Creating Augmented Reality Applications Using Large Language Models: Experiments with the CMAG Framework and ARWFMM

Fabian Muff, Hans-Georg Fill

[1]Research Group Digitalization and Information Systems, Department of Informatics, University of Fribourg
Bd. de Pérolles 90
Fribourg, 1700 CH - Switzerland
fabian.muff@unifr.ch — hans-georg.fill@unifr.ch

Abstract

In recent years, there has been a notable advancement in machine learning technology, resulting in the creation of commercial and open-source large language models (LLMs) such as ChatGPT and Llama. These models are currently being investigated for their potential applications in various fields, including the generation of code for augmented reality applications. Given the complexity of augmented reality, the direct generation of code for such applications using LLMs remains complex and hardly verifiable. Therefore, we examine in this paper how the previously introduced concept of Conceptual Model Augmented Generative Artificial Intelligence (CMAG) can support the comprehensibility of an LLM's output on the basis of the Augmented Reality Workflow Modeling Method (ARWFMM). We illustrate the results from a first experiment, which are promising for future work in this area.

Introduction

Developing context-aware augmented reality (AR) applications is a challenging task that requires a systematic approach (Yigitbas et al. 2020). Several approaches exist for facilitating the creation of AR applications, e.g., in the form of no-code or low-code tools. However, creating AR applications remains a difficult task due to various challenges faced by AR developers, such as not knowing where to start or making the right design choices (Ashtari et al. 2020).

The use of large language models (LLMs) could be helpful to create executable code for AR applications, or the direct interaction with AR applications on the basis of a natural language input (Srinidhi, Lu, and Rowe 2024). However, also the interpretation of code generated by LLMs remains complex, since the LLM output can be very comprehensive and the end user might not be capable to verify if it corresponds to the provided input.

To overcome this problem, conceptual modeling can be used as an intermediate layer to enhance comprehensibility and identify LLM hallucinations (Muff and Fill 2024a). For this purpose, the *Conceptual Model Augmented Generative Artificial Intelligence* (*CMAG*) framework has been introduced (Fill et al. 2024). *CMAG* provides a framework and a prompt structure for describing how to augment the output

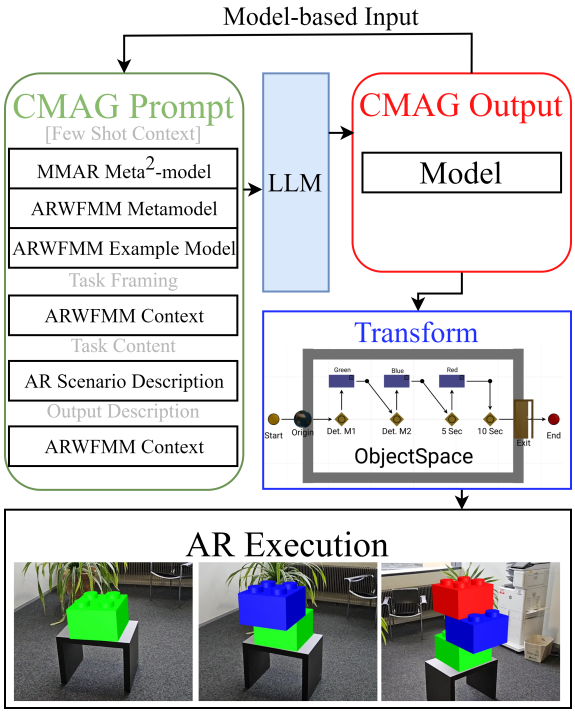

Figure 1: Illustration of the CMAG framework in the context of ARWFMM model generation, with a natural language description of the resulting AR application scenario.

of generative artificial intelligence (AI) models using conceptual models, thereby enhancing the understandability of the output and identifying potential hallucinations.

By combining conceptual modeling languages for the creation of AR applications with generative artificial intelligence and LLMs, the initially described complexity for creating AR applications could be lowered. Given the availability of conceptual modeling approaches for the no-code definition of AR applications (Muff and Fill 2023; Wild et al. 2014; Grambow et al. 2021), *CMAG* offers an optimal means of bridging the gap between natural language LLM inputs and the typically very complex and hardly understandable AR application code.

Thus, in this short paper, we apply in a first experiment

the *CMAG* framework to the "Augmented Reality Workflow Modeling Method" (ARWFMM) (Muff and Fill 2023) for the no-code definition of AR applications. The paper is structured as follows. First, the *CMAG* framwork an the *ARWFMM* are briefly introduced. Then, an experiment for the application of *CMAG* to an augmented reality use case is shown. The results of this experiment are subsequently discussed. The last section draws a conclusion and an gives outlook to future work.

Foundations

In this section, we briefly describe the *CMAG* framework and the *ARWFMM* as the basis for our experiments.

Conceptual Model Augmented Generative Artificial Intelligence (CMAG)

Conceptual models have a long tradition for supporting human understanding and communication in requirements engineering and are used today in many fields of computer science (Mylopoulos 1992; Härer and Fill 2020). Soon after the release of ChatGPT in 2022 it was found that the large language models can be successfully used to create and interpret conceptual models (Fill, Fettke, and Köpke 2023). This is due to: a.) the underlying training data, which typically includes code samples for generating conceptual models, e.g., in PlantUML[1] syntax, and b.) the unique abilities of these foundation models to learn and apply new languages on-the-fly using so-called few shot prompts. This means that LLMs can be forced to create and interpret conceptual models of standard types such as UML, BPMN or ArchiMate, as well as those based on domain-specific languages. Despite current limitations in the size and complexities of the input that is processable by state-of-the-art LLMs for this purpose (Muff and Fill 2024b), the use of reduced grammars or standard data formats showed the feasibility of such approaches (Baumann et al. 2024; Fill, Fettke, and Köpke 2023; Dolha and Buchmann 2024).

On this basis, it has recently been proposed to use conceptual models as a means for facilitating the understanding of the output of large language models (Fill et al. 2024). Given the fact that LLMs often produce large amounts of output which has to be subsequently verified by human experts, conceptual models can aid in visually representing this output and thus enabling humans to more quickly assess its quality. This approach has been denoted as Conceptual Model Augmented Generative AI (CMAG). It proposes to include in the few shot context of CMAG prompts to an LLM, information about the conceptual modeling language to be used, e.g., by referring to languages the LLM has been trained on or by providing a simplified, formal description of a meta[2]-model, a metamodel and a sample model instance. Thereafter, the actual task for the LLM is described, which includes the task framing, the task content, and the output description by referring to the few shot context - see Figure 1. In this manner, the LLM is instructed to operate within the context of the conceptual modeling language. The output generated by the LLM are then models according to the provided language description in the few shot context.

Augmented Reality Wokflow Modeling Method

The Augmented Reality Workflow Modeling Method (ARWFMM) (Muff and Fill 2023) is a domain-specific visual modeling method designed to facilitate the creation of augmented reality applications without requiring programming expertise. It enables users to define AR workflows through a model-based approach, thereby facilitating the development process for complex AR scenarios.

The language comprises three so-called *SceneTypes*: the *ObjectSpace* model, the *Statechange* model, and the *FlowScene* model. The *ObjectSpace* model defines the virtual and real-world objects relevant to the AR environment, including the concepts of (1) *Augmentation*, e.g., 3D objects, images, or labels and (2) *Detectables*, e.g., image markers or 3D objecs recognized by the AR system. The *Statechange* model specifies dynamic alterations to the properties of *Augmentations*, such as visibility, position, or rotation, enabling responsive and interactive AR experiences. The concept used in a *Statechange* model is called *Reference* and creates a reference to the targeted *Augmentation*. Finally, the *FlowScene* model outlines the overall workflow of the AR application, defining *Start* and *End* of an AR workflow, *Conditions* that trigger *Statechanges*, determining how the application responds to specific events or environmental factors. Furthermore, there is a concept *ObjectSpace* that links the *ObjectSpace* model, including an *Origin* to link a *Detectable* from the *ObjectSpace* as world-origin reference point. Furthermore, there is an *Exit* to exit an AR workflow. To facilitate the implementation of *ARWFMM*, Muff et al. (2024c) introduced *MMAR*, a web-based modeling environment relying on a 3D metamodeling language (Muff and Fill 2021). *MMAR* supports the visual definition and execution of AR workflows. Built on state-of-the-art web technology, *MMAR* provides a flexible and intuitive platform for creating three-dimensional AR scenarios without requiring programming skills. The *ARWFMM* and its supporting toolset represent a significant advancement in democratizing AR application development, enabling users to create complex and dynamic AR workflows through an accessible, visual modeling approach.

Experiment with CMAG und ARWFMM

To demonstrate the application of the *CMAG* framework in a use case, we conducted an experiment to create an augmented reality application on the basis of the *ARWFMM*. Figure 1 shows the entire process of the *CMAG* framework in the context of this experiment. It must be noted, that the experiment only considers the interface to the LLM. The transformation of the LMM output into an appropriate format for further processing of the models is not within the scope of this investigation.

In a first step, the *few shot context* of the prompt was defined. Since we are relying on *MMAR* as metamodeling language and *ARWFMM* as modeling language, the *few shot context* consists of the formal concepts of the *MMAR*

[1] See https://plantuml.com/en/

Listing 1: CMAG prompt example including the MMAR meta^2-model and the ARWFMM metamodel description.

```
Assume the following meta2model:
Attribute (w),
Class (x) := (Attribute(w1), Attribute(w2), ...),
Role (y),
RelationClass (z) := (Role(a), Role(b)),

Assume the following metamodels:

ObjectSpace:
Class (Augmentation) := (Attribute (Name)),
Class (Detectable) := (Attribute (Name)),

Statechange:
Class (Reference) := (Attribute (Name), Attribute
↪  (ReferenceToAugmentation)),

FlowScene:
Class (Start): (Attribute (Name)),
Class (End) : (Attribute (Name)),
Class (ObjectSpace) : (Attribute (Name), Attribute
↪  (ReferenceToObjectSpaceModel)),
Class (Condition) := (Attribute (Name),
↪  Attribute(ConditionType), Attribute(Seconds?),
↪  Attribute(ReferenceToDetectable?)),
Class (Statechange) := (Attribute (Name), Attribute
↪  (ReferenceToStatechangeModel)),
Class (Origin) := (Attribute (Name), Attribute
↪  (ReferenceToDetectable)),
Class (Exit) := (Attribute (Name)),
Role(from_start) = (Start),
Role(to_start) = (Origin),
RelationClass(start) := (from_start, to_start),
Role(from_end) = (Exit),
Role(to_end) = (End),
RelationClass(end) := (from_end, to_end),
Role(has_condition_from) = (Origin, Statechange),
Role(has_condition_to) = (Condition),
RelationClass(has_condition) := (has_condition_from,
↪  has_condition_to),
Role(triggers_from) = (Condition),
Role(triggers_to) = (Statechange),
RelationClass(triggers) := (triggers_from,
↪  triggers_to)
```

meta^2-model, the *ARWFMM* metamodel described with the concepts of the meta^2-model and example instance models of an *ARWFMM* use case. Listing 1 shows the CMAG prompt example of the *MMAR* meta^2-model and the *ARWFMM* metamodel description. The first part of Listing 2 shows the example description of an *ARWFMM* model describing one *ObjectSpace* model, two *Statechange* models, and one *FlowScene* model.

In the second step, the *Task Content* was defined. In this case, the goal was to describe the resulting AR workflow in natural language. This is the task a potential end user of the approach can describe. The lower part of Listing 2 shows this *AR Scenario Description*. In this example, the prompt describes an AR application with three very simple *Statechanges*. First, the user must scan the origin marker to calibrate the AR environment. Then, a second color marker must be scanned to visualize a green Lego cube in the scene (1. *Statechange*). After that, the application waits, until a third color marker is scanned. This triggers the visualization of a blue lego cube (2. *Statechange*) on top of the first one. After a five second time trigger, a blue lego cube is visualized on top of the two others (3. *Statechange*). After a final ten second time trigger, the application terminates. On the

Listing 2: CMAG prompt example including the ARWFMM model instance example description and the AR scenario description.

```
Assume the following sample models:

ObjectSpace Model: 'Use Case 1 ObjectSpace Model'
Detectable(Name: 'Origin'),
Detectable(Name: 'Marker1'),
Detectable(Name: 'Marker2'),
Augmentation(Name: 'Screw 1'),
Augmentation(Name: 'Arrow 1')

Statechange Model: 'Place Screw 1 Statechange Model'
Reference(Name: 'Place Screw 1',
↪  ReferenceToAugmentation: 'Screw 1')

Statechange Model: 'Push Button 1 Statechange Model'
Reference(Name: 'Push Button 1',
↪  ReferenceToAugmentation: 'Arrow 1')

FlowScene Model: 'Use Case 1 FlowScene Model'
Start(Name: 'Start1'),
Origin(Name: 'Origin', ReferenceToDetectable:
↪  'Origin'),
Condition(Name: 'Detect Marker 1', ConditionType:
↪  'Detect', ReferenceToDetectable: 'Marker1'),
Condition(Name: 'Detect Marker 2', ConditionType:
↪  'Detect', ReferenceToDetectable: 'Marker2'),
Condition(Name: '5 Sec Timer', ConditionType:
↪  'Timer', Seconds: '5'),
Statechange(Name: 'Place Screw 1',
↪  ReferenceToStatechangeModel: 'Place Screw 1
↪  Statechange Model'),
Statechange(Name: 'Push Button 1',
↪  ReferenceToStatechangeModel: 'Push Button 1
↪  Statechange Model'),
Exit(Name: 'Exit1'),
End(Name: 'End1'),
ObjectSpace(Name: 'Use Case 1 ObjectSpace',
↪  ReferenceToObjectSpaceModel: 'Use Case 1
↪  ObjectSpace Model'),
from_start(r1) = (Start1),
to_start(r2) = (Origin),
start(r1, r2),
has_condition_from(r5) = (Origin),
has_condition_to(r6) = (Detect Marker 1),
has_condition(r5, r6),
triggers_from(r7) = (Detect Marker 1),
triggers_to(r8) = (Place Screw 1),
triggers(r7, r8),
has_condition_from(r9) = (Place Screw 1),
has_condition_to(r10) = (Detect Marker 2),
has_condition(r9, r10),
triggers_from(r11) = (Detect Marker 2),
triggers_to(r12) = (Push Button 1),
triggers(r11, r12),
has_condition_from(r13) = (Push Button 1),
has_condition_to(r14) = (5 Sec Timer),
has_condition(r13, r14),
triggers_from(r15) = (5 Sec Timer),
triggers_to(r16) = (Exit1),
triggers(r15, r16),
from_end(r17) = (Exit1),
to_end(r18) = (End1),
end(r17, r18)

Create the necessary model according to this
↪  metamodels for the following AR application
↪  workflow::

Make a proposal for a Workflow that represents an
↪  augmented reality application. First, a origin
↪  color marker should be detected. After the
↪  detection of the second color marker, a green
↪  lego cube is shown. After the detection of a
↪  third color marker, a blue lego cube is shown on
↪  top of the green cube. After 5 seconds, a red
↪  lego cube is shown on top of the blue lego cube.
↪  After 10 seconds, the application exits.

Output the result as in the notation shown before.
Only show the code surrounded with triple backticks.
```

bottom of Figure 1 an example of the resulting three *Statechanges* in a running AR application is visible.

In a next step, the output generated by the LLM must be transformed into an adequate format. The output generated in our experiment is neither directly executable, nor in the format of the *MMAR* metamodeling language. Thus, to make the generated models interpretable by the *MMAR* platform and comprehensible to the AR application designer, the output must be transformed into the full *ARWFMM* language. This, however, is not in the scope of this contribution.

Results

The LLMs used for the experiment are *GPT-4*[2], *GPT-4o*[3] and *Llama 3.3*[4]. Listing 3 shows the results of the *CMAG* prompt on the basis of the input prompt shown in Listings 1 and 2 while using *GPT-4*. As visible in the LLM output, the results are quite good. The LLM created the *ObjectSpace*, the three *Statechange* and the *FlowScene* models required for the described use case. All the *Detectables*, *Augmentations*, *References*, as well as the *FlowScen* concepts were generated. The attributes and their values were also inferred correctly from the AR scenario description. The syntax of the output format was correct.

Furthermore, we also tested the prompt with *GPT-4o* and *Llama 3.3*. The two outputs of *GPT* are identical in the provided content, with minor naming differences. However, the *Llama* LLM introduced some confusion regarding the differentiation between the various *SceneTypes*.

Conclusion and Outlook

In this paper we applied the *CMAG* framework to the *ARWFMM* for bridging the gap between natural language LLM inputs and the intricate generation of AR application code.

In conclusion, the results of this experiment are highly promising. The generated output of the experiment is of excellent quality and could be used for further model transformation. This work, however, also has some limitations. First, it must be noted that this is only a preliminary experiment, and the provided sample metamodel of *ARWFMM* does not contain all the attributes that would be required for transforming the generated output into a valid *ARWFMM* model. For example, there are no attributes for defining the 3D manipulations in *Statechanges*, and the required input data for images and 3D objects is missing. Therefore, in future work, further evaluations, e.g. with the LLM evaluation framework RAGA[5], will be performed with a more detailed metamodel and the transformation into valid *ARWFMM* models will be addressed.

Acknowledgments

Financial support is gratefully acknowledged by the *Smart Living Lab* (https://www.smartlivinglab.ch/en/) funded by the University of Fribourg, EPFL, and HEIA-FR.

[2]See https://platform.openai.com/docs/models/gpt-4

[3]See https://platform.openai.com/docs/models/gp#gpt-4o

[4]See https://www.llama.com/docs/overview/

[5]https://docs.ragas.io/en/stable/

Listing 3: CMAG output describing the ARWFMM model instances needed for the described AR use case.

```
ObjectSpace Model: 'AR Application ObjectSpace
↪  Model'
Detectable(Name: 'Origin Color Marker'),
Detectable(Name: 'Second Color Marker'),
Detectable(Name: 'Third Color Marker'),
Augmentation(Name: 'Green Lego Cube'),
Augmentation(Name: 'Blue Lego Cube'),
Augmentation(Name: 'Red Lego Cube')

Statechange Model: 'Show Green Lego Cube Statechange
↪  Model'
Reference(Name: 'Show Green Lego Cube',
↪  ReferenceToAugmentation: 'Green Lego Cube')

Statechange Model: 'Show Blue Lego Cube Statechange
↪  Model'
Reference(Name: 'Show Blue Lego Cube',
↪  ReferenceToAugmentation: 'Blue Lego Cube')

Statechange Model: 'Show Red Lego Cube Statechange
↪  Model'
Reference(Name: 'Show Red Lego Cube',
↪  ReferenceToAugmentation: 'Red Lego Cube')

FlowScene Model: 'AR Application FlowScene Model'
Start(Name: 'Start1'), Origin(Name: 'Origin',
↪  ReferenceToDetectable: 'Origin Color Marker'),
Condition(Name: 'Detect Second Marker',
↪  ConditionType: 'Detect', ReferenceToDetectable:
↪  'Second Color Marker'),
Condition(Name: 'Detect Third Marker',
↪  ConditionType: 'Detect', ReferenceToDetectable:
↪  'Third Color Marker'),
Condition(Name: '5 Sec Timer', ConditionType:
↪  'Timer', Seconds: '5'),
Condition(Name: '10 Sec Timer', ConditionType:
↪  'Timer', Seconds: '10'),
Statechange(Name: 'Show Green Lego Cube',
↪  ReferenceToStatechangeModel: 'Show Green Lego
↪  Cube Statechange Model'),
Statechange(Name: 'Show Blue Lego Cube',
↪  ReferenceToStatechangeModel: 'Show Blue Lego
↪  Cube Statechange Model'),
Statechange(Name: 'Show Red Lego Cube',
↪  ReferenceToStatechangeModel: 'Show Red Lego Cube
↪  Statechange Model'),
Exit(Name: 'Exit1'),
End(Name: 'End1'),
ObjectSpace(Name: 'AR Application',
↪  ReferenceToObjectSpaceModel: 'AR Application
↪  ObjectSpace Model'),
from_start(r1) = (Start1),
to_start(r2) = (Origin),
start(r1, r2),
has_condition_from(r3) = (Origin),
has_condition_to(r4) = (Detect Second Marker),
has_condition(r3, r4),
triggers_from(r5) = (Detect Second Marker),
triggers_to(r6) = (Show Green Lego Cube),
triggers(r5, r6),
has_condition_from(r7) = (Show Green Lego Cube),
has_condition_to(r8) = (Detect Third Marker),
has_condition(r7, r8),
triggers_from(r9) = (Detect Third Marker),
triggers_to(r10) = (Show Blue Lego Cube),
triggers(r9, r10),
has_condition_from(r11) = (Show Blue Lego Cube),
has_condition_to(r12) = (5 Sec Timer),
has_condition(r11, r12),
triggers_from(r13) = (5 Sec Timer),
triggers_to(r14) = (Show Red Lego Cube),
triggers(r13, r14),
has_condition_from(r15) = (Show Red Lego Cube),
has_condition_to(r16) = (10 Sec Timer),
has_condition(r15, r16),
triggers_from(r17) = (10 Sec Timer),
triggers_to(r18) = (Exit1),
triggers(r17, r18),
from_end(r19) = (Exit1),
to_end(r20) = (End1),
end(r19, r20)
```

References

Ashtari, N.; Bunt, A.; McGrenere, J.; Nebeling, M.; and Chilana, P. K. 2020. Creating Augmented and Virtual Reality Applications: Current Practices, Challenges, and Opportunities. In Bernhaupt, R.; Mueller, F. F.; Verweij, D.; Andres, J.; McGrenere, J.; Cockburn, A.; Avellino, I.; Goguey, A.; Bjøn, P.; Zhao, S.; Samson, B. P.; and Kocielnik, R., eds., *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, 1–13. ACM.

Baumann, N.; Diaz, J. S.; Michael, J.; Netz, L.; Nqiri, H.; Reimer, J.; and Rumpe, B. 2024. Combining Retrieval-Augmented Generation and Few-Shot Learning for Model Synthesis of Uncommon DSLs. In Giese, H.; and Rosenthal, K., eds., *Modellierung 2024 - Workshop Proceedings, Potsdam, Germany, March 12-15, 2024*, 7. Gesellschaft für Informatik e.V.

Dolha, D. N.; and Buchmann, R. A. 2024. Generative AI for BPMN Process Analysis: Experiments with Multi-modal Process Representations. In Repa, V.; Matulevicius, R.; and Laurenzi, E., eds., *Perspectives in Business Informatics Research - 23rd International Conference on Business Informatics Research, BIR 2024, Prague, Czech Republic, September 11-13, 2024, Proceedings*, volume 529 of *Lecture Notes in Business Information Processing*, 19–35. Springer.

Fill, H.; Fettke, P.; and Köpke, J. 2023. Conceptual Modeling and Large Language Models: Impressions From First Experiments With ChatGPT. *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, 18: 3.

Fill, H.; Härer, F.; Vasic, I.; Borcard, D.; Reitemeyer, B.; Muff, F.; Curty, S.; and Bühlmann, M. 2024. CMAG: A Framework for Conceptual Model Augmented Generative Artificial Intelligence. In Gallinucci, E.; Yasar, H.; Liaskos, S.; Marcel, P.; Chen, P. P.; de Cesare, S.; and Gailly, F., eds., *Companion Proceedings of the 43rd International Conference on Conceptual Modeling: ER Forum, Special Topics, Posters and Demos Co-located with ER 2024, Pittsburgh, Pennsylvania, USA, October 28-31, 2024*, volume 3849 of *CEUR Workshop Proceedings*, 56–69. CEUR-WS.org.

Grambow, G.; Hieber, D.; Oberhauser, R.; and Pogolski, C. 2021. *A Context and Augmented Reality BPMN and BPMS Extension for Industrial Internet of Things Processes*, volume 436 of *Lecture Notes in Business Information Processing*, 379–390. Cham: Springer. ISBN 978-3-030-94342-4.

Härer, F.; and Fill, H. 2020. Past Trends and Future Prospects in Conceptual Modeling - A Bibliometric Analysis. In *Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings*, 34–47. Springer.

Muff, F.; and Fill, H. 2021. Initial Concepts for Augmented and Virtual Reality-based Enterprise Modeling. In *Proceedings of the ER Demos and Posters 2021 co-located with 40th International Conference on Conceptual Modeling (ER 2021), St. John's, NL, Canada, October 18-21, 2021*, volume 2958 of *CEUR Workshop Proceedings*, 49–54. CEUR-WS.org.

Muff, F.; and Fill, H. 2023. A Domain-Specific Visual Modeling Language for Augmented Reality Applications Using WebXR. In Almeida, J. P. A.; Borbinha, J.; Guizzardi, G.; Link, S.; and Zdravkovic, J., eds., *Conceptual Modeling - 42nd International Conference, ER 2023, Lisbon, Portugal, November 6-9, 2023, Proceedings*, volume 14320 of *Lecture Notes in Computer Science*, 334–353. Springer.

Muff, F.; and Fill, H. 2024a. Limitations of ChatGPT in Conceptual Modeling: Insights from Experiments in Metamodeling. In Giese, H.; and Rosenthal, K., eds., *Modellierung 2024 - Workshop Proceedings, Potsdam, Germany, March 12-15, 2024*, 8. Gesellschaft für Informatik e.V.

Muff, F.; and Fill, H. 2024b. Limitations of ChatGPT in Conceptual Modeling: Insights from Experiments in Metamodeling. In Giese, H.; and Rosenthal, K., eds., *Modellierung 2024 - Workshop Proceedings, Potsdam, Germany, March 12-15, 2024*, 8. Gesellschaft für Informatik e.V.

Muff, F.; and Fill, H. 2024c. M2AR: A Web-based Modeling Environment for the Augmented Reality Workflow Modeling Language. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24)*. ACM. ISBN 979-8-4007-0622-6/24/09.

Mylopoulos, J. 1992. Conceptual modelling and Telos. In *Conceptual Modeling, Databases, and Case: An Integrated View of Information Systems Development*, 49–68. John Wiley & Sons, Inc. ISBN 0471554626.

Srinidhi, S.; Lu, E.; and Rowe, A. 2024. XaiR: An XR Platform that Integrates Large Language Models with the Physical World. In Eck, U.; Sra, M.; Stefanucci, J. K.; Sugimoto, M.; Tatzgern, M.; and Williams, I., eds., *IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2024, Bellevue, WA, USA, October 21-25, 2024*, 759–767. IEEE.

Wild, F.; Scott, P.; Lefrere, P.; Karjalainen, J.; Helin, K.; Naeve, A.; and Isaksson, E. 2014. Towards data exchange formats for learning experiences in manufacturing workplaces. In Kravcik, M.; Mikroyannidis, A.; Pammer, V.; Prilla, M.; Ullmann, T. D.; and Wild, F., eds., *Proceedings of the 4th Workshop on Awareness and Reflection in Technology-Enhanced Learning, ARTELEC-TEL 2014, Graz, Austria, September 16, 2014*, volume 1238 of *CEUR Workshop Proceedings*, 23–33. CEUR-WS.org.

Yigitbas, E.; Jovanovikj, I.; Sauer, S.; and Engels, G. 2020. *On the Development of Context-Aware Augmented Reality Applications*, volume 11930 of *Lecture Notes in Computer Science*, 107–120. Cham: Springer International Publishing. ISBN 978-3-030-46539-1.