Wirtschaftsinformatik*, 46(3):94–103, 2009.
 - [159] Michael Walch and Dimitris Karagiannis. How to connect design thinking and cyber-physical systems: the s* iot conceptual modelling approach. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.
 - [160] Jorn Bettin, William Cook, Tony Clark, and Steven Kelly. Knowledge industry survival strategy (kiss) fundamental principles and interoperability requirements for domain specific modeling languages. In *Proceedings of the 24th ACM SIG-PLAN conference companion on Object oriented programming systems languages and applications*, pages 709–710, 2009.
 - [161] Mark Strembeck and Uwe Zdun. An approach for the systematic development of domain-specific languages. *Software: Practice and Experience*, 39(15):1253–1292, 2009.

- [162] Bart Meyers and Hans Vangheluwe. A framework for evolution of modelling languages. *Science of Computer Programming*, 76(12):1223–1246, 2011. Special Issue on Software Evolution, Adaptability and Variability.
- [163] Colin Atkinson, Ralph Gerbig, and Mathias Fritzsche. A multi-level approach to modeling language extension in the enterprise systems domain. *Information Systems*, 54:289–307, 2015.
- [164] Richard Braun and Werner Esswein. Towards an integrated method for the extension of mof-based modeling languages. In Ladjel Bellatreche and Yannis Manolopoulos, editors, *Model and Data Engineering*, pages 103–115, Cham, 2015. Springer International Publishing.
- [165] Bran Selic. *The Theory and Practice of Modeling Language Design for Model-Based Software Engineering—A Personal Perspective*, pages 290–321. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [166] Afef Awadid, Dominik Bork, and Selmin Nurcan. Towards assessing the multi-view modeling capability of enterprise modeling methods. In *IFIP Working Conference on The Practice of Enterprise Modeling*, pages 351–361. Springer, 2018.
- [167] Dominik Bork, Robert Buchmann, and Dimitris Karagiannis. Preserving multi-view consistency in diagrammatic knowledge representation. In *International conference on knowledge science, engineering and management*, pages 177–182. Springer, 2015.
- [168] Hyun Cho and Jeff Gray. Design patterns for metamodels. In *Proceedings of the Compilation of the Co-Located Workshops on DSM’11, TMC’11, AGERE! 2011, AOOPEs’11, NEAT’11, & VMIL’11, SPLASH ’11 Workshops*, page 25–32, New York, NY, USA, 2011. Association for Computing Machinery.
- [169] Ana-Maria Ghiran, Robert Andrei Buchmann, and Dimitris Karagiannis. Towards a framework of techniques for enabling semantics-driven secondary notation in conceptual models. In *2018 12th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–6, 2018.
- [170] Dominik Bork and Ben Roelens. A technique for evaluating and improving the semantic transparency of modeling language notations. *Software and Systems Modeling*, 20(4):939–963, 2021.
- [171] Harald Störrle. On the impact of size to the understanding of uml diagrams. *Software & Systems Modeling*, 17(1):115–134, 2018.
- [172] Conrad Bock and Maged Elaasar. Reusing metamodels and notation with diagram definition. *Software & Systems Modeling*, 17(4):1079–1103, 2018.
- [173] Bernhard Hoisl, Stefan Sobernig, and Mark Strembeck. Reusable and generic design decisions for developing uml-based domain-specific languages. *Information and Software Technology*, 92:49–74, 2017.

- [174] Alexander Bergmayr. Reuseme-towards aspect-driven reuse in modelling method development. In *International Conference on Model Driven Engineering Languages and Systems*, pages 4–18. Springer, 2010.
- [175] Ulrich Frank. Prolegomena of a multi-level modeling method illustrated with the fmmlx. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 521–530, 2021.
- [176] Bernhard Hoisl and Stefan Sobernig. Open-source development tools for domain-specific modeling: Results from a systematic literature review. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 5001–5010, 2016.
- [177] Michael Walch and Dimitris Karagiannis. Service-driven enrichment for kbr in the omilab environment. In *International Conference on Serviceology*, pages 164–177. Springer, 2017.
- [178] Michael Walch. Operating cyber-physical systems with microservices: the s* iot conceptual modelling approach. In *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 787–792. IEEE, 2018.
- [179] Stefan Strecker, Ulrike Baumöl, Dimitris Karagiannis, Agnes Koschmider, Monique Snoeck, and Rüdiger Zarnekow. Five inspiring course (re-) designs. *Business & Information Systems Engineering*, 61(2):241–252, 2019.
- [180] Niksa Visic, Hans-Georg Fill, Robert Andrei Buchmann, and Dimitris Karagiannis. A domain-specific language for modeling method definition: From requirements to grammar. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pages 286–297, 2015.
- [181] Sybren de Kinderen, Monika Kaczmarek-Heß, Qin Ma, and Iván S. Razo-Zapata. Towards meta model provenance: A goal-driven approach to document the provenance of meta models. In Geert Poels, Frederik Gailly, Estefania Serral Asensio, and Monique Snoeck, editors, *The Practice of Enterprise Modeling*, pages 49–64, Cham, 2017. Springer International Publishing.
- [182] Marc Stickdorn, Adam Lawrence, Markus Edgar Hormess, and Jakob Schneider. *This Is Service Design Doing: Applying Service Design Thinking in the Real World : A Practitioners’ Handbook*. O’Reilly, 2018.
- [183] Falk Uebernickel, Walter Brenner, Britta Pukall, Therese Naef, and Bernhard Schindlholzer. *Design Thinking: Das Handbuch*. Frankfurter Allgemeine Buch, 2015.
- [184] Per Kristiansen and Robert Rasmussen. *Building a better business using the Lego serious play method*. John Wiley & Sons, 2014.
- [185] Kaoru Ishikawa and Kaoru Ishikawa. *Guide to quality control*, volume 2. Asian Productivity Organization Tokyo, 1982.

Bibliography

- [186] Alexander Osterwalder, Yves Pigneur, Gregory Bernarda, and Alan Smith. *Value proposition design: How to create products and services customers want*. John Wiley & Sons, 2015.
- [187] Nikola Babic. From text to conceptual model : a text understanding transformation approach. mathesis, University of Vienna, Universitätsring 1, 1010 Wien, 2023.
- [188] Florian Schierlinger-Brandmayr, Birgit Moesl, Philipp Url, Wolfgang Vorraber, and Siegfried Vössner. *A Modeling Tool for Exploring Business Ecosystems in a (Pre-)conceptual Phase*, pages 315–338. Springer International Publishing, Cham, 2022.
- [189] Neli Petkova. Scene2model framework : relation prediction using logical tensor networks. mathesis, University of Vienna, Wien, 2023.
- [190] Giacomo Frisoni, Gianluca Moro, and Antonella Carbonaro. Unsupervised descriptive text mining for knowledge graph learning. In *KDIR*, pages 316–324, 2020.
- [191] Fatima N Al-Aswadi, Huah Yong Chan, and Keng Hoon Gan. Automatic ontology construction from text: a review from shallow to deep learning trend. *Artificial Intelligence Review*, 53:3901–3928, 2020.
- [192] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic notes in theoretical computer science*, 152:125–142, 2006.
- [193] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of computer programming*, 72(1-2):31–39, 2008.
- [194] Elena-Teodora Miron, Christian Muck, Dimitris Karagiannis, and David Götzinger. Transforming storyboards into diagrammatic models. In Peter Chapman, Gem Stapleton, Amirouche Moktefi, Sarah Perez-Kriz, and Francesco Bellucci, editors, *Diagrammatic Representation and Inference*, pages 770–773, Cham, 2018. Springer International Publishing.
- [195] Christian Muck, Elena-Teodora Miron, Dimitris Karagiannis, and Lee Moonkun. Supporting service design with storyboards and diagrammatic models: The scene2model tool. In *Joint International Conference of Service Science and Innovation (ICSSI 2018) and Serviceology (ICServ 2018)*, September 2018.
- [196] Alexander Völz, Christian Muck, Danial M. Amlashi, and Dimitris Karagiannis. Bridging haptic design thinking and cyber-physical environments through digital twins using conceptual modeling. In *BIR-WS 2023: BIR 2023 Workshops and Doctoral Consortium, 22nd International Conference on Perspectives in Business Informatics Research (BIR 2023)*. CEUR Workshop Proceedings (CEUR-WS.org), September 2023.

Acronyms

AI Artificial Intelligence.

AMME Agile Modeling Method Engineering.

ANN artificial neural network.

API applicaiton programming interface.

BMC Business Model Canvas.

BPMN Busines Process Model and Notation.

CNN convolutional neural network.

CPS Cyber-Physical system.

CV Computer Vision.

DIEn Digital Innovation Environment.

EM Enterprise Modelling.

ER Entity-Relationship.

Faster R-CNN Faster Region Based Convolutional Neural Networks.

GPU graphical processing unit.

GUI graphical user interface.

HCI human-computer interaction.

HTTP Hypertext Transfer Protocol.

IoT Internet of Things.

IS information system.

IT information technology.

Acronyms

JSON JavaScript Object Notation.

MEMO multi-perspective enterprise modelling.

NLP Natural Language Processing.

OCR Optical Character Recognition.

OMG Object Management Group®.

OOP object-oriented programming.

OWL Web Ontology Language.

PCA Principal Components Analysis.

RDF Resource Description Framework.

RDFS RDF Schema.

REST Representational State Transfer.

s2m Scene2Model.

SME Situational Method Engineering.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

XML Extensible Markup Language.

YOLO You Only Look Ones.

A. Appendix

A.1. Knowledge Base

This section will contain additional materials for the knowledge base that can enhance information from image recognition.

A.1.1. Assignment of Abstraction Layer to Recognised Objects with RDF

This section contains an example in the form of snippets for a RDF knowledge base, which supports an automatic assignment of the abstraction layers for the recognised objects. This is an excerpt for a knowledge base on how it can be used in the transformation algorithm introduced in section 5.4.1.1.

The code in Listing A.1 shows a snippet for the possible knowledge base in Turtle syntax. It uses RDFS and OWL, which are introduced in section 2.4.1.

In example the lines 21 to 25 (Listing A.1) show the RDF implementation of the rules introduced section 5.4.1.1 (rule 5.1 and rule 5.2). First, the classes for the abstraction layers are introduced, and then the examples for the domain objects are specified.

```
1      @prefix : <http://www.omilab.org/scene2model/concepts#> .
2      @prefix owl: <http://www.w3.org/2002/07/owl#> .
3      @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4      @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5      @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
6      @base <http://www.omilab.org/s2m/concepts> .
7
8      # -----
9      # Object Abstraction Layer class definition
10     # -----
11     :GenericObject a :NetAbstractionLayer;
12     :isAbstractionLevel 2.
13     :ClassObject a :NetAbstractionLayer;
14     :isAbstractionLevel 1.
15     :ConcreteObject a :NetAbstractionLayer;
16     :isAbstractionLevel 0.
17
18     :ClassObject owl:onProperty rdfs:subClassOf;
19     owl:someValuesFrom :GenericObject.
20
21     :ConcreteObject owl:onProperty rdfs:subClassOf;
22     owl:someValuesFrom :ClassObject.
23
24     # -----
25     # Object Abstraction Layer Structure
```

A. Appendix

```
26      # -----
27
28      # generic object layer
29      :Object a :GenericObject.
30
31      # class object layer
32      :Person rdfs:subClassOf :Object.
33
34      :Car    rdfs:subClassOf :Object.
35
36      # concrete object layer
37      :Accountant rdfs:subClassOf :Person;
38      rdfs:label "Accountant".
39
40      :Engineer rdfs:subClassOf :Person;
41      rdfs:label "Engineer".
42
43      :Bus      rdfs:subClassOf :Car;
44      rdfs:label "Bus".
45
46      :Convertible rdfs:subClassOf :Car;
47      rdfs:label "Convertible".
```

Listing A.1: RDF knowledge base example for the abstraction layers of recognised Objects (example based on Figure 5.15)

Then, the queries defined in Listing A.2 and Listing A.3 can be used to query the abstraction layer of the provided query. To adapt the query, the concept *:Person* or *:Accountant* must be replaced with the concept for which the abstraction layer should be received.

```
1 PREFIX : <http://www.omilab.org/scene2model/concepts#>
2 select (MIN (?layerNum) AS ?abstractionLayer ) where {
3     :Accountant a ?hierarchyLayer .
4     ?hierarchyLayer a :NetAbstractionLayer.
5     ?hierarchyLayer :isAbstractionLevel ?layerNum.
6 }
```

Listing A.2: SPARQL query to get the abstraction layer of a concrete object

```
1 PREFIX : <http://www.omilab.org/scene2model/concepts#>
2 select (MIN (?layerNum) AS ?abstractionLayer ) where {
3     :Person a ?hierarchyLayer .
4     ?hierarchyLayer a :NetAbstractionLayer.
5     ?hierarchyLayer :isAbstractionLevel ?layerNum.
6 }
```

Listing A.3: SPARQL query to get the abstraction layer of a class

The queries will return the following results if applied to the knowledge base defined in Listing A.1:

- for Listing A.2: 0
- for Listing A.3: 1

A.1.2. Assignment of Domain Tags to Recognised Object with RDF

This subsection provides an example of how RDF can be used to assign domain tags to concepts, as discussed in section 5.4.1.1. It implements rules provided in formula 5.3, 5.5 and 5.6. It uses the example in the knowledge base of Listing 5.3. The used standard concepts are explained in section 2.4.1.

The rule of formula 5.3 (assigning the *DomainTag* class based on the *hasDomainTag* property) is specified by line 6. It uses the standard RDFS concept of *rdfs:range*. The rule of formula 5.5 (inherit domain tag) is defined in line 7, using the standard OWL concept of *owl:propertyChainAxiom*.

The rule of formula 5.6 is defined using the *TriG* syntax and uses the definition of graphs (see line 12-16 of Listing A.4) within the knowledge base. This graph is then used within the query (see Listing A.6) to get all the domain tags assigned to the group of statements.

```

1  @prefix : <http://www.omilab.org/scene2model/concepts#> .
2  @prefix owl: <http://www.w3.org/2002/07/owl#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4  @base <http://www.omilab.org/s2m/elements> .
5
6  :hasDomainTag rdfs:range :DomainTag.
7
8  :hasDomainTag owl:propertyChainAxiom (rdfs:subClassOf :hasDomainTag).
9
10 :workshop1 :hasDomainTag :publicTransportation.
11
12 :workshop1{
13   :Engineer rdfs:subClassOf :Person.
14   :Engineer :hasDomainTag :production.
15   :Person :hasDomainTag :smartCity
16 }

```

Listing A.4: RDF knowledge base example for the assignment of domain tags to concepts (example based on Figure 5.15)

```

1  PREFIX : <http://www.omilab.org/scene2model/concepts#>
2  select * where {
3    ?s a :DomainTag .
4  }

```

Listing A.5: SPARQL query to get all domain tags

The query defined in Listing A.5 queries all the used domain tags in the knowledge base and returns (results assume the standard prefixes defined in the knowledge base of Listing A.4):

- :publicTransportation
- :production
- :smartCity

A. Appendix

```
1      # get all domains of :Engineer
2      PREFIX : <http://www.omilab.org/scene2model/concepts#>
3      select distinct ?x where {
4          BIND(:Engineer as ?subject)
5          {{GRAPH ?g {
6              ?subject ?a ?b.
7          }}
8          UNION
9          {GRAPH ?g {
10             ?a ?b ?subject
11          }}
12          UNION
13          {GRAPH ?g {
14             ?a ?subject ?b
15          }}
16          ?g :hasDomainTag ?x}
17      UNION {?subject :hasDomainTag ?x}
18  }
```

Listing A.6: SPARQL query to get all domain tags of *Engineer*

The query defined in Listing A.6 queries all the domain tags assigned to the *engineer* within the knowledge base and returns (results assume the standard prefixes defined in the knowledge base of Listing A.4):

- :publicTransportation
- :production
- :smartCity

A.2. A Metamodel for Design Thinking Techniques

This section contains the visualisation of the metamodel, introduced in section 5.2.2 in landscape format to be more readable.

A.3. A Implementation-Oriented Metamodel for *Adaptive Modelling Environment*

This section contains the visualisation of the implementation-oriented metamodel, introduced in section 5.2.3 in landscape format so that it is more readable.

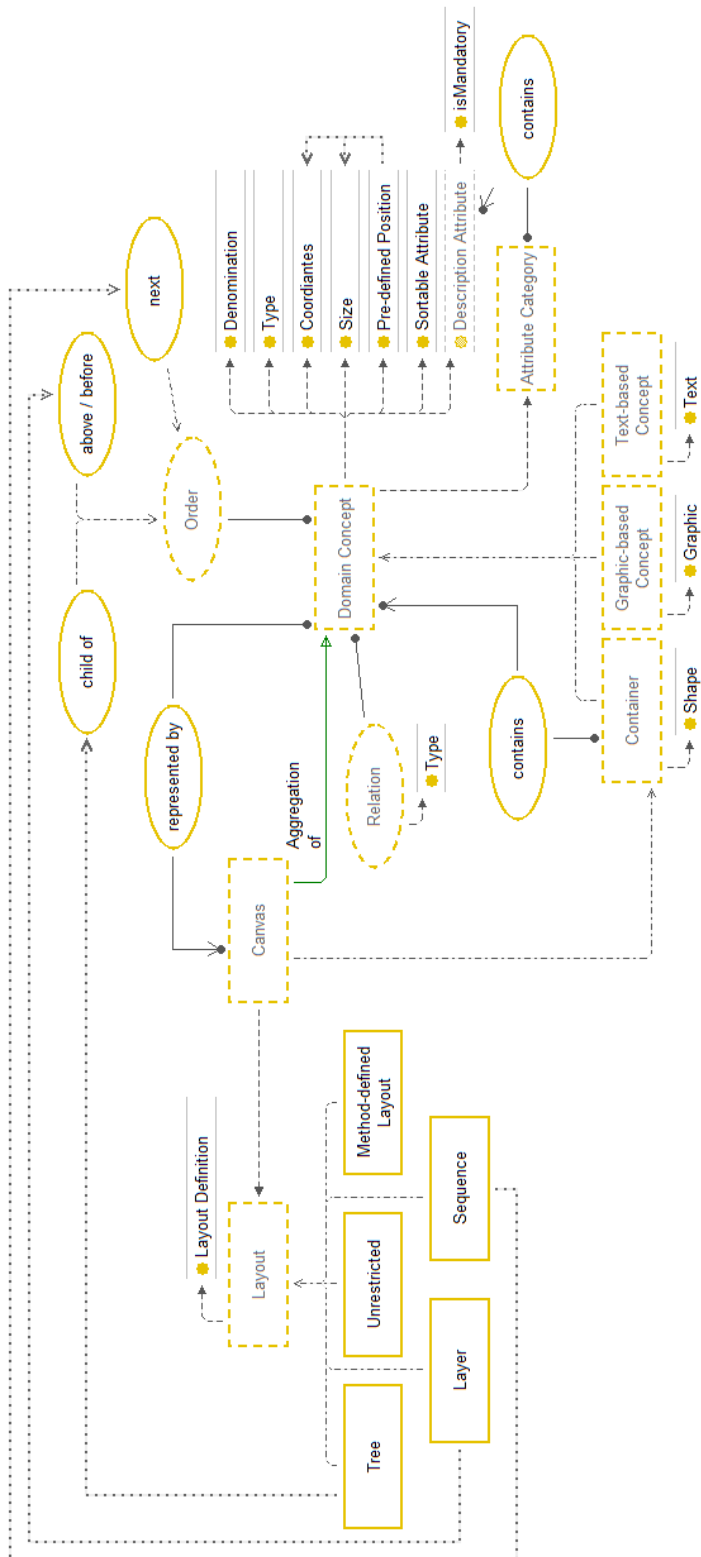


Figure A.1.: Abstract metamodel for design thinking techniques in landscape format

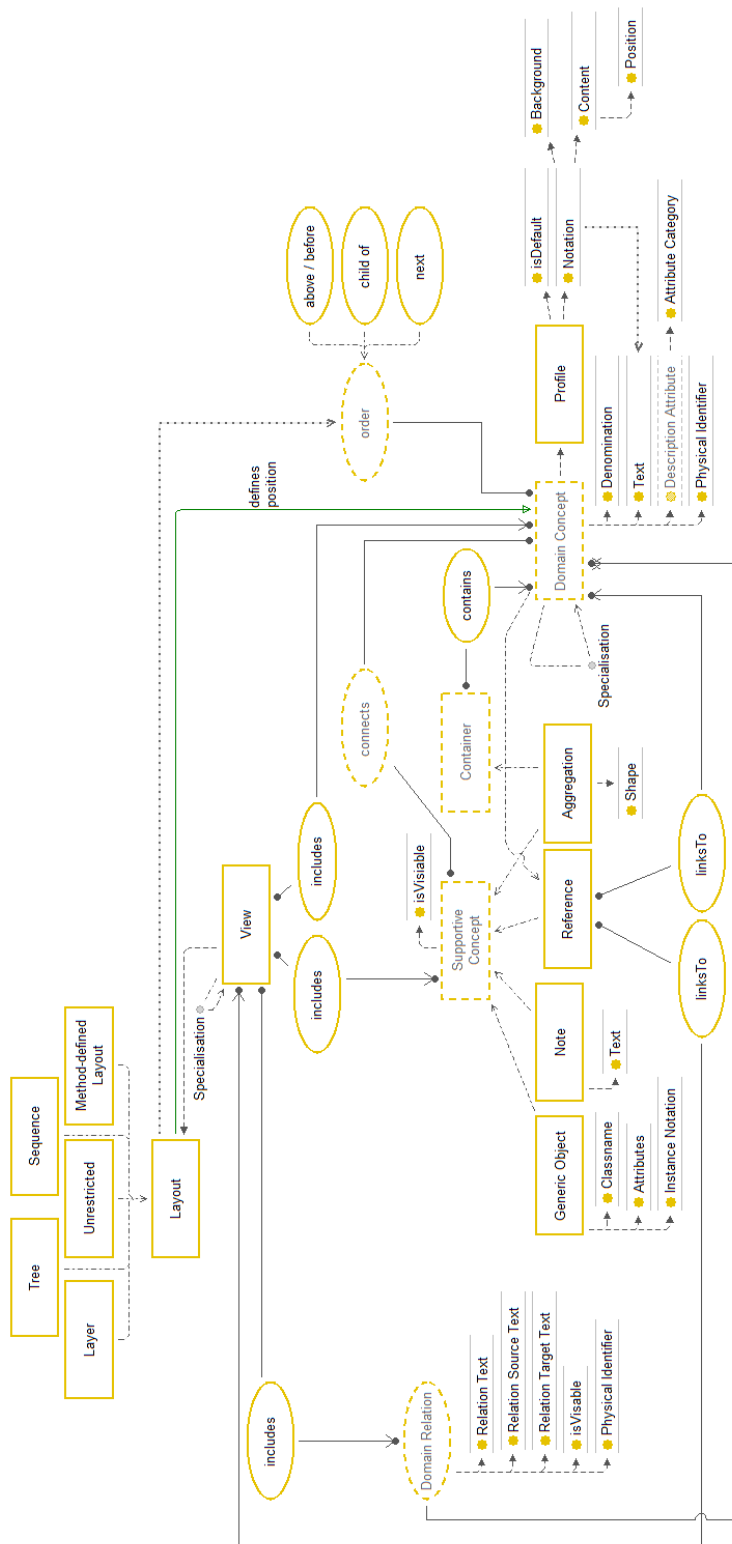


Figure A.2.: Implementation-oriented metamodel for *Adaptive Modelling Environment* in landscape format

A.4. Used Technologies and Methods

A.4.1. ADOxx

ADOxx¹ is a metamodeling platform allowing to create modelling tools, by specifying the metamodel. It is used to implement diagrammatic modelling methods. Additionally, it offers the capability to add functionality by using a build-in scripting language called AdoScript. Therefore, with the resulting modelling tool, one can create diagrams and use mechanisms and algorithms on the created models. Having the ability to create modelling tools for using a modelling language and offering ways to implement functionality is an essential aspect of metamodeling platforms [2].

ADOxx has a stand-alone implementation, meaning that the platform is installed on a device and can be used over its GUI. It offers two toolkits, one for developing the modelling tool and one for using it. As soon as metamodel is defined in the developing toolkit, it can be used in the modelling toolkit. For finished methods, it is also possible to package the modelling method into an installer, which enables a stand-alone installation of a concrete modelling tool without the development toolkit.

AdoScript is ADOxx's scripting language, allowing, among other things, the processing or creation of models. This includes but is not limited to creating/deleting objects and relations, setting attribute values, gathering information from the models and transforming them into another model or data structure, and so on. It also can integrate other applications by making HTTP calls or executing terminal commands. Further, it allows users to read and save data as files on the device. Last but not least, AdoScript can be used to change the metamodel of the modelling tool.

A.4.2. CoreNLP

CoreNLP² is a Java library for natural language processing (NLP). It offers multiple functionalities that are important during NLP and supports different languages. The library's core is the pipeline, combining different processing steps and producing the final result.

For this thesis, we used the sentiment analysis capabilities to annotate comments made on social media with their sentiment. In this way, the comments are grouped into positive, negative, and neutral categories.

A.4.3. DarkMark

DarkMark³ is a stand-alone application allowing to create picture annotation for CNN that should be trained with DarkNet (see appendix A.4.4). It offers a GUI for labelling the picture by allowing it to draw a bounding box over the object that should be recognised and assign it a label. Further, it supports creating the configuration files for training the

¹<https://www.adoxx.org/> (accessed: 01-06-2024)

²<https://stanfordnlp.github.io/CoreNLP/> (accessed 31-05-2024)

³<https://www.coderun.ca/darkmark/> (accessed: 01-06-2024)

A. Appendix

CNN. Last but not least, DarkMark can also apply some data augmentation algorithms, improving the trained CNN

A.4.4. Darknet

Darknet⁴ is a framework to train neural networks. It is written in C and allows the training of neural networks on GPUs, improving the training performance. The framework offers functionality to define the architecture and configuration of neural networks and run the training, including convolutional neural network. The results are three files. The most important is the trained net. The file with the configuration and the file defining the labels are also needed to use the trained net.

Based on the original implementation, different forks were created supporting different operating systems. The darknet framework also offered pre-defined CNN architectures. For more information on the one we used for the prototype, see appendix A.4.12.

A.4.5. EasyOCR

EasyOCR⁵ is a Python library focusing on Optical Character Recognition (OCR). With this library, text in pictures can be recognised and returned as strings that can be used in the code.

The prototype recognises text written on the paper figures and makes it accessible to the modelling tool.

A.4.6. FFmpeg

FFmpeg⁶ is a framework for working with multimedia content. For example, it can be used for decoding, encoding, streaming or filtering. It can be used on multiple operating systems and is controllable via the command line.

Among others, we use the framework in the provider component for the prototype. It opens the USB camera and sends the pictures to the other components. In this way, the picture can be accessed by the Scene2Model prototype as input for transforming the physical figures into digital models.

A.4.7. GraphDB

*GraphDB*⁷ is a graph database compliant to the W3C standard. Therefore, it allows to store and query RDF triplets. It also includes a reasoner to derive additional knowledge. The reasoner can be used with the standard RDF ontologies (e.g., OWL) or user-defined rules can be created.

⁴<https://github.com/pjreddie/darknet> (accessed: 01-06-2024)

⁵<https://github.com/JaidedAI/EasyOCR> (accessed: 01-06-2024)

⁶<https://ffmpeg.org/> (accessed 01-06-2024)

⁷<https://graphdb.ontotext.com/> (accessed: 01-06-2024)

A.4.8. imgaug

The Python library *imgaug*⁸ offers diverse functions with which pictures can be augmented. This means that their appearance is changed. For example, the focus or the size is adapted, parts are cut out, or the colour is changed.

In the context of image recognition with CNN, this is useful as it artificially increases the training data by creating new ones out of the existing ones. During the augmentations are applied, the library also recalculates the bounding boxes of the figures that should be recognised. This is important so that the created pictures must not be annotated manually.

A.4.9. lxml

The Python library *lxml*⁹ can be used to manipulate XML files. This thesis uses the library to transform the XML files of two modelling tools created with ADOxx. The library offers functionality to find the needed parts in the file and replace them according to the defined transformation rules.

A.4.10. Olive Microservice Framework

The *Olive*¹⁰ microservice framework supports the users in creating configurable microservices. These can then be easily instantiated for concrete problems.

A web interface is available for configuration. This allows the service adaptation for different workshops by keeping the interface the same. Therefore, in the modelling tool, only the configuration for the endpoint must be added, and then the functionality can be used.

Instantiated Olive microservices offer JSON endpoints, which can be called via HTTP. The return is a JSON file, which can be further processed.

A.4.11. Python opencv

OpenCV¹¹ is a famous library for computer vision tasks. It offers over 2.500 algorithms for processing pictures in applications. For OpenCV also, a Python library exists, which is called *opencv-python*¹². This enables the manipulation of pictures and the integration of video streams over the internet or from locally connected USB cameras. For example, this was used for one prototype of the image provider component.

This library also offers to use Yolo networks (see <https://opencv-tutorial.readthedocs.io/en/latest/yolo/yolo.html>, accessed 01-06-2024) trained with Darknet directly in the Python code.

⁸<https://imgaug.readthedocs.io/en/latest/> (accessed: 01-04-2024)

⁹<https://lxml.de/> (accessed: 01-06-2024)

¹⁰<https://www.adoxx.org/live/olive> (accessed: 01-06-2024)

¹¹<https://opencv.org/> (accessed: 01-06-2024)

¹²<https://github.com/opencv/opencv-python> (accessed: 01-06-2024)

A. Appendix

Last but not least, ArUco¹³ is also available in the opencv-python library. ArUco contains algorithms for finding pre-defined markers in pictures. For example, this is used in computer vision applications like robot navigation or augmented reality. These pre-defined markers can be easily and efficiently found in pictures. These markers can also be created with the library, printed, and cut out. For the Scene2Model prototype, this marker were used as the tags, which were attached to the paper figures and then recognised for one approach for the object identification.

A.4.12. Yolo

Yolo (You Only Look Once) is a CNN architecture which fast in training and recognising objects in pictures [137]. Different versions of the architecture improve certain aspects like training performance or accuracy. There exists a version named with *tiny* or *tinier*, which reduces the size of the network and the size of the files by keeping a good accuracy [141]. In this thesis, we used *Yolov4-tiny*, which provided good results and performed well by small figures in the picture.

A.4.13. Designing Metamodels with CoChaCo

CoChaCo is a modelling method to create and design modelling methods. [6] specifies and discusses how CoChaCo can be used to capture the requirements for a modelling language or even method. The focus is usually on domain-specific language, which uses concepts from a specific domain and directly uses them in the language itself.

CoChaCo supports defining the requirements of a modelling method and its design. This means it can describe the meta-model and the *model language*, the *modelling procedure* and also hint of the offered *mechanisms & algorithms*. It thereby captures the three main parts of a modelling method as introduced in [2].

The foundation of CoChaCo is based on using three terms to define the modelling methods: **C**oncept, **C**haracteristic and **C**onector. More concepts are available in the method, but those three form the foundation. The modelling tool can be downloaded at <https://www.omilab.org/activities/cochaco/> (accessed: 01-06-2024).

An overview of the important classes and relations can be seen in Figure A.3. The here important classes and relations are:

- **Concepts** represent the classes that should be available in the modelling method.
- **Characteristic** represent the attributes a class or connector should have.
- A **Connector** represents possible relations between concepts.
- **Purpose** is used to represent the goals.
- **Functionality** represents the modelling method's processing capabilities.
- **Connects** is a relation symbolizing that the objects have a relation to each other.

¹³https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (accessed: 01-06-2024)

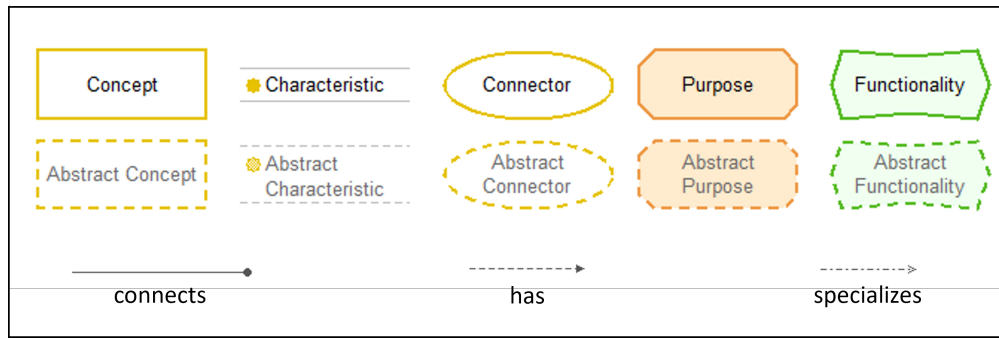


Figure A.3.: Overview of the important classes and relations offered by the CoChaCo modelling method

- **Has** is a relation symbolizing that one object possesses another object.
- **Specializes** is a relation symbolizing that one object is a specialized form of another, representing inheritance structures.

A.5. Workshop Materials and Evaluation

This section contains materials used for the workshop-based evaluation of this thesis. For example, it contains texts of pre-defined scenarios or questionnaire examples. Additionally, result data not directly used in the evaluation chapter is listed here, e.g., the answers list of the open questions.

A.5.1. Pre-Defined Smart Mobility Scenario

A.5.1.1. Introduction

For many people, traveling is part of their jobs. Planning the different phases of a journey can become a time-consuming and expensive task. To ease this problem as a service provider it is very important to understand the problems, which the target group is really facing.

For example, take Sabine. She is 42 years old and is head of a department of a Vienna company. She recently had to visit a business partner in Germany and is arriving back in Vienna with her plane. Because the meeting with her business partner lasted longer than planned she didn't had time to plan her trip home. Further, unfortunately she received an E-Mail from her colleague in Vienna, that they need her in the office too.

A.5.1.2. Possible problems on the journey

Upon her arrival at the airport, Sabine doesn't know which public transportation services she must take for getting to her office. After she found a possible route, she has no idea if additional tickets are needed or if her annual season-ticket for the city of Vienna is

A. Appendix

sufficient. She finds a map of the different fare zones but needs some time to understand it. It turned out that she needs an additional ticket and went to the next ticket machine, which was out of order. After she gathered all the information of the route, was able to buy the ticket at another machine and arrived at the platform her train was leaving right in front of her eyes.

During the train ride Sabine decides that the way from the destination station to the office is too long for walking with her suitcase. Therefore, she starts to look for a taxi upon the arrival on the destination station. At the nearest taxi stand, no free one was available. Therefore, she starts walking towards her office instead, carrying her suitcase. Half way through she is able to catch a taxi but as she is looking at all the other cars on the road her environmentally friendly thinking starts and remorse about taking a car alone starts to grow. As she walks to the entrance of her office building, the meeting has already begun.

After the meeting, her hunger starts to grow but she needs another half an hour to take some notes at her office desk. As she is getting appetite for a burger she can't remember if all ingredients are available at home. After this long day, she decides to take her company car for driving home. The company cars are parked in a separate garage building, where she has to search for her car.

Upon the arrival at her home, she looks up the available ingredients and hopes that she is able to prepare the desired burger. Unfortunately, she must prepare herself another burger, because some are missing. It was getting really late as the burger was finished. Therefore, she ate quickly and ended the day exhausted.

A.5.1.3. One solution scenario

Upon her arrival at the airport she takes her smart phone, opens her travel app and searches for a route to her workplace. The app knows which ticket Sabine already possesses and which one are still needed. She buys the additional ticket with one click in the app. Then goes to the train station and waits for her train.

During the train ride Sabine decides that the way from the destination station to the office is too long for walking with her suitcase. Because of her environmentally friendly attitude, she doesn't want to take a taxi for her alone. Therefore, she opens a travel app and orders a car pool from the destination station to her office. The app shows different possibilities from which she picks one. As she walks to the entrance of her office building, she has some time left to prepare for the meeting.

After the meeting, she goes to her desk in the office building to take notes of the meeting. She estimates that this will take a half hour and afterwards she will go home. She opens the company's app again and orders her company car, so that it will wait for her at the entrance in half an hour.

Because she will arrive home late in the evening, she opens her smart home app and looks up the ingredients available at her home. She decides that she wants a burger, which she orders in the smart home app and enters her arrival time. A robot arm will finish the burger shortly after she arrives at home. She now can eat calmly and is able to relax before her day ends.

A.5.2. Questionnaire for the Second Workshop Evaluation

Scene2Model Questionnaire

Thank you very much for participating in this survey. To start things off, please fill out the following questions about your person:

1. Personal information:

Markieren Sie nur ein Oval pro Zeile.

	18-24	25-34	35-44	>45
Age range	☐	☐	☐	☐

2. Academic field:

3. Occupation

4. How knowledgeable are you in the topic of design thinking?

Markieren Sie nur ein Oval.

	1	2	3	4	5
Not ☐ ☐ ☐ ☐ ☐ Very much					

5. How knowledgeable are you in the topic of conceptual modelling?

Markieren Sie nur ein Oval.

	1	2	3	4	5
Not ☐ ☐ ☐ ☐ ☐ Very much					

In the following, please tell us about your experience with Scene2Model gained during the guided workshop on Friday, 15.7.22, and the group-internal workshops in the second week and provide us with your valuable feedback. Please, answer the following questions about the Scene2Model tool as accurate as possible.

The first section of questions focuses on the general idea of combining tangible figures with digital models and adapting the concepts to your need.

In the second section, please compare the two variants of the Scene2Model tool by first evaluating the variant of the Scene2Model tool where the figures were provided to you (Friday). Afterwards, answer the same questions about the variant of the tool where you had to define the figures on your own.

First Section:

Please rate the following statements on a scale from 1 to 5.

6. Reflecting on the concepts that were needed during the working session had a positive influence on the created idea/design.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

7. Adapting the available figures (paper and digital) to the specific case improved the visualization and comprehensibility of the created story.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

8. The added value from adapting the figure outweighs the additional effort.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

9. The digital model outperforms a photograph in capturing the knowledge collected in the workshop.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

10. Combining paper figures and digital models improves the overall experience.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

11. The tangible paper figures foster creativity and cooperation in the workshops.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

12. The tangible paper figures foster creativity and cooperation in the workshops.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

Second Section (A):

Please rate the following statements on a scale from 1 to 5 in regards to the Scene2Model tool where the figures were provided to you.

13. I was able to represent all the notational concepts that I wanted to represent.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

14. The existing concepts in the tool were not sufficient for modeling the desired scenario(s).

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

15. It was easily possible to add additional concepts in the tool according to my needs.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

16. Based on the time I spent using the tool, I would require more time to feel familiar with its use.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

Second Section (B):

Please rate the following statements on a scale from 1 to 5 in regards to the Scene2Model tool where you had to define the figures on your own.

17. I was able to represent all the notational concepts that I wanted to represent.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

18. The existing concepts in the tool were not sufficient for modeling the desired scenario(s).

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

19. It was easily possible to add additional concepts in the tool according to my needs.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

20. Based on the time I spent using the tool, I would require more time to feel familiar with its use.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

Scene2Model Tutorial Videos

Please let us know how helpful the Scene2Model tutorials were.

(https://www.omilab.org/activities/projects/details/?id=131&path=DOCUMENTATION%2F03-Usage%20Guides#page_title)

21. These instructional videos helped me when using Scene2Model.

Markieren Sie nur ein Oval.

1 2 3 4 5

Stro ☐ ☐ ☐ ☐ ☐ Strongly Agree

22. How many times did you watch the videos?

Markieren Sie nur ein Oval.

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 3+

23. Overall I feel I was able to learn the information from these instructional videos as well as I would have in a face-to-face class presentation.

Markieren Sie nur ein Oval.

- ☐ Yes
- ☐ No

24. Do you have any feedback regarding the tutorial videos?

Additional Statements

Please use the following questions to tell us your thoughts about the Scene2Model tool.

25. What did you like about the Scene2Model tool?

26. What did you not like about the Scene2Model tool?

27. Do you have any other comments about the Scene2Model tool?

Dieser Inhalt wurde nicht von Google erstellt und wird von Google auch nicht unterstützt.

Google

A.5.3. Questionnaire for the Second Workshop: Open Question Answers

In this section, the list of answers provided for the open questions of the evaluation with the second workshop is provided. For each open question, one list is provided.

What did you like about the Scene2Model tool? Answers:

- The easy to use it
- Very intersecting
- That at least for us it "just worked".
- Easy to use
- It was refreshing to see a different approach than that I was familiar with.
- The quick implementation possibility and the ability to create digital scenes as well and share them with the team members, not requiring simultaneous work, but every member could work on one scene
- The possibility of devising conceptual models starting from very real, tangible objects which helps in visualizing the overall structure and processes.
- Cool interaction tool if pictures are already there.
- easy way to digitalize scenes created in a workshop
- Digitalization of design thinking scenarios
- It's very intuitive and easy to use
- Intuitive
- Collaborative working, visual understanding, fast adaption
- the feeling of creating a usecase together as a group and the design process in general, which is then transferred into a digital version
- I was enjoy very much using pictures and tool

What did you not like about the Scene2Model tool? Answers:

- That at least for others it did not "just work"
- UX could be improved - create it more "self-explanatory"
- The metamodel is very stiff, not allowing us to deploy our own metamodel to group the additional concepts. This made it harder to use the digital version, as we had to group concepts together that were quite different in our metamodel but belonged to the same categories in the scene2model metamodel

A.5. Workshop Materials and Evaluation

- User-friendliness could be improved a little bit, maybe some more modern user interface.
- It is unnecessary work if you have to think about concepts and representations and the camera and the recognition is not working.
- problems with recognizing QR codes
- It is very burdensome and introduces a lot of overhead for at least this purpose
- That there is no way of rotating elements on the canvas
- The steps until all pictures are solved in the folders. It was faster and easier to put the picture directly into the folder structure and change the name.
- some old-some design concepts
- This is the first time with the tool so every steps, I tried to get familiar with how to use the tool correctly.

Do you have any other comments about the Scene2Model tool? Answers:

- No
- I have never done anything with a tool like it but I can see why it could be useful, in my experience I would probably just draw separately and model separately, but if tech improves perhaps I would use it. For now it is not user friendly enough and does not add enough value to the process in my mind.
- no
- It is a nice experience to play around with
- Not about Scene2Model, but about the working sessions. Maybe the sessions should have an external coordinator that's a bit more involved in checking the workshop's progress, guiding the team, and making sure the steps are followed. Of course, the coordinator should not micromanage everything, but act as a guide making sure everything is on track. Thank you.
- Please implement the rotation functionality
- No.

A.5.4. Questionnaire for the Third Workshop Evaluation

Digital Leader Survey

Thank you very much for participating in this survey.

Please answer the following questions based on your experiences during the practical *Digital Leader* sessions at the NEMO Summerschool. Your feedback is valuable and will help us improve the learning experience.

1. Personal information:

Markieren Sie nur ein Oval pro Zeile.

	18-24	25-34	35-44	>45
Age range	☐	☐	☐	☐

2. Academic field:

3. Occupation:

4. How knowledgeable are you in the topic of design thinking?

Markieren Sie nur ein Oval.

	1	2	3	4	5
Not : ☐	☐	☐	☐	☐	☐ Very much

5. How knowledgeable are you in the topic of conceptual modelling?

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Not :	☐	☐	☐	☐	☐	Very much

6. How knowledgeable are you in setting up cyber-physical environments (e.g., setting up the sensors and actuators)?

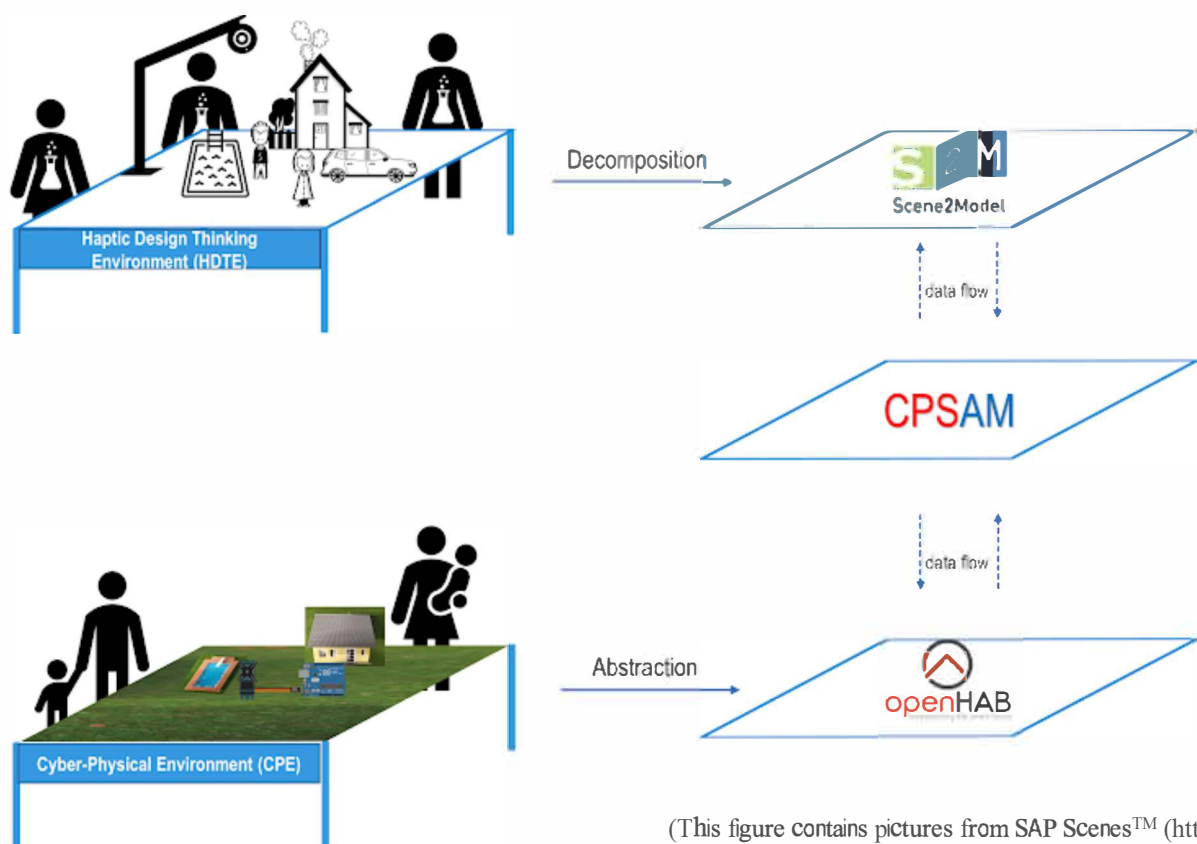
Markieren Sie nur ein Oval.

	1	2	3	4	5	
Not :	☐	☐	☐	☐	☐	Very much

Digital Innovation Ecosystem

Please answer the following questions based on your experiences during the practical *Digital Leader* sessions at the NEMO Summerschool. Your feedback is valuable and will help us improve the learning experience.

7. On a scale from 1 to 5, how well do you understand the three-layered architecture for digital innovations presented in the Summerschool?



Markieren Sie nur ein Oval.

1 2 3 4 5

Not : ☐ ☐ ☐ ☐ ☐ Very much

(This figure contains pictures from SAP Scenes™ (<https://apphaus.sap.com/resource/scenes>) and pixabay.com (<https://pixabay.com/>))

8. Use your own words to describe the three-layered architecture for digital innovations.

Digital Twin of Haptic Design Thinking Environment with Scene2Model

In the following, please tell us about your experience with Scene2Model tool gained during the guided workshop on Monday, 24.7.23, and the group-internal workshops in the second week.

9. The haptic design thinking environment was beneficial in generating innovative scenarios.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

10.

The Scene2Model tool effectively supported the digitalization of the created scenarios in the haptic design thinking environment.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

11. Adapting the available figures (paper and digital) to the specific case improved the visualization and comprehensibility of the created story.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

12. The added value from adapting the figure outweighs the additional effort.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

13. The digital model outperforms a photograph in capturing the knowledge collected in the workshop.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

14. The tangible paper figures foster creativity and cooperation in the workshops.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

15. Within the workshop and the Scene2Model tool, I was able to represent all the notational concepts that I wanted to represent.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

16. Based on the time I spent using the tool, I would require more time to feel familiar with its use.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Strongly	☐	☐	☐	☐	☐	Strongly Agree

Cyber-Physical Environment

In the following, please tell us about your experience with the cyber-physical environment gained during the *Digital Leader* sessions.

17. It was challenging to establish a real-world proof-of-concept within the cyber-physical environment.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Strongly	☐	☐	☐	☐	☐	Strongly Agree

18. Setting up a cyber-physical environment was beneficial for understanding and abstracting the relevant technical capabilities needed in the context of the created scenario.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Strongly	☐	☐	☐	☐	☐	Strongly Agree

Conceptual Modeling layer

In the following, please tell us about your experience
with the conceptual modeling layer gained during the *Digital Leader* sessions.

19. The conceptual modeling layer helped me to understand the relationships between the haptic design thinking environment and the cyber-physical environment.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

20. The conceptual modeling layer helped me to understand the value of using conceptual models in a domain for a specific purpose.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Stroi	☐	☐	☐	☐	☐	Strongly Agree

21.

The CPSAM modeling method eased the definition and deployment of the capabilities needed for the defined scenario.

Markieren Sie nur ein Oval.

	1	2	3	4	5	
Strongly Disagree	☐	☐	☐	☐	☐	Strongly Agree

Additional Statements

Please use the following questions to tell us your thoughts about the *Digital Leader* sessions.

22. What did you like about the *Digital Leader* sessions?

23. What did you not like about the *Digital Leader* sessions?

24. Would you recommend the approach used during the *Digital Leader* sessions to your peers?

Markieren Sie nur ein Oval.

☐ Yes

☐ No

25. Please share any final comments, suggestions, or feedback about the *Digital Leader* sessions.

Dieser Inhalt wurde nicht von Google erstellt und wird von Google auch nicht unterstützt.

Google