An Agile and Ontology-based Meta-Modelling Approach for the Design and Maintenance of Enterprise Knowledge Graph Schemas

Emanuele Laurenzi[*,a]

[a] FHNW - University of Applied Sciences and Arts Northwestern Switzerland. Riggenbachstrasse 16, 4006 Olten, Switzerland.

Abstract. *Enterprise Knowledge Graphs (EKGs) are increasingly created and used by organizations for structuring knowledge of a particular application domain and consequent exploitation through analysis, reasoning and integration of information extracted from different data sources. Yet, one of the main challenges is designing and maintaining EKG's schema, which requires high expertise in ontology engineering and the addressed application domain. Various approaches and tools offer visual aids, but they target ontology engineers and neglect the domain experts. Domain-specific modelling languages (DSML), in contrast, offer concepts that domain experts easily understand because of their tailored graphical notations. DSMLs can be created with a meta-modelling approach, which does not require ontology expertise but can be adopted for the equivalent creation of EKG schema. This paper presents an approach that extends the traditional and sequential meta-modelling approach with an agile one, allowing domain-specific adaptations of modelling languages and testing them on the fly. In this way, the domain experts are facilitated to be in the engineering loop. Moreover, the approach foresees automatic mechanisms to ensure that an EKG schema is designed while performing the visual domain-specific adaptations. The approach has been developed by following the Design Science Research methodology, which led to the creation of a prototypical tool called AOAME. The latter has been used to implement real-world scenarios to evaluate the proposed approach's utility. The correct design of the approach has been evaluated by tracing the prototype functionalities back to the requirements.*

Keywords. Agile and Ontology-based Meta-Modelling • Enterprise Knowledge Graphs • Domain-specific Adaptations

Communicated by Robert Buchmann, Georg Grossmann, Andreas L. Opdahl. Received 2022-09-22. Accepted on 2023-10-01.

1 Introduction

Knowledge Graphs (KGs) Chaudhri et al. (2022) have matured as a topical technique that is increasingly adopted by enterprises for structuring knowledge and its subsequent analysis and reasoning as well as for integrating information extracted from different data sources.

KGs also play a central role in Artificial Intelligence systems, as (1) their structured knowledge can be used as input to improve predictions of Machine Learning, and (2) they can represent knowledge extracted by Machine Learning in a formal and explainable manner (Peng et al. 2023; Van Harmelen and Teije 2019).

At its core, a Knowledge Graph (KG) consists of a directed labelled graph in which domain-specific meanings are associated with nodes and edges (Chaudhri et al. 2022). While a node represents any real-world entity, an edge label captures the relationship of interest between the

* Corresponding author.
E-mail. emanuele.laurenzi@fhnw.ch

two nodes. Nodes and relationships can be categorized by types, which supplement them with meaning. Edges also capture the subclass relationship among entity types, which is known as a taxonomy. Entity types and their relationships form the schema of a knowledge graph and can be defined as an ontology (Peng et al. 2023).

In this paper, the terms "knowledge graph schema" and "ontology" are treated as synonyms and used interchangeably.

Nodes, relationships and their types are typically expressed in an ontology language, which makes knowledge machine-interpretable, thus reducing ambiguity, deriving meaningful information that is specific to an application's domain and enabling knowledge automation.

Differently from a generic KG, Enterprise Knowledge Graphs (EKG) Gomez-Perez et al. (2017) address domain-specific problems (Kejriwal et al. 2019), where concepts belong to a realm of business enterprise. They are the equivalent of 'lightweight' Enterprise Ontology because the expressivity of the ontology language is rather low, e. g., like the W3C standard RDF(S) (W3C 2014).

Haase (2019) argues that EKGs became a key technology to enable enterprise data management, which serves as an integration hub across enterprise data sources. With the ever-increasing availability of data and the adoption of data-driven approaches in companies, the establishment of EKG schemas is becoming a competitive advantage. In fact, EKG schemas not only provide meaning to data, but they enable quick reuse and repurposing of content for flexible and timely support of business decisions.

Despite the several benefits, the creation of schemas for EKGs remains a highly complex engineering effort (Abu-Salih 2021; Kejriwal 2019; Li et al. 2020; Meissner and Thor 2021; Xiaohan 2020). Existing visual editors like Protégé[1] or Stardog[2] can support this activity but target ontology experts only. However, the engineering of an EKG does not only require high expertise in knowledge engineering but also domain expertise. It is not common to find both expertise in one person, which commonly leads to several back and forth between the knowledge engineers and the domain experts, hence a time-consuming practice. Moreover, entity types and their relations are likely to change over time because of the high correlation to an addressed enterprise reality. Hence, maintaining a schema for an enterprise knowledge graph is as important as its creation to ensure high quality and meaningful use over time (Abu-Salih 2021).

To tackle this challenge, this paper presents an approach that includes domain experts in the creation and maintenance of the EKGs schemas. For this, the approach builds on techniques and approaches from the Enterprise Modelling discipline. Specifically, techniques of meta-modelling and semantic lifting of meta-models were investigated by consulting literature, interviewing modelling experts from the industry and analysing two cases through a case study strategy. Findings led to suggest a novel approach that, on the one hand, allows performing domain-specific adaptations of modelling languages. Modelling languages (e. g., BPMN (OMG 2011)) are equipped with syntax and semantics and can be easily understood by domain experts because of their graphical notations. Domain-specific adaptations are performed by a language engineer to tailor modelling languages to a particular addressed enterprise reality, which results in so-called Domain-Specific Modelling Languages (DSMLs). Concepts and relations of a DSML are automatically specified in an ontology language, while domain-specific adaptations are performed. Thus, the language engineer can add or remove concepts and properties and test them visually. This has the advantage (1) of not requiring ontology expertise and (2) of including the domain expert in the engineering process of the DSML language. As a result, once the DSML is ready, the schema of the EKG is ready, too.

The paper is structured as follows. First, Sect. 2 presents the background, including the definitions of relevant terms and then related work with a

[1] https://protege.stanford.edu/
[2] https://www.stardog.com/

focus on a stat-of-the-art of those approaches that combine meta-models and models with ontologies or knowledge graphs. The remaining sections conform to the Design Science Research phases. Sect. 3 describes the awareness of problem phase, which includes findings from expert interviews and the analysis of two cases as well as the design requirements are reported. Sect. 4 describes the agile and ontology-based meta-modelling approach. Next, Sect. 5 elaborates on the agile and ontology-based meta-modelling tool AOAME, which instantiates the proposed approach. Sect. 6 deals with the evaluation, and Sect. 7 points out the limitations of the approach. Finally, Sect. 8 concludes the paper and points to future work.

2 Background and Related Work

This section provides an overview of definitions around the most relevant concepts of this work and the related work. The main theory-based requirements (TBRs) for the design of the proposed artifact are pointed out.

2.1 Knowledge Graphs

The term Knowledge Graph became popular in academia and research with the advent of the Google Knowledge Graph (Singhal 2012) as an aid of the Google search engine for better query results. As of today, there is no common agreement on the definition of the term in research, but rather subjective attempts to provide a description of it (Ehrlinger and Wöss 2016). Many authors active in the AI research discipline (e. g., (Chaudhri et al. 2022)), see the idea behind Knowledge Graphs as not new and KG are regarded as the extension of the knowledge representation technique *Semantic Network* (SN) with some differences at a computational level, e. g., differences in scalability (thousands of objects vs. billion of objects in KGs), and in development methods (top-down only for SN). Ehrlinger and Wöss (2016) define a KG as a "large network of entities, their semantic types, properties, and relationships between entities". Knowledge Graphs also relate to Semantic Web technologies, which include inference engines, semantic rules and queries and lightweight W3C ontology languages such as RDF (R. et al. 2014) and RDF(S).

Knowledge graphs can be distinguished into generic and domain-specific ones (Ehrlinger and Wöss (2016)). The generic KGs are open-world, cross-domain, or domain-independent. Examples for these, are DBPedia[3] , Cyc[4] , BabelNet[5] , CliGraph[6] .

According to Li et al. (2020), most of the existing KGs are domain-specific. Domain-specific KGs address domain-specific problems Kejriwal et al. (2019). A domain-specific knowledge graph can be defined as follows: "an explicit conceptualisation to a high-level subject-matter domain and its specific subdomains represented in terms of semantically interrelated entities and relations" (Abu-Salih 2021). Enterprise Ontology Dietz (2006) is regarded as domain-specific ontologies that represent concepts and relations that describe the realm of an enterprise. Examples of enterprise ontologies can be found in (Bertolazzi et al. 2001; Fox et al. 1996; Leppänen 2007; Uschold et al. 1998). ArchiMEO Hinkelmann et al. (2020) is an example of a lightweight enterprise ontology, which contains concepts and relations of enterprises and standard modelling languages, specified with RDF(S) (W3C 2014). Lightweight enterprise ontologies, in contrast to highly expressive ontologies (e. g., defined in some OWL languages (Allemang and Hendler 2011)), can be regarded as Enterprise Knowledge Graphs (EKGs).

2.2 Enterprise Modelling for Enterprise Knowledge Graphs (EKGs)

Enterprise Modelling (EM) is a pivotal field in Information Systems (IS) research (Frank 2014). An enterprise can be defined as a highly complex heterogeneous socio-technical information system, whose parts are interrelated within a nexus of interdependencies on different abstraction levels (Vernadat 2003). Similarly, Giachetti (2010) defines an enterprise as a "complex socio-technical

[3] https://wiki.dbpedia.org/
[4] https://www.cyc.com/
[5] https://babelnet.org/
[6] http://caligraph.org/ontology/Scientist

system that comprises interdependent resources of people, information, and technology that must interact with each others and their environment in support of a common mission". EM strives to cope with such complexity through models. Benyon et al. (1999) define a model as "a representation of something, constructed and used for a particular purpose". The "representation of something" reverts to an often commonly agreed abstraction, which describes and represents the relevant aspects of a "system under study" (SUS, also known as the "Universe of Discourse", "subject" or "domain") (Efendioglu et al. 2017). There exist many different kinds of models like graphical models, mathematical models, logical models or formal-based models like enterprise ontologies (Dietz 2006).

2.2.1 Enterprise Models

Enterprise models are graphically represented and capture relevant-enterprise aspects, which include data structure and organisation (i. e., static phenomena of Information Systems), as well as behavior, i. e., dynamic phenomena of Information Systems (Vernadat 2003). Specifically, these aspects can relate to activities, processes, information, resources, people, goals, and constraints of one or many interlinked businesses (Fox and Gruninger 1998). The ultimate goal of enterprise models is of creating value for an enterprise (Sandkuhl et al. 2018). These externalize enterprise knowledge for the support of common understanding, inter-subjective communication, documentation, analysis and operations (Vernadat 2003), e. g., answering queries, simulating behavior, and performing reasoning as well as business transformation (Zachman 1987). Examples of enterprise models are business process models and enterprise architecture models.

Bridgeland and Zahavi (2009) discusses eight ways where enterprise models generate business values: communication between people, training and learning, persuasion and selling, analysis of a business situation, compliance management, development of software requirements, direction in software engineering, knowledge management and reuse. To better understand the value of models, Karagiannis and Woitsch (2015) elaborated on the knowledge space that relates to a model. This consists of the *content*, *form*, *use* and *interpretation* dimensions. The *use* determines the domain of discourse. The *content* is the actual knowledge contained in the model. The *form* relates the formalism or representation. The *interpretation* dimension is specified in human or machine interpretation of knowledge. While the human interpretation of knowledge is fostered by graphical notations, the machine interpretation of knowledge can be enabled by expressing it in a logic-based formalism.

2.2.2 Modelling Language and Meta-Modelling for Enterprise Models

Enterprise models are created with modelling languages. Karagiannis and Kühn (2002) define a modelling language with three specifications: notation, syntax and semantics. Abstract syntax refers to the class hierarchy of modelling constructs together with their relations and attributes, through which the language terminology is defined. Modelling constructs are syntactic elements typically expressed through a graphical or textual notation. The notation is the concrete syntax of a modelling language, through which it is possible to create a model. Graphical notations should be cognitively adequate to ensure users' understanding of models (Moody 2009). A graphical notation is also known as a visual connector and is used in conjunction with modelling elements to design a model (Karagiannis and Buchmann 2018). Thus, both modelling elements and modelling relations are regarded as modelling constructs. The semantics allows determining the truth value of elements in the model with respect to the underlying reality being conceptualised (Parreiras 2012). In other words, the semantics defines the meaning of syntactic elements.

Frank (2013) argues that the semantics of a modelling language consist of abstract syntax and constraints. Constraints supplement the abstract syntax to govern how the modelling constructs

can be combined to produce valid models. For example, the BPMN specification OMG (2011) expresses constraints in natural language, which supplement the abstract syntax of BPMN. This is, however, a semantic specification limited to what Atkinson and Kuhne (2003) call the *linguistic view*. Atkinson and Kuhne (2003) stress the importance, in a modelling language, of not only specifying the *linguistic view* but also the *domain of discourse view*. The latter describes the domain knowledge, and according to D. Harel and B. Rumpe (2004), it is also known as the semantic domain and can be defined independently of the syntax. Furthermore, the authors suggest that the semantics of a modelling language can be specified in two parts: (a) the semantic domain, which provides information regarding the domain of discourse; and (b) the semantic mapping, which maps the abstract syntax into the semantic domain. The semantic mapping relates concepts from the abstract syntax to the domain semantic and is also expressed with the homonym name in the framework proposed by Karagiannis and Kühn (2002). Again according to David Harel and Bernhard Rumpe (2000), the semantic mapping should be made explicit and it is not satisfactory to define it by examples because it would not allow for analysis.

TBR1: Consistently with these findings, the proposed approach shall allow the definition of the semantics of a modelling language (1) in the abstract syntax, (2) in the semantic domain, and (3) by making explicit the mapping of modelling constructs from abstract syntax to a semantic domain.

A commonly adopted technique to create or adapt modelling languages is known as meta-modelling Karagiannis and Kühn (2002) and Strahringer (1996), also known as meta-model customization or ad-hoc customisation of meta-models Atkinson and Kuhne (2003). The meta-modelling hierarchy (see Fig. 1) distinguishes different layers of abstraction, where three layers are commonly used to specify modelling languages and related models (Efendioglu et al. 2017). Each underneath layer instantiates concepts that are specified in the upper layer. In the top layer, concepts are self-represented, and there is no need for additional upper layers. The Object Management Group (OMG) specifies modelling standards, like BPMN, at layer 2 (meta-model), using modelling constructs of UML Class Diagram. As a consequence, a direct instantiation of the BPMN meta-model leads to the creation of a BPMN model.

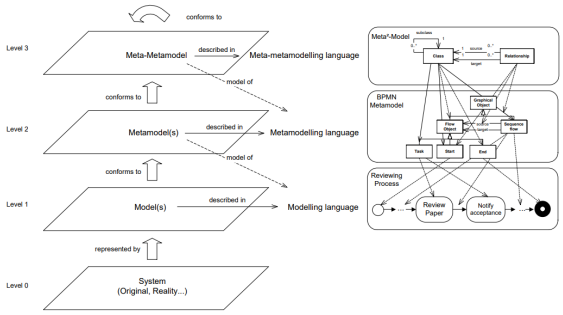

Figure 1: Meta-modelling hierarchy. Adapted from (Efendioglu et al. 2017)

When a meta-model contains concepts that capture a particular domain, context or industry, it can be categorised as a domain-specific modelling language (DSML) (Zečević et al. 2017). Conversely, when a modelling language offers only rudimentary constructs such as "class", "relation", "attribute", any kind of reality can be modelled, thus it can be categorised as a general-purpose modelling language (GPML) (Frank 2010). UML Class Diagram is an example of GPML.

In this work, we define a DSML as a graphical modelling language focused on a particular problem domain that offers expressive power through adequate cognitive notations and abstractions for humans. A DSML in this context serves to visualise, specify, construct and document aspects of an enterprise.

The notion "domain" does not have a fixed definition as it might refer to a paradigm, a business sector, an application area or a single case in an enterprise (Karagiannis et al. 2016). Frank (2010) claims that this term has never been used consistently in conceptual modelling. Also the

difference between "specific-domain", "crossed-domain" and "general purpose" is not clearly defined in literature. A domain can, however, be less narrow - or narrower than others - and this determines the difference between languages that are less or more domain-specific, respectively. Karagiannis et al. (2016) introduce the notion of the "domain-specificity degree", where a higher specificity degree means assimilating concepts in the meta-model that target a more specific domain.

Buchmann (2022) argues that modelling should not follow rigid standards but rather shifting specificities driven by ever-changing needs. That is, domain requirements can be accommodated directly in a modelling language by adapting or extending the meta-model and the graphical notation that is cognitively adequate to a domain expert. The resulting domain-specific models have the advantage of increasing both the understanding of domain experts and modelling productivity because model constructs are tailored to an application domain Frank (2013).

In this work, the primary purpose of DSMLs is to retain enterprise knowledge, which can be used as documentation and/or for knowledge automation. Therefore, literature on DSMLs for software development or DSLs as programming languages is not discussed as considered out of scope.

Karagiannis et al. (2016) suggest creating a DSML starting from one or more modelling standards instead of developing it from scratch. Modelling standards come with sets of proven and well-known concepts with established syntax and semantics, which can be borrowed. Existing meta-modelling architectures and tools like ADOxx[7] and MetaEdit+[8] allow for the adaptation or extension of meta-models for the creation of domain-specific models.

2.2.3 The Domain-Specific Adaptation of Modelling Languages is Yet a Challenge

The *domain-specific adaptation* is a term that refers to the practice of extending or customizing existing modelling languages to tailor them to the needs of an application domain. In this sense, the approach allows increasing the degree of domain specificity of an existing language as domain-specific requirements are incorporated. Karagiannis et al. (2022) contains a collection of domain-specific conceptual models that are based on ADOxx. Despite the availability of tools, the domain-specific adaptation practice is still highly challenging. The identification of an adequate abstraction level of modelling constructs is a complex task (Karagiannis et al. 2016; Wegeler et al. 2013). It requires both language development expertise and domain knowledge, and few people rarely have both Chiprianov et al. (2014), Cho et al. (2012), and Mernik et al. (2005). The domain knowledge mostly resides in users' or domain experts' minds. Therefore, it is a necessity for the language engineer to cooperate with them to extract and make explicit the needed knowledge. Barišić et al. (2018) add that the lack of cooperation with end-users while developing a DSML is likely to cause misinterpretations, which hamper the development process and the quality of the DSML. This can also lead to problems in finding, setting and maintaining a suitable scope for the DSML (Frank 2010).

Existing DSML engineering lifecycles Atkinson and Kuhne (2003), Barišić et al. (2018), Cho et al. (2012), Izquierdo et al. (2013), Kleppe (2008), Selic (2007), Stahl and Völter (2006), and Strembeck and Zdun (2009) share two common characteristics. On the one hand, they have sequential phases: design, implementation and finally evaluation. That is, the implementation cannot start before the conceptualisation has been completed. As well, the evaluation phase cannot start before the DSML has been completely implemented. On the other hand, architectures, approaches, and tools for the design and implementation phases mostly address language engineers, thus making difficult the early inclusion of domain experts.

Both characteristics, (1) the subsequentiality of phases and (2) the non-early inclusion of end-users, show a lack of agility as they violate agile

[7] https://www.omilab.org/adoxx/
[8] https://www.metacase.com/products.html

principles that are reported in software development (Beck et al. 2001) and business development (Burlton et al. 2017).

TBR2: it is an objective of this work to investigate the cause of such a lack of agility and propose a solution. The findings aim to contribute to the ultimate objective of creating an approach that enables the creation and maintenance of EKG schemas.

2.2.4 Related Work

Although relatively young as a practice, there are already numerous scientific works that attempt to combine enterprise models and meta-models with ontologies or knowledge graphs.

Early attempts can be categorized in the so-called *semantic lifting* Hinkelmann et al. (2016a). *Semantic lifting* adds or maps ontological annotations to existing enterprise meta-models and/or models (Azzini et al. 2013; Hrgovcic et al. 2013; Kappel et al. 2006). Some of these works are hereby reported. Opdahl and Berio (2006) propose a Unified Enterprise Modelling Language (UEML) intended as an intermediate language for a wide variety of existing enterprise modelling languages, where each concept is mapped onto a common ontology. The latter has the benefit of semantically anchoring the modelling constructs of the various modelling languages. Similarly, Bräuer and Lochmann (2007) introduce the use of different types of ontologies (i. e., upper, core, domain and application ontologies) to semantically integrate multiple domain-specific languages. Building on the work of Opdahl and Berio (2006), Harzallah et al. (2012) elaborate on the support for ontological analysis of modelling languages. Other works with a focus on ontological analyses of modelling languages and models can be found in (Green and Rosemann 2000; Green et al. 2007; Opdahl and Henderson-Sellers 2002; Rohde 1995; Wand et al. 1999; Zhang et al. 2007). The work of Voigt (2011) maps meta-models into a graph structure to support the matching of different meta-models to ultimately resolve the integration issue across multiple heterogeneous data sources. Fill (2011) adds ontological annotations to business process models to support business process benchmarking. Sandro Emmenegger et al. (2013) annotate the meta-models with enterprise ontology concepts to ultimately detect early warning signals in procurement risk management. Gailly et al. (2017) automatically annotate models using enterprise ontologies. Huang et al. (2016) map UML Class Diagram models into knowledge graph to abstract original class diagrams and generate more hierarchical and refined class diagrams. Similarly, the work described by Panich and Vatanawood (2016) transform UML Class Diagram models and associated sequence diagrams into ontology languages to detect design patterns in an early stage of software design. The work in Hinkelmann et al. (2016b) describes an approach that maps a BPMN meta-model extension into an ontology to then annotate business- and IT-related models with ontology concepts.

As already pointed out in Hinkelmann et al. (2016a), semantic lifting keeps the graphical and ontological representations separate, thus potentially causing inconsistencies between the two if either representation is modified. The alignment of both is a knowledge-intensive task that, if manually done, is prone to errors and requires considerable effort. In real-world scenarios, such an alignment is a frequent activity because it should keep up with the constant changes an enterprise is subject to. While this issue would not affect GPMLs, it does impact DSMLs and domain-specific models because they contain a certain degree of an addressed application domain.

Recent works strive to relieve this alignment effort, mostly by proposing automatic transformation approaches from models to ontology. Some of these works are hereby reported. In S. Emmenegger et al. (2016) different interlinked enterprise models are transformed into an ontology to make automatic recommendations in workplace learning settings. Buchmann and Karagiannis (2016) focus on legacy data sources containing domain-specific graphical models and export them into ontologies to make them queryable in a Linked Data environment. The automatic transformation

is made possible by an ADOxx-to-RDF plugin [9] . A similar approach is discussed in (Karagiannis and Buchmann 2018). Similarly, the plugin Archi-to-Neo4J [10] converts a model in ArchiMate language into a property graph. Compared to this plugin, the work of Smajevic and Bork (2021) provides a more generic approach where any conceptual model can be transformed into a property graph (also in Neo4J). The conceptual models can be created in EMF or ADOxx metamodeling platforms, but also from the Ecore-based modeling platforms Papyrus (for UML, SysML, and UML profiles) and Archi (for ArchiMate). Most recently, Bachhofner et al. (2022) proposed BPMN2KG as a transformation tool from BPMN2.0 process models into knowledge graphs.

Although the automatic transformation of models to ontology relieves the manual annotation issue, the misalignment problem remains because of the two separate knowledge representations (Fig. 2 depicts the issue graphically). This is problematic because whenever one of the two representations changes, actions shall be taken, making the maintainability a challenge.

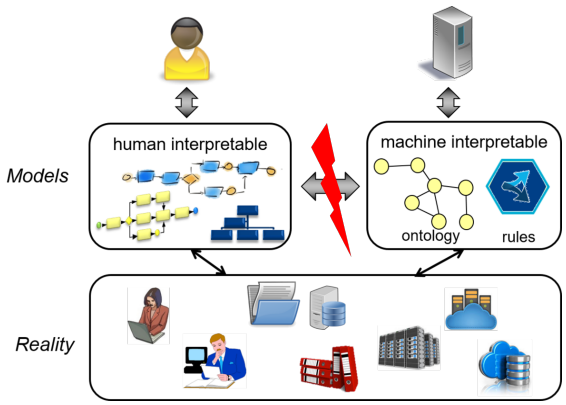

Figure 2: Misalignment between the human- and machine-interpretable knowledge. Adapted from (Hinkelmann et al. 2016a)

OntoUML[11] attempts to address this misalignment issue by incorporating ontological distinctions into modelling primitives, which extend UML. The underlying foundational ontology is based on the Unified Foundational Ontology (UFO) (Guizzardi 2005). In Benevides and Guizzardi (2009), the authors present a model-based tool for ontology-driven conceptual modelling in OntoUML. While these approaches make use of a GPML, this work focuses on the use of DSMLs for the creation of EKGs. In addition, the proposed approach aims to create DSMLs that are specified in an ontology language, ready to be used by inference engines.

TBR3: While creating a DSML, the proposed approach shall ensure a continuous alignment between human-interpretable knowledge (which is retained in the form of modelling languages and models) and machine-interpretable knowledge (which is retained in the form of knowledge graphs). In this way, enterprise knowledge graphs can be not only created but also maintained through enterprise modelling.

3 Analysis, Design Challenges and Requirements

Core findings from the reviewed literature (*TBR3*) point to a prerequisite for the creation and maintenance of enterprise knowledge graphs through enterprise modelling, which is an approach that, while allowing for domain-specific adaptations of modelling languages, ensures alignment between the human- and machine-interpretable knowledge. This approach takes the name of *agile and ontology-based meta-modelling* and was created by following the phases of Design Science Research (DSR) methodology (Vaishnavi and Kuechler 2007): awareness of problem, suggestion, development, evaluation, and conclusion.

The *Problem Awareness* phase dealt with deepening the understanding of domain-specific adaptations through traditional meta-modelling and related challenges (addressing *(TBR2)*), with respect to real-world application domains. Hevner et al. (2004) emphasize the importance of eliciting requirements from an outside environment to ensure the relevance of the final designed artifact. For this, two research sub-objectives were posed:

[9] https://www.omilab.org/adoxx/modules/details/?id=175
[10] https://www.hosiaisluoma.fi/blog/archimate-neo4j/
[11] https://ontouml.readthedocs.io/en/latest/intro/ontouml.html

(1) understanding the need for domain-specific adaptations from the industry and (2) understanding the lack of agility and the misalignment issue caused by the semantic lifting of meta-models and models in real-world scenarios.

3.1 Expert interviews and findings

The *first research sub-objective* was achieved by conducting four semi-structured interviews with four senior consultants and modelling experts. To ensure a high quality of data collection, the interviewees were selected according to the following criteria: senior position; modelling is a daily business activity to address the client's problems; at least two years of experience with enterprise modelling languages in consulting projects; diversity in enterprise modelling expertise, i. e., each expert has a different enterprise modelling focus, e. g., process modelling, enterprise architecture modelling; diversity on the adopted modelling tool, i. e., each expert uses a different modelling tool in project consulting; the consultant's company operates worldwide or has world-wide recognized partners. Each interviewed fell into one of the following roles: enterprise architect, business process modeller, workflow modeller, and enterprise modeller. In total three companies were involved: two experts from the BOC Group[12] , one expert from Camunda[13] and one expert from KnowGravity[14] .

Details about the interview analysis can be found in Laurenzi (2020). Results led to the identification of three levels of complexity of domain-specific adaptations. From levels 1 to 3, the adaptation complexity increases. Namely:

- At **level one**, there is the simplification of a modelling language. That means aspects in the abstract syntax that do not match with the elicited requirements are removed. The abstract syntax refers to the knowledge captured by the meta-model, i. e., modelling elements, relations and attributes (including types and values).
- At **level two**, we find the change of abstract syntax and notation. Changing attribute values is mainly used to restrict the possible range they can get, which is a way to constrain the modelling language. Changing the graphical notation means that it can be replaced with a new one.
- **Level three** adds complexity to level two such that the abstract syntax is extended as well as new graphical notations are provided. That is, the meta-model is extended or constrained, which requires a higher level of expertise than the previous two. The meta-model extension also includes cross-reference relations for connecting concepts from different modelling languages or modelling views.

3.2 Case study analysis and findings

The *second research sub-objective*, was achieved by creating and analyzing two cases. A case study strategy (Yin 2018) was followed to ensure scientific rigour. Each case focuses on one different application domain and was derived from a different research project:

- patient transferal management (PTM), i. e., a digital healthcare application domain), based on the Swiss research project Patient-Radar (E. Laurenzi et al. 2017).
- business process as a service (BPaaS) (Woitsch et al. 2016), i. e., Cloud Computing application domain), based on the European research project CloudSocket[15] .

The meta-modelling tool ADOxx was used in both cases. The focus on ADOxx can be justified with two main reasons: (1) it is one of the most widely used meta-modelling tools in research and for research projects where numerous companies worldwide have been involved; (2) ADOxx has been used for the semantic lifting of meta-models and models in numerous research projects. The

[12] https://www.boc-group.com/en/
[13] https://camunda.com/
[14] https://www.knowgravity.com/
[15] www.cloudsocket.eu - project in the European Union's Horizon 2020 Framework Programm under grant agreement No 644690

generic architecture for the meta-modelling platform was first described in Karagiannis and Kühn (2002) and later further refined in Fill and Karagiannis (2013).

In both cases, domain-specific adaptations of modelling languages were performed to accommodate new domain requirements, along with semantic lifting activities. The AMME engineering lifecycle proposed in Karagiannis (2018) was followed as it embraces agile principles. Fig. 3 depicts the lifecycle (top of the figure) and its instantiation (bottom of the figure) for the BPaaS case.

The detailed description of the analysis of the cases can be found in Laurenzi (2020).

The analysis of the two cases led to the confirmation of a lack of agility in DSMLs engineering approaches, and revealed the below list of seven problems (Fig. 4 shows the position of each problem along the language engineering lifecycle):

- *Problem 1*: Language engineers and domain experts have different types of expertise. This might result in misinterpretation of requirements. As a consequence, the quality of the new version of DSML is hampered. This problem manifests particularly at the beginning of the project, as the language engineers have little knowledge about the addressed domain. Further engineering iterations are required until a DSML is good enough to be released, which is time-consuming.
- *Problem 2*: Extracting, documenting, prioritizing, and categorizing requirements is time-consuming and prevents the quick advance of the engineering lifecycle.
- *Problem 3*: The longer the list of requirements and the more dependencies among requirements, the more time-consuming their maintenance (i. e., synchronization and update).
- *Problem 4*: Aligning requirements from the Create phase to the Design phase is time-consuming.
- *Problem 5*: Starting the Develop phase only after the Design phase is finished is not convenient as changes can arise while implementing the language. One reason for changes is due to the inability of meta-modelling tools to accommodate the desired conceptualisation.
- *Problem 6*: The sequential re-iteration of all engineering phases before a new version of a DSML is deployed is time-consuming. That is, a new version of the language cannot be evaluated until the whole lifecycle is ended.
- *Problem 7*: In the Formalize phase, each new requirement originating from the Design Phase (see *7a*) or Develop phase (see *7b*) leads to adapting the formalized meta-model, which is a tedious, intensive and time-consuming engineering task requiring high expertise.

Additional findings revolve around the use of traditional meta-modelling architecture for the domain-specific adaptations of modelling languages and mapping of meta-models and models to ontology, i. e., semantic lifted DSMLs. Fig. 5 depicts the four main issues, which are described as follows:

1. *Issue 1*: The early inclusion of users (or domain experts) in the development of DSMLs is hindered by the separation of the two components meta-modelling and modelling.
2. *Issue 2*: Long waiting time since a new feedback-based requirement is generated until it is accommodated in the language and tested. A DSML can be tested through the creation of models only after it is deployed (i. e., sequentiality of phases). Usually, from the Deploy/Validate phase, new requirements arise, which we call feedback-driven requirements.
3. *Issue 3*: Performing domain-specific adaptations leads to a loss of the semantic mapping with the ontology.
4. *Issue 4*: Issue 3 propagates to the instance level, causing a misalignment between models and ontology instances. For example, in the BPaaS case, there was the need to adapt the requirements of cloud services frequently, which led

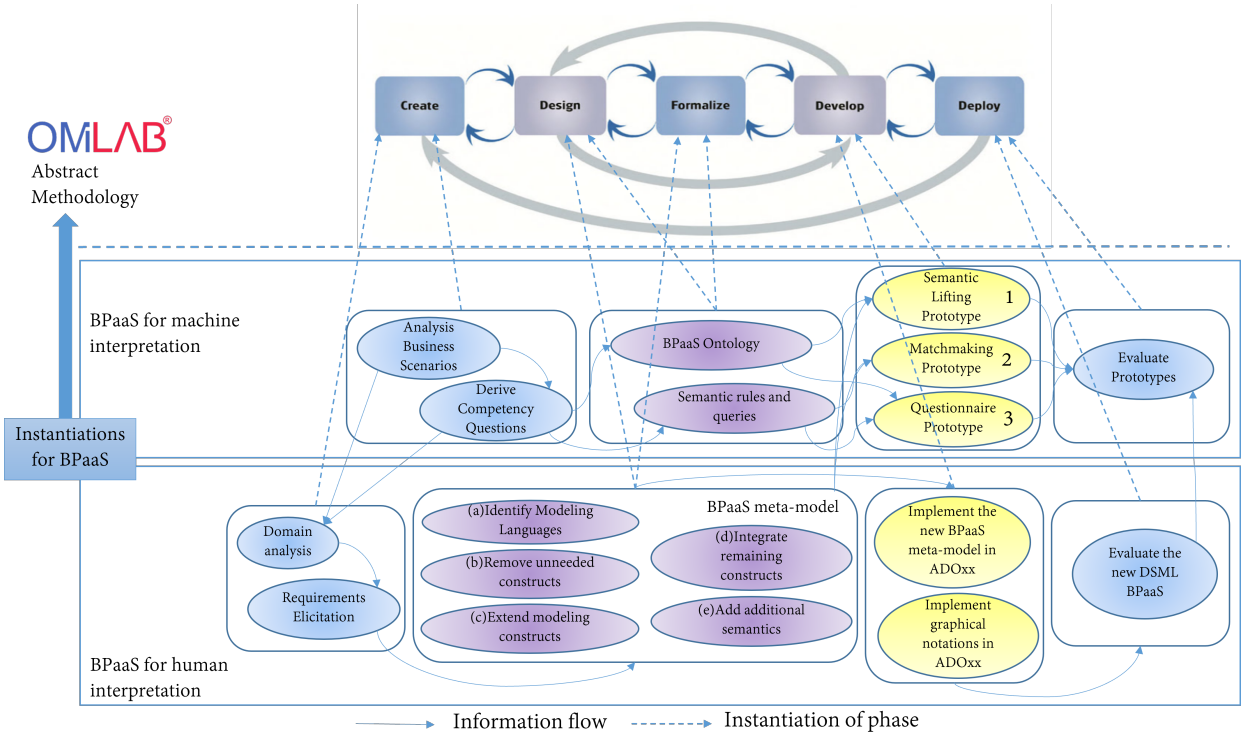

Figure 3: Two AMME Lifecycle instantiations for the human and machine interpretation of BPaaS DSML

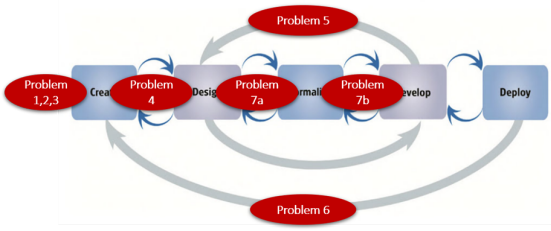

Figure 4: Main problems hindering agility of DSML engineering faced during the development of DSML4PTM and BPaaS DSML

to a constant manual update of the schema in the knowledge graph.

3.3 Main Design Challenges and Requirements

This sub-section presents the two main design challenges and a list of requirements to be fulfilled by the new agile ontology-based meta-modelling approach in terms of domain-specific adaptations and the continuous alignment between the two knowledge representations: the graphical and the ontology one. Each requirement was conceived with a triangulation approach by considering (1) complexity levels derived from the interviews' findings; (2) findings from the two cases; and (3) literature review. The details about the triangulation approach can be found in Laurenzi (2020).

Design challenges:

1. **Design Challenge 1**: The agile and ontology-based meta-modelling approach shall promote tight cooperation between domain experts and language engineers by avoiding the separation between the language engineering and modelling components.
2. **Design Challenge 2**: The agile and ontology-based meta-modelling approach shall avoid sequential DSML engineering phases while performing domain-specific adaptations of modelling languages.

Requirements (req. from 5 to 8 derive from *TBR1*):

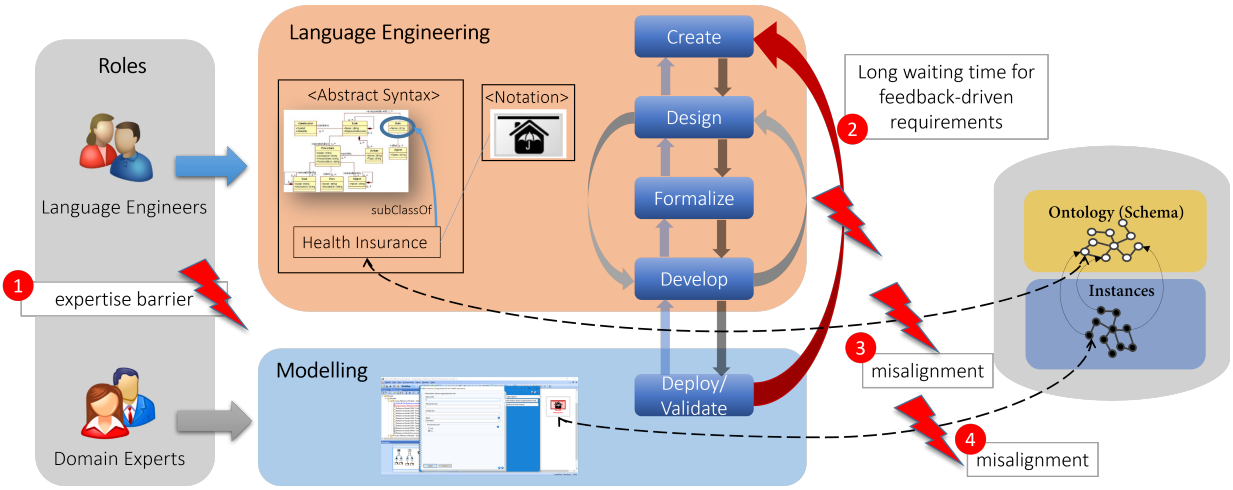

Figure 5: Issues with the traditional meta-modelling architecture when developing semantic lifted DSMLs

1. *Requirement 1*: The agile and ontology-based meta-modelling approach shall enable the language engineer to simplify a modelling language;
2. *Requirement 2*: The agile and ontology-based meta-modelling approach shall enable the language engineer to change abstract syntax and notation;
3. *Requirement 3*: The agile and ontology-based meta-modelling approach shall enable the language engineer to extend abstract syntax and add new notation;
4. *Requirement 4*: The agile and ontology-based meta-modelling approach shall enable the language engineer to integrate concepts that belong to different modelling languages or different modelling views.
5. *Requirement 5*: The agile and ontology-based meta-modelling approach shall enable the language engineer to create new semantic domain concepts.
6. *Requirement 6*: The agile and ontology-based meta-modelling approach shall enable the language engineer to create new semantic mappings between concepts from an abstract syntax (linguistic view) to a semantic domain (domain view).
7. *Requirement 7*: The agile and ontology-based meta-modelling approach shall enable the language engineer to modify the semantic mapping between concepts from an abstract syntax (linguistic view) to a semantic domain (domain view).
8. *Requirement 8*: The agile and ontology-based meta-modelling approach shall enable the language engineer to delete the semantic mapping between concepts from an abstract syntax (linguistic view) to a semantic domain (domain view).

4 Conceptualization of the Agile and Ontology-Based Meta-modelling Approach

The two design challenges were addressed by proposing an agile meta-modelling approach that integrates language engineering and modelling into one component (see right side of Fig. 6). Thus, activities of language engineering, modelling and evaluation can be interleaved and iterated. Such an approach addresses the first challenge as it creates the conditions for the language engineer and domain expert to cooperate tightly while engineering a DSML. To address the second design challenge, language adaptations are performed on the fly. That is, language adaptations are applied in a piecemeal manner as demands arise.

While language engineers apply adaptations to the DSML, domain experts can visualise the results in real time, and thus they can provide immediate suggestions. The sequential approach is therefore avoided as modelling and evaluation activities do not have to wait until a new DMSL is deployed to be performed.

Such an approach also has the benefit of avoiding propagation of changes throughout the whole engineering lifecycle, i. e., Create, Design, Formalise, Develop and Deploy/Validate phases Karagiannis (2018). Instead, new requirements can be directly implemented into the DSML in a manner that results are immediately available for testing. The accommodation of requirements through direct implementation also overcomes some of the problems listed in sub-Sect. 3.2:

- The two problems about the documentation of requirements (see problems 2 and 3);
- The problem about the alignment between the Create and Design Phase (see problem 4);
- The problem about waiting for the design phase to be finished until the implementation starts (problem 5).
- The formalisation of the meta-model can be done through the direct implementation of the requirements into an ontology-based meta-model. Such an approach overcomes problem 7.

4.1 An Example of the Tight Cooperation in the Integrated Component

In the following, an example is described to clarify the idea of an agile meta-modelling. In the example, we assume the request from a physician (i. e., domain expert) to introduce the concept of an acute hospital as a special class of hospital. The language engineer can accommodate such a request by extending the class "Hospital" with a class "Acute Hospital", in the integrated component (see Fig. 7).

The two attributes, name, and address are inherited from the class Hospital. New attributes are then added in the new class, such as a list of main acute treatments and a graphical notation. As soon as the extension is accomplished, the graphical notation of the modelling construct appears among the modelling constructs of the language, which can be selected to be instantiated as a model element. The physician can now visualise the new concept and use it. In this way, models can be created to include the new concept. For instance, the new concept can be integrated into the context of an organisational model which includes roles and employees. Such straight implementation of the concept and its immediate use creates the conditions for the physician to provide immediate suggestions for adaptations, should he or she have them. Eventually, these suggestions lead to a more accurate representation of the concept. For instance, new treatment activities might be added, or existing ones deleted, as well as potential desired changes to the graphical notation. Thus, the language engineer and the domain expert tightly cooperate to carry out engineering, modelling and evaluation activities until a satisfactory version of the DSML is achieved.

4.2 On-the-Fly Domain-Specific Adaptations in the Integrated Component

On-the-fly domain-specific adaptations are actions that shall be performed in the component that integrates language engineering and modelling. For this, the integrated component foresees a palette (see left-hand side of Fig. 8) in which the graphical notations of the DSML are displayed. The palette is conceptualised to provide access to the language engineering component. In the latter, the meta-model can be adapted. Each graphical notation is a gate to the related concept that resides in the meta-model. For example, the left-hand side of Fig. 8 depicts the palette displaying the graphical notations of BPMN. From the BPMN Task one can access the language engineering activities (see orange rectangle in Fig. 8, which displays concepts, relations, attributes and notation about the Task). The language engineering activities shall, thereby, allow for domain-specific adaptations of the knowledge residing in the meta-model. For this, the following three specifications should be affected:

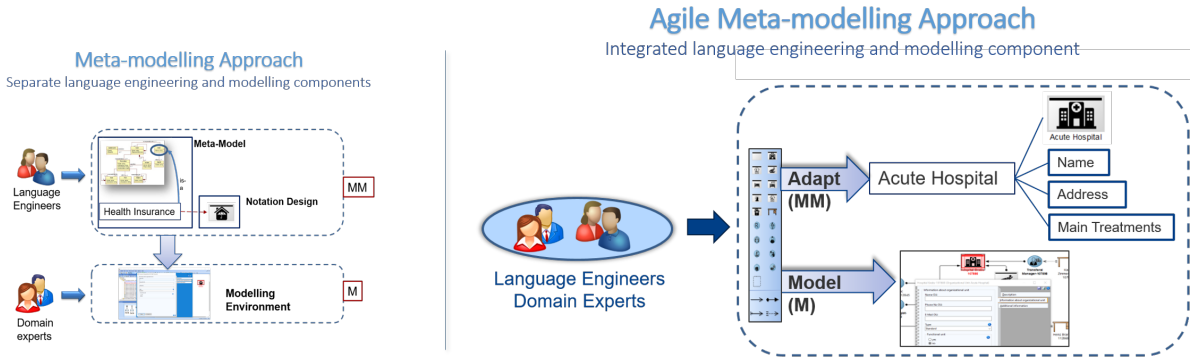

Figure 6: Integration of Meta-modelling and Modelling

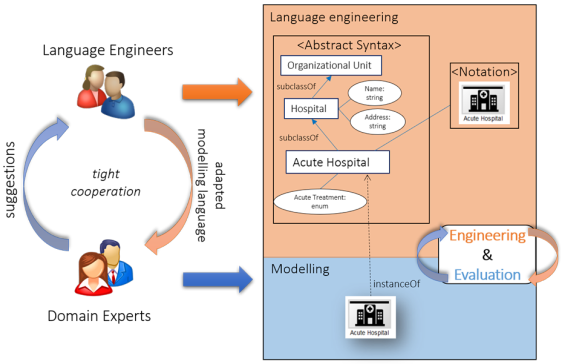

Figure 7: Tight cooperation between language engineers and domain experts

graphical notation, abstract syntax, and semantics. The next sub-section discuss the operators that were derived for the on-the-fly domain-specific adaptations.

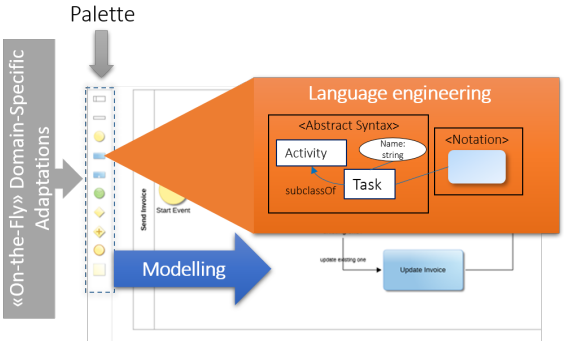

Figure 8: Integration of Meta-modelling and Modelling

4.3 Operators for On-the-Fly Domain-Specific Adaptations

To perform on-the-fly domain-specific adaptations on one or more modelling languages, a list of 10 operators has been identified. The operators allow applying changes and extensions over the specifications of a modelling language. The list of requirements presented in Sec. 3.3 served as a basis for the creation of the operators. The requirements were mapped to the four CRUD functions, which are common in relational database applications: Create, Read, Update and Delete. Each function can map to a SPARQL statement: Create maps to INSERT, Read maps to SELECT, Update maps to UPDATE, Delete maps to DELETE. Specifically, Requirements 3 to 6 are fulfilled by the function Create, requirements 2 and 7 by Update, and requirements 1 and 8 by Delete. There is no requirement mapping to the function Read, which is obvious, as the function does not fit the purpose of changing or extending the modelling language specifications. Each of the three functions Create, Update and Delete, performs one specific domain-specific adaptation, e. g., create a new modelling element as a sub-class of an existing modelling element, create a new sub-class relation between two existing modelling elements, update the name of a class etc. In this sense, domain-specific adaptations affect one of the following basic generic modelling concepts: (1) Class, (2) Relationship, (3) Attribute, (4) Attribute types and values.

The complete list of operators is described below.

1. *Operator 1*: Create sub-class. This operator aims to fulfil three requirements. It allows modelling elements and modelling relations to be extended (requirement 3). It allows the integration of modelling elements (classes) from different modelling languages (requirement 4). It allows extension of concepts in the semantic domain (requirement 5).
2. *Operator 2*: Update class. This operator allows modifying an existing modelling construct (requirement 2), which includes attributes such as name, comment, and graphical notations.
3. *Operator 3*: Delete sub-class. This operator allows the removal of unneeded modelling constructs (i. e., modelling elements and modelling relations) from the abstract syntax (requirement 1).
4. *Operator 4*: Create relation. This operator allows the creation of new relations (i. e., bridging concept) between modelling elements in the abstract syntax (requirement 4) as well as new relations from modelling elements to semantic domain concepts, i. e., semantic mapping (requirement 6). Bridging concepts refers to the relations that connect modelling elements between different modelling languages or modelling views.
5. *Operator 5*: Update relation. This operator allows the modification of the existing relations between modelling elements as well as relations from modelling constructs to semantic domain concept(s) (requirement 7).
6. *Operator 6*: Delete relation. This operator allows the deletion of existing relations between modelling elements as well as relations from modelling constructs to semantic domain concept(s) (requirement 8).
7. *Operator 7*: Create attribute: This operator allows for adding new attributes to modelling elements (requirement 3).
8. *Operator 8*: Assign attribute type or value. This operator allows for assigning types (e. g., String, Integer, Boolean) or concrete values to attributes of modelling constructs. A concrete value is the reference to a graphical notation (requirement 3).
9. *Operator 9*: Delete attribute: This operator allows for deleting existing attributes from a modelling element (requirement 1).
10. *Operator 10*: Update attribute: This operator allows for modifying existing attributes associated to modelling constructs (requirement 2), i. e., the name and the value type of the attribute.

A graphical depiction of each user action and respective operator is shown in Fig. 9. Each operator is triggered by a user action (e. g., a domain-specific adaptation of extending a class with a sub-class) and calls a method for the creation of an update, e. g., INSERT rdfs:subClassOf relation between the two targeted ontology-based modelling elements. Fig. 10 depicts *Operator 1:Create sub-class* and its relation to the user action and the subsequent two updates. The second update INSERT hasParent is used to graphically visualize the dependency between the two graphical notations of the two modelling elements. Such mechanisms allow the specialization of existing concepts on-the-fly for any level of specificity that may be required. In total, 11 updates were created in association with each operator and subsequently implemented in Java methods (Laurenzi 2020).

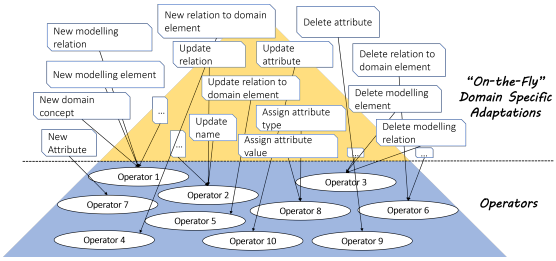

Figure 9: Operators for the on-the-fly domain-specific adaptations of modelling languages

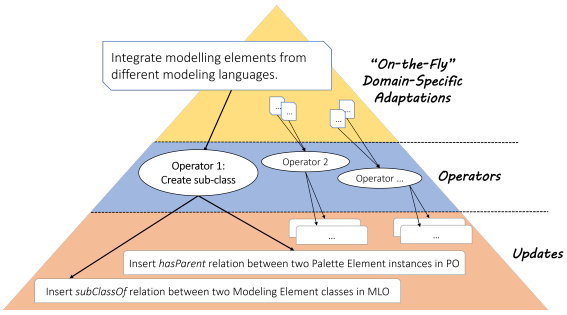

Figure 10: Meta-modelling operators and updates for the ontology propagation

4.4 High-Level Ontology Architecture of an Agile and Ontology-based Meta-modelling Approach

To support the agile and ontology-based meta-modelling approach, an ontology architecture is proposed, which extends the theoretical foundation of the ontology-based meta-modelling (Hinkelmann et al. 2018). The architecture is depicted in Fig. 11. The architecture ensures the separation of concerns among the most relevant aspects of a modelling language, which are:

- Abstract syntax (see def. in Sect. 2.2.2),
- Semantic domain and a mapping between concepts from the abstract syntax to those of the semantic domain (see def. in Sect. 2.2.2),
- Graphical notation (see def. in Sect. 2.2.2).

Respectively, the following three main ontologies were conceptualized:

- Meta-Model Ontology (MMO),
- Domain Ontology (DO),
- Palette Ontology (PO).

As Fig. 11 shows, the three ontologies reside in the ontology-based meta-model layer. The Meta-Model Ontology reflects the abstract syntax, while the Domain Ontology reflects the semantic domain. Concepts from the Meta-Model Ontology are mapped with concepts of the Domain Ontology (i. e., semantic mapping). Concepts in the Palette Ontology represent the graphical notations of the language for the palette, as well as for the graphical models; and are linked to the concepts of the Meta-Model Ontology. There are two different specifications for the graphical notation: one for the graphical models and one for the palette (see violet and yellow arrows pointing to two different notation types "for palette" and "for canvas" in Fig. 11). This distinction allows showing different graphical notations between the palette and a graphical model. More details can be displayed in the graphical notation of a model element than in the corresponding notation in the palette, where typically the space is limited.

The arrows from the model layer to the meta-model layer shown in Fig. 11 are labelled and indicate an instantiation relation.

Instances from the Palette Ontology should be further instantiated for the creation of models to inherit graphical properties. Therefore, an ontology language shall support the knowledge representation of an "instance of an instance". Moreover, every model, like every modelling construct and model element, is different from each other, thus they should be uniquely identified. The Resource Description Framework Schema RDF(S) supports such representational needs and has the benefit of being a W3C standard. RDF(S) is also commonly adopted for the specification of knowledge graphs (or lightweight ontologies), and if required, additional expressivity can be added by the more recent W3C standard Shapes Constraint Language (SHACL) (W3C 2017). For example, SHACL can be used to express the constraints of the BPMN ontology meta-model (every Start Even must have at least one outgoing sequence flow), which are then used to validate BPMN models that are in the form of RDF graphs. For these reasons, the choice of the ontology language fell into RDF(S).

4.5 Specification of the Ontology-based Meta-Model

The description of the ontology-based meta-model is underpinned by Fig. 12.

The Palette Ontology (see blue bubbles in Fig. 12) contains classes, object properties and instances that specify:

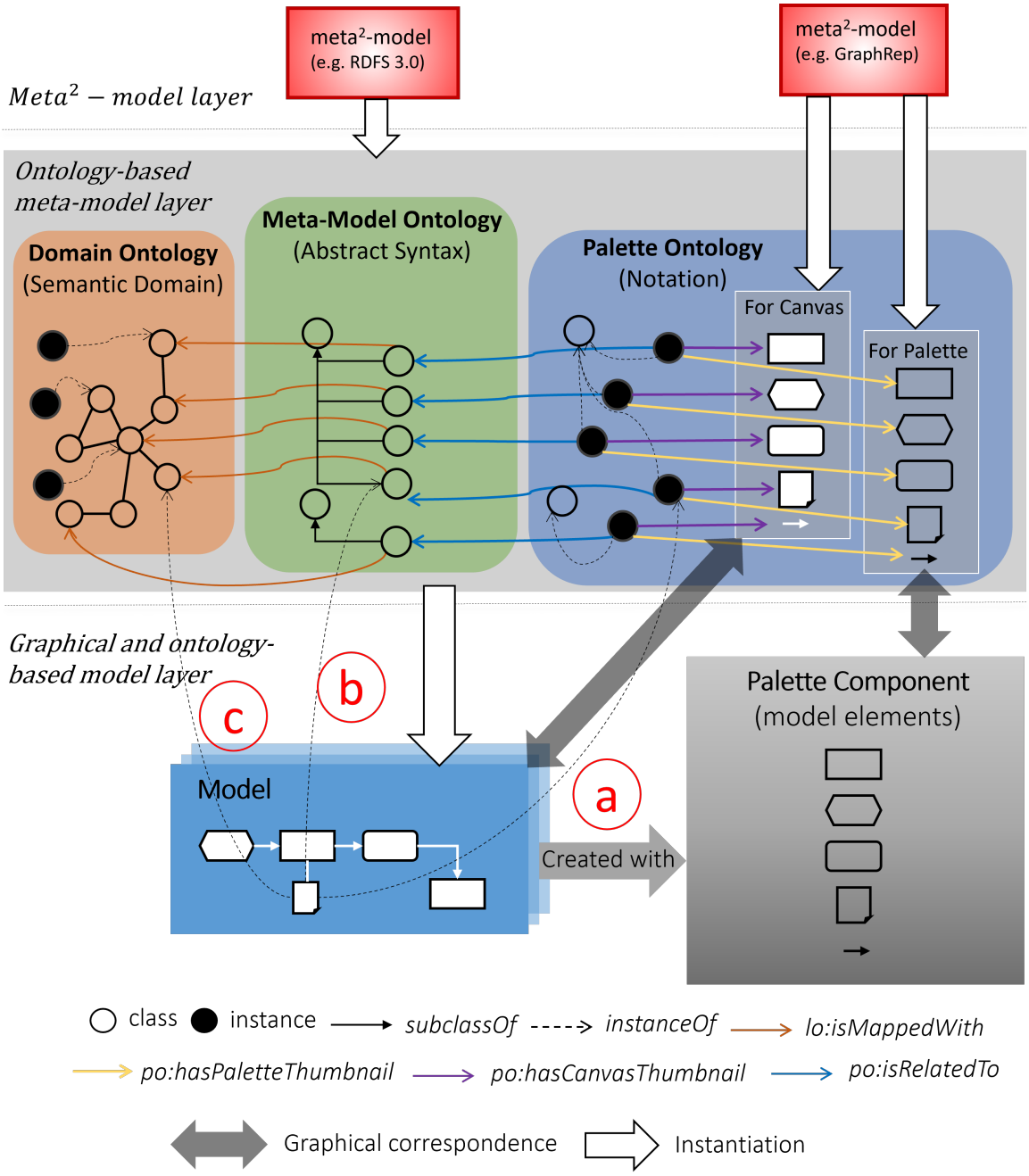

Figure 11: Ontology architecture of AOAME

- the graphical notations (for the palette and for the graphical model) of the modelling language (i. e., *po:PaletteConstruct)*.
- Knowledge for positioning the graphical notations over the palette (i. e., *po:PaletteCategory*).
- In detail, the class *po:PaletteConstruct* class has two sub-classes: *po:PaletteConnector* and *po:PaletteElement*.

po:PaletteConnector contains instances reflecting connectors of one or more modelling languages (e. g., message flow and sequence flow for BPMN), while the class *po:PaletteElement*

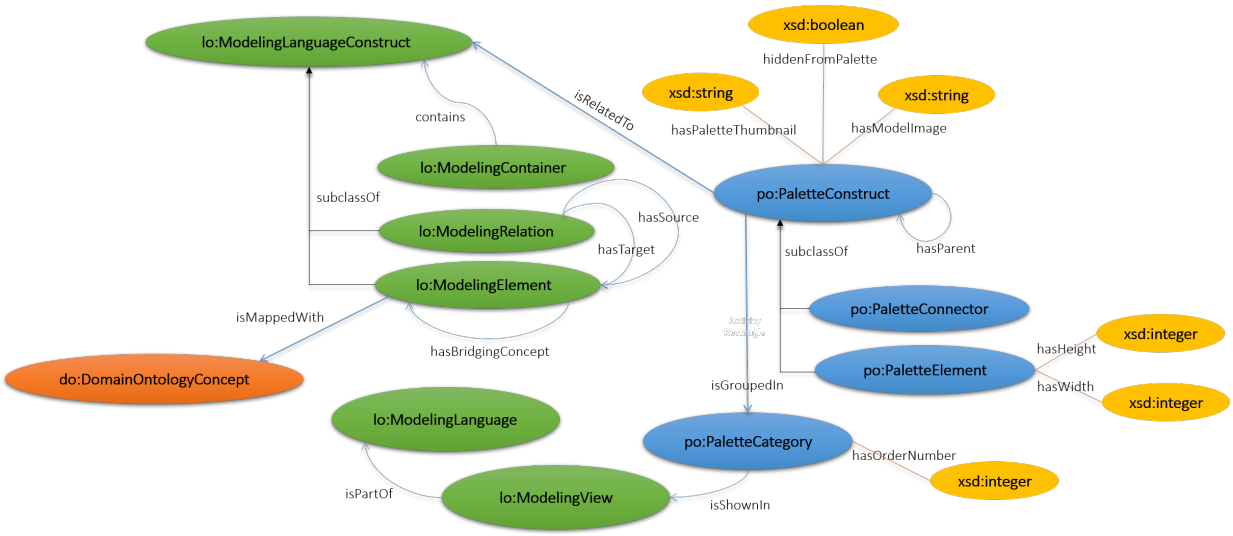

Figure 12: Root concepts and relations of the Domain Ontology, Meta-Model Ontology and Palette Ontology

contains instances reflecting modelling elements of one or more modelling languages (e. g., task or data object for BPMN). Instances from both classes inherit three datatype properties:

- *po:paletteConstructIsHiddenFromPalette*, which is associated with a Boolean datatype property. A "true" value means that the graphical notation for the palette will not be displayed in the palette, and a "false" value means that it will be displayed. The default value for each instantiation is "false".
- *po:paletteConstructHasPaletteThumbnail*, which is associated with a string datatype property that will contain the Uniform Resource Identifier (URI) of a graphical notation to be shown in the palette.
- *po:paletteConstructHasModelImage*, which is associated with a string datatype property that will contain the Uniform Resource Identifier (URI) of a graphical notation to be shown in a graphical model

The class *po:PaletteElement* has two datatype properties:

- *po:paletteConstructHasWidth*, which is associated with an integer datatype property that will contain the default value of the width of the graphical notation for the model;
- *po:paletteConstructHasHeight*, which is associated with an integer datatype property that will contain the default value of the height of the graphical notation for the model.

Note that the two datatype properties do not apply for modelling relations, thus they are inserted in the palette Element class. The two properties do not regard the size of the thumbnail to be displayed in the palette. The rationale is that all the thumbnails in the palette should have the same fixed width and height in order to have a homogenous appearance. Hence, they should be fixed values, which will be hardcoded in the user interface and will be used for any new thumbnail. Conversely, the image size to be shown in the model may vary from element to element. Although the two datatype properties are meant to store default values, they may be subject to change depending on the taste of the modeller or imposed modelling conventions, thus they should not be hard-coded.

Also, the class *po:PaletteConstruct* has five object properties:

- a self-relationship *po:palette Construct Has Parent Palette Construct*. This relation determines

the hierarchy among modelling constructs with the purpose to show it in the palette.

- *po:palette Construct Is Related To Modelling Construct* pointing to the class *lo:Modelling Language Construct*. This object property connects instances of the classes *po:Palette Connector* or *po:Palette Element* (i. e., graphical notations) with classes of the Meta-Model Ontology (i. e., abstract syntax).
- *po:palette Construct Is Grouped In Palette Category* pointing to the class *po:Palette Category*. This object property specifies which Palette Constructs are grouped into which category.
- The class *po:Palette Category* has the purpose of grouping graphical notations of similar types into categories. When a modelling language or DSML presents many concepts that are to be shown in the palette, grouping them into categories is a good solution to avoid cognitive overload. e. g., the modeller Bizagi[16] foresees several categories to group graphical notations in the palette, e. g., for BPMN the palette it shows five categories: (1) flow elements, (2) connecting objects, (3) data (4) swimlanes, and (5) artifacts.

The *po:Palette Category* has the following two datatype properties and one object property:

- *po:palette Category Has Order Number*, which is associated to an integer datatype property that will contain a number. The number will be used for ordering the categories from top to bottom, where the value 1 corresponds to the top element in the palette.
- *po:category Is Shown In Modelling View*, which points to the class *lo:Modelling View*. This object property binds a category to a modelling view, so that for each selected modelling view the correspondent categories are shown in the palette. In turn, since each Palette Construct has its category, the latter is populated with the graphical notations of the constructs. The same category can be shared by two different modelling views. This is the case of the modelling language BPaaS, which since extends BPMN, it also shares the same process modelling view.

The Meta-Model Ontology (see green bubbles Fig. 12) contains classes, taxonomy of classes, and properties (i. e., relations and attributes) describing the abstract syntax of a modelling language. As shown in Fig. 12, the Meta-Model Ontology is based on five classes:

- *lo:Modelling Language* – this class specifies the modelling language.
- *lo:Modelling View* – this class specifies the views of a modelling language. One or more modelling views comprise a modelling language. This knowledge is captured by the object property isPartOf between the two classes *lo:Modelling View* and *lo:Modelling Language*.
- *lo:Modelling Language Construct* – this class generalises the concepts modelling elements and modelling relations.
- *lo:Modelling Element* – this class is a sub-class of *lo:Modelling Language Construct* and specifies the modelling elements of a language. This class has two object properties: *isMappedWith* and *hasBridgingConcept*. The object property *isMappedWith* reflects the formal explication of the semantic mapping. This connects elements from the Meta-Model Ontology to those in the Domain Ontology. The self-relation of the Modelling Element class *hasBridgingConcept* indicates that a modelling element has a bridging concept targeting a modelling element from either the same modelling language (but different modelling view) or from a different modelling language. This relation allows the user to navigate between two elements when they are instantiated in the model.
- *lo:Modelling Relation* – this class is a sub-class of *lo:Modelling Language Construct* and specifies the modelling relations of a modelling language. In a model, each instantiated relation has a source and a target modelling element. For example, a sequence activity in a BPMN

[16] https://www.bizagi.com/en

model may have a start event as a source element and a user task as a target element. This knowledge representation is captured with the two object properties *hasSource* and *hasTarget*, which point to the class *lo:Modelling Element*.

The Meta-Model Ontology imports one or more modelling languages. Each concept has the prefix of the language it belongs to, e. g., Task in BPMN is shown as a class *bpmn:Task*. The taxonomy of classes supports the inheritance of properties from a class to its sub-classes. In ontology languages, like RDF(S), the inheritance mechanism is supported when specifying the rdfs property subClassOf between classes, where the sub-class inherits the properties specified in the class. This is convenient when extending a modelling construct as properties do not need to be re-created. Also, the inheritance mechanism applies from the first defined class to the last sub-class. Therefore, when creating an instance of a class (i. e., a model element), all the properties that were defined in the class and the above super-classes are inherited, creating the conditions for a modeller to specify the properties.

The Domain Ontology Concept (see left-hand side of Fig. 12) is the root concept for domain ontologies containing classes and properties that describe the semantic domain. As already mentioned, the semantic domain is independent of the abstract syntax of a language and describes a domain of discourse. Domain ontologies are, therefore, existing ontologies that are imported to further specify a language construct. Domain ontologies are typically aligned with Upper Ontologies, which are also known as Top Level Ontologies, containing a general semantic level, i. e., general terms like events, time, and location. Examples of such ontologies can be found across the literature base (Chavula and Maria Keet 2015). Similar to the work in Sandro Emmenegger et al. (2013), where enterprise domain concepts are extended with general concepts, the Domain Ontology in this work extends to general and language-independent concepts like top-level elements. Thus, both concepts from a Domain Ontology and concepts extended from the Domain Ontology can be mapped with concepts from the Meta-Model Ontology. The extension of the Domain Ontology is left to an ontology engineer.

In order to support the interlink between the three ontologies in the ontology architecture, the Palette Ontology imports the Meta-Model Ontology, which in turn imports Domain Ontology.

5 Implementation of the Agile and Ontology-Based Meta-modelling Approach

The approach is implemented in the modelling tool AOAME, and is publicly available[17] . Fig. 13 depicts the high-level three-tier software architecture of AOAME. Apache Jena Fuseki[18] is a SPARQL server providing the triplestore. The web service is a JEE application. The graphical user interface (GUI) is an Angular[19] application integrating the canvas library written in JavaScript, GoJS library[20] .

The triplestore contains the Meta-Model Ontology, the Palette Ontology and the Domain Ontology. The Domain Ontology can be extended with external ontologies or graphs. Palette elements or connectors from the Palette Ontology have relationships with classes in the Meta-Model Ontology, which in turn have relationships with the classes in the Domain Ontology. The three ontology files can be found in GitHub[21] .

The GUI shows two main components: (a) the palette (left-hand side of the GUI in Fig. 13) in which the graphical notations of an ontology-based meta-model are displayed; (b) the model editor (right-hand side of the GUI in Fig. 13) in which models can be designed by selecting the graphical notations from the palette.

The web service (see top of Fig. 13) processes all the incoming and outgoing requests and manipulates data between the GUI and the triplestore.

[17] https://aoame.herokuapp.com/
[18] https://jena.apache.org/documentation/fuseki2/
[19] https://angular.io/
[20] https://gojs.net/
[21] https://github.com/manulaur/Ontology4ModelingEnvironment

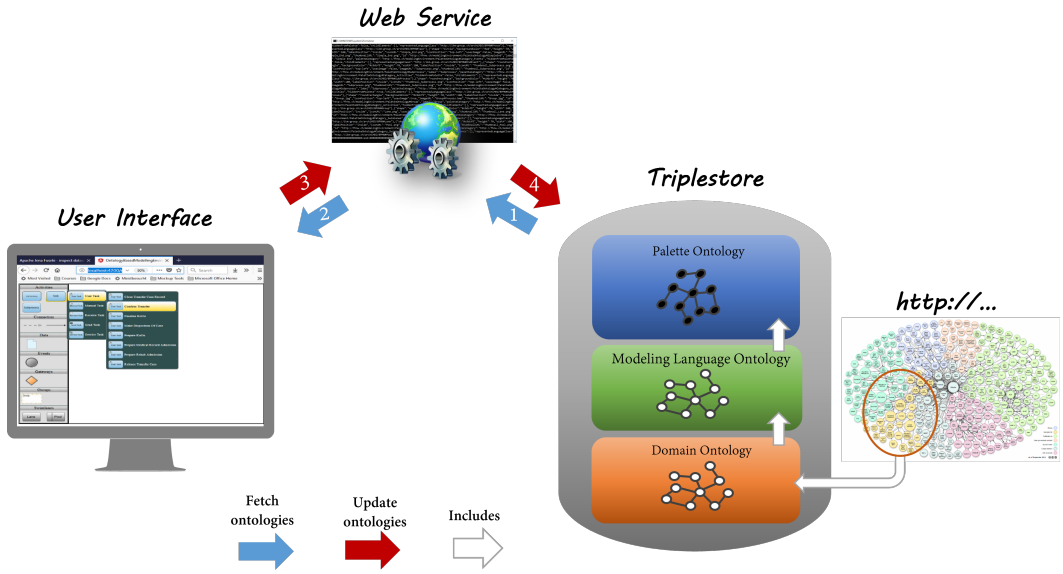

Figure 13: Software architecture of the prototypical tool AOAME

Java libraries of Fuseki have been used for the development of two sets of APIs. One set of APIs fetches the ontologies from the triplestore and display them in the GUI, in a visual form (see blue arrows 1 and 2 in Fig. 13). The other set of APIs implements the meta-modelling operators (see sub-Sect. 4.3), thus allowing the generation of SPARQL statements and subsequent update of the triplestore (see red arrows 3 and 4 in Fig. 13).

The presented software architecture, therefore, supports the achievement of the continuous consistence between the human- and machine-interpretable knowledge of modelling languages, while being subject to domain-specific adaptations.

The domain-specific adaptations are enabled by four main features that were implemented in the palette component, which are described as follows:

- **Feature 1.** Extending a Modelling Construct: this feature allows for the extension of both the modelling constructs and properties.
- **Feature 2.** Editing a Modelling Construct: this feature allows for the editing of both modelling constructs and properties and for the removal of properties.
- **Feature 3.** Hiding a Modelling Construct: this feature allows hiding modelling constructs from the palette.
- **Feature 4.** Deleting a Modelling Construct: this feature allows for the removal of modelling constructs.

For space reasons, only Feature 1 is presented. As shown in Fig. 14, the view consists of a pop-up window, which appears after the selection of the extension feature (see red arrow) from the context menu, e. g., Extend User Task. The pop-up shows fields for and from the ontology-based meta-model.

The description of the remaining features can be found in Laurenzi (2020).

6 Evaluation of the Agile and Ontology-Based Meta-modelling Approach

Tab. 1 shows the evaluation strategy for the Agile and Ontology-Based Meta-modelling Approach.

The prototype method type is used for both an artificial and a naturalistic evaluation (Pries-Heje et al. 2008). The artificial evaluation form aims to evaluate the correct design of the approach against the previously defined requirements. The

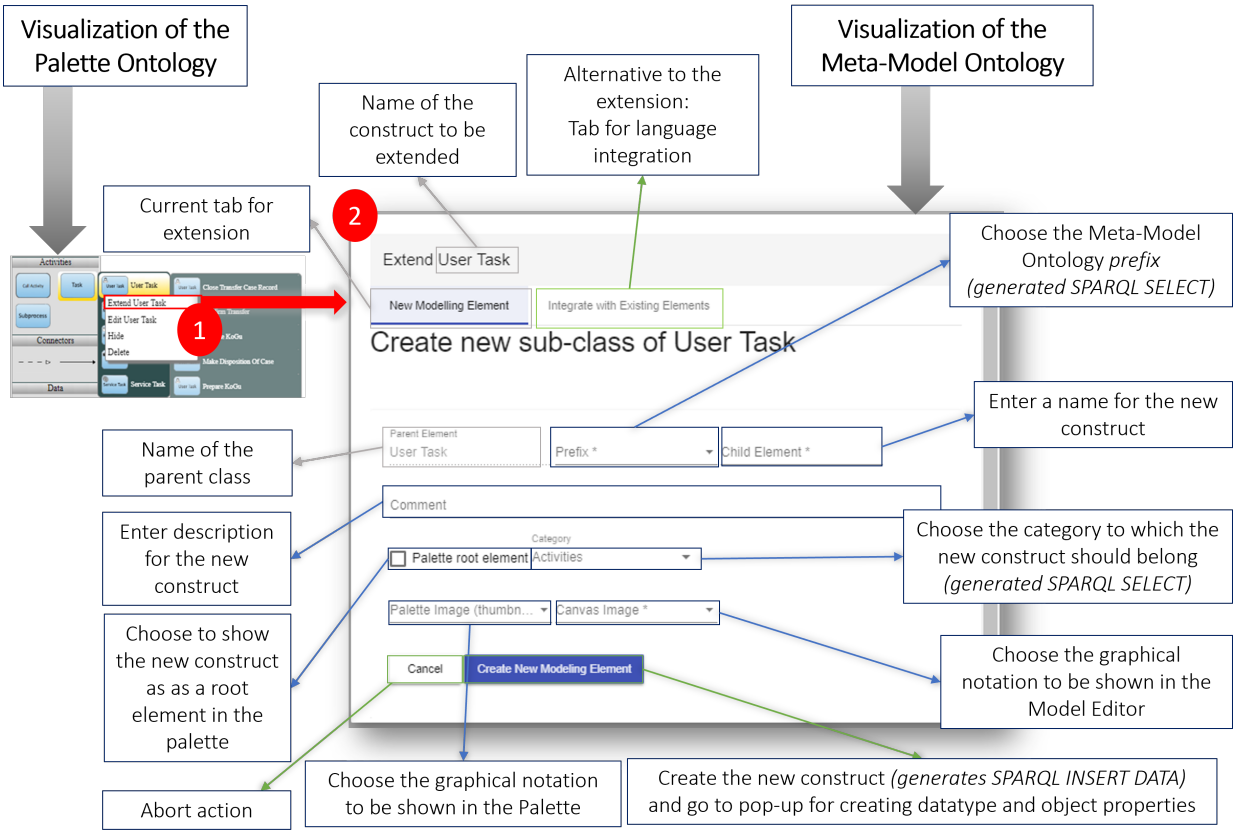

*Figure 14: Visualization of "**Feature 1.** Extending a Modelling Construct" in the GUI of AOAME*

Table 1: Evaluation strategy for the agile and ontology-based meta-modelling approach

Dimension	Characteristic values		
What to evaluate	The agile and ontology-based meta-modelling approach		
When to evaluate	Ex post *(Eval3)*		
How to evaluate	Evaluation form	Naturalistic	Artificial
	Evaluation method	Prototype and illustrative scenario	Prototype
	Evaluation purpose	Utility of the artifact / Purpose and scope	Correct design of the artifact
	Artifact type	Method	
	Evaluation criteria	Operationability of the approach / Generality of the approach	AOAME requirement fulfillment
	Measurement Evaluation Approach	Qualitative	

naturalistic form is used to evaluate the utility of the approach. Additionally, the illustrative scenario method type is considered for the evaluation of the utility of the artifact. Namely, real-world use cases are proposed to be implemented in the prototype. The considered evaluation criteria are operationality and generality, which are both for the purpose of utility. Hence, the two evaluation criteria fit both the artifact type “method” and the evaluation activity chosen in the “where dimension”: Eval3. The two evaluation criteria are specified and contextualised to fit this research work:

- Operationability of the approach: The ability of the approach to preserve continuous alignment between the graphical and the machine-

interpretable representation while performing on-the-fly domain-specific adaptions of modelling languages.

- Generality of the approach: The ability of the approach to be applied in different application domains.

To evaluate the correct design of the approach against the requirements, the criteria are contextualised as follows:

- the extent to which the requirements are satisfied by the implemented approach AOAME.

Finally, a qualitative approach is adopted to measure the artefact with respect to the proposed evaluation criteria. Hence, understanding the extent to which operationability, generality and design requirements are satisfied and descriptively emphasised.

6.1 Evaluation of the Correct Design of the Artifact

The evaluation is presented by elaborating on the basis of each of the eight requirements introduced in sub-Sect. 3.3. The description is underpinned by Fig. 15, which provides a visual traceable map between the requirements (see top of the figure) and the implemented functionalities in AOAME (see bottom of the figure). The different layers from top to bottom reflect the gradual steps adopted in this research work. After the elicitation of eight requirements, ten operators were conceived, accordingly (see mapping between requirements and operators). Subsequently, each operator is implemented by at least one SPARQL statement. The latter allows propagating domain-specific adaptations from the graphical to the machine-interpretable representation of the modelling language.

For space reasons, only the traceability map of one requirement is described.

- **Requirement 3**: An agile and ontology-based meta-modelling approach should enable the language engineer to extend abstract syntax and add new notation.

Requirement three is satisfied by two functionalities of *Feature 1: Extending Modelling Construct*. These are described as follows:

(1) *Extending Modelling Construct* incorporates *SPARQL 2*, which allows the creation of new modelling constructs as a specialization of existing ones. The new modelling construct includes a new name (i. e., Uniform Resource Identifier), label, comment and graphical notation. *SPARQL 2* supports *Operator 1 – create sub-class* and *Operator 8 – Assign concept, attribute type or value*.

(2) *Creating New Datatype Properties* incorporates *SPARQL 4*, which allows the creation of new datatype properties for a modelling construct. *SPARQL 4* supports *Operator 7 – create attribute*.

6.2 Evaluation of the Operationability of the Artefact

For this evaluation activity, six use cases were implemented through the prototype AOAME. The use cases are chosen based on two criteria: (1) validation of all the AOAME's functionalities; (2) significance in real-world applications. The validation of all functionalities proves the operationability of the agile and ontology-based meta-modelling approach. All AOAME's functionalities and the six use cases are described in Laurenzi (2020). The center of Fig. 16 depicts the six use cases, and the used functionalities are on their right- and left-hand sides. The arrows indicate which functionality is used in which use case.

For space reasons, only the third use case scenario is described, which is derived from the patient transferal management case (see sub-Sect. 3.2).

In this use case, the transferal manager needs to document the medical information of patients according to the International Classification Of Functioning Disability Health, i. e., ICF Standard[22] . Such medical information is required to be in the cost reimbursement request, which must be sent to the health insurance provider. A request that

[22] https://www.who.int/standards/classifications/international-classification-of-functioning-disability-and-health

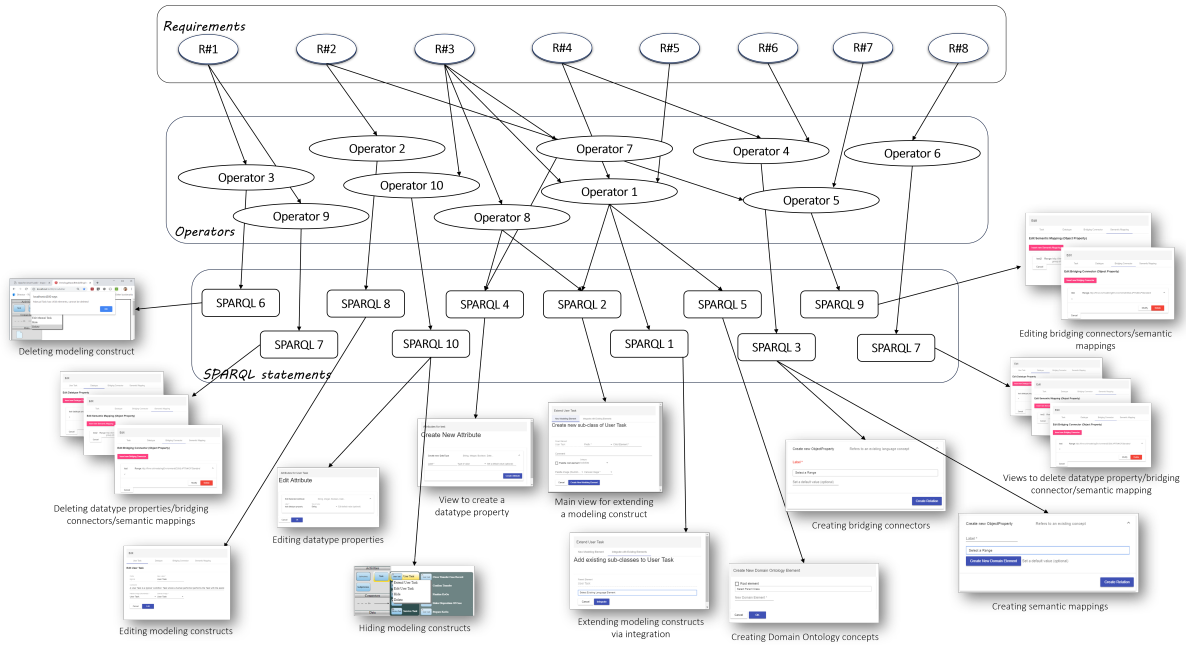

Figure 15: Requirement mapping for the correct design of the artifact

includes medical information of patients that does not comply with the standard is rejected. Therefore, the transferal manager needs quick access to such medical information while preparing the request for the patient's case. The ICF Standard is a specific document, and as such, it is added as a specialisation of the concept Data Document. The latter belongs to the Document and Knowledge Modelling View of the DSML4PTM (i. e., Domain Specific Modelling Language for Patient Transferal Management) (E. Laurenzi et al. 2017) (see bottom of Fig. 17). On the other hand, the preparation of the cost reimbursement request is a BPMN User Task called 'Prepare KoGu', i. e., KoGu is the acronym of cost reimbursement in German language. The request itself is modelled with a data object called KoGu Data Object, which is an input for the task Prepare KoGu. Both the user task and the data object belong to the Process Modelling View of DSML4PTM (see upper part of Fig. 17). Next, given the need for the transferal manager to quickly access medical information, a bridging connector "is part of" is added between the ICF Standard document and the KoGu Data Object (see Fig. 17). In this use case, we assume that both the ICF Standard and the connection to the KoGu Data Object are not yet available from the DSML. Thus, domain-specific adaptations are to be performed.

The AOAME's functionalities "Extend modelling construct", "Create semantic mapping", "Create bridging connector", and "Created datatype property" are used to implement the third use case. Use Case 3 is from the Patient Transferal Management case (see sub-Sect. 3.2).

Fig. 18 depicts the conceptual solution of this use cases both (a) before and (b) after the domain-specific adaptations, which are on top and bottom of the figure, respectively. The available elements and relations before the language adaptation are: *dkmm:DataDocument* from the Document and Knowledge Meta-Model (DKMM); *dsml4ptm:KoGuDataObject* from DSML4PTM, and the parent class *bpmn:DataObject* from BPMN. The concept *dsml4ptm:KoGuDataObject* extends the *bpmn: Data Object*. Medical information regarding the ICF Standard is also available. They already exist in the ICF Ontology. Since this contains domain knowledge and is not a language per se, it is imported and integrated into

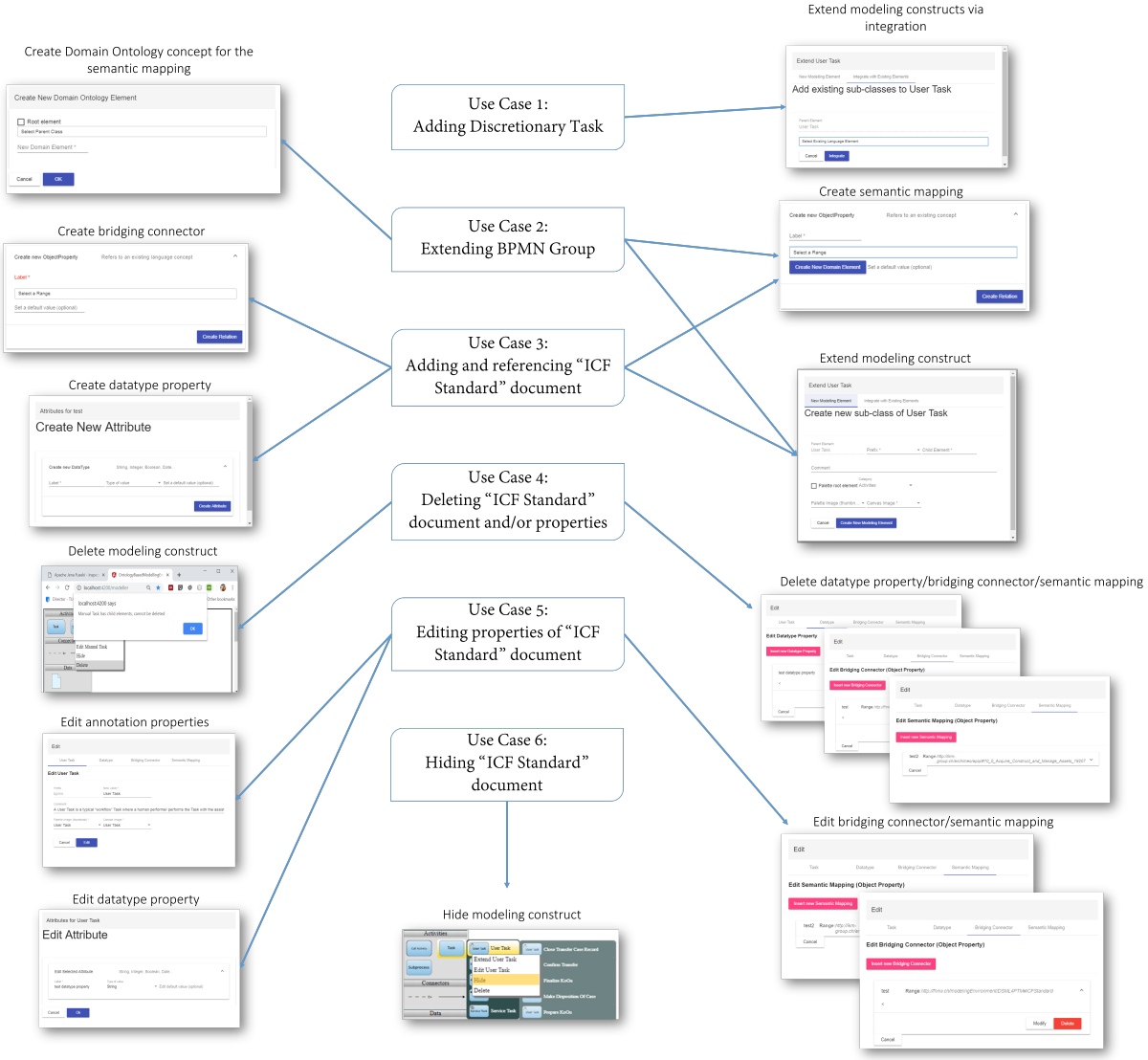

Figure 16: Use cases implemented through AOAME's functionalities

the Domain Ontology. Further properties could also be considered in the ICF Standard, e. g., the patient progress status as well as general concepts like the physical location, which is defined in the Top Level Ontology. However, they are left out to avoid stretching too much the use case.

The bottom of Fig. 18 shows the expected result after performing the domain-specific adaptations. The result contains the following new resources.

The set of ontologies required to implement Use Case 3 is the following:

- The BPMN Ontology.
- The DKMM Otology, which contains the class hierarchy, attributes and relations of the Document and Knowledge Meta-Model (DKMM).
- The DSML4PTM Ontology, which contains the class hierarchy, attributes and relations of the DSML4PTM. DSML4PTM extends, among others, BPMN and DKMM.
- The Meta-Model Ontology. The three language ontologies BPMN and DKMM and DSML4PTM are imported and integrated in

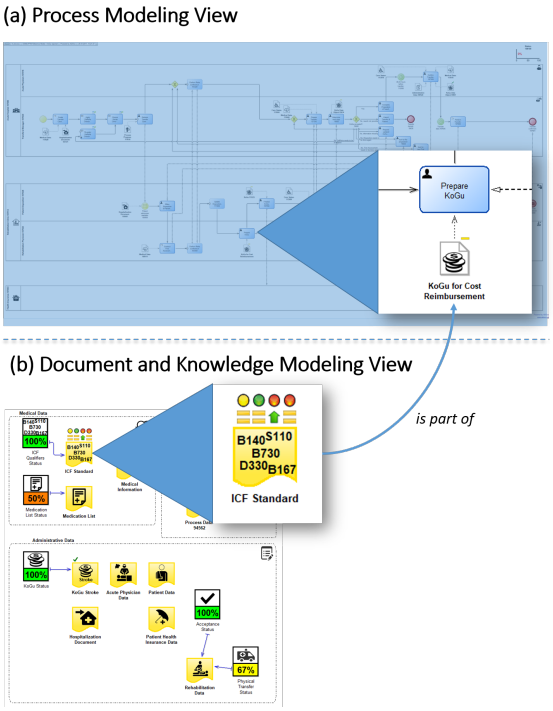

Figure 17: Bridging connector 'is part of' between the ICF Standard document and KoGu Data Object

the Meta-Model Ontology. Modelling elements from the three ontologies are entered as sub-classes of *lo:Modelling Element* while modelling relations are entered as sub-classes of *lo:Modelling Relation*. The modelling language DSML4PTM has several modelling views, i. e., process modelling view, document and knowledge modelling view, organisation modelling view, Decision Modelling View and the control element modelling view. The green quadrant in Fig. 19 shows an excerpt of the Meta-Model Ontology containing concepts of both DKMM and BPMN.

- An excerpt of the ICF Ontology, which contains a few concepts of the National Centre for Biomedical Ontology.
- The Domain Ontology. The ICF ontology is imported into the Domain Ontology. The orange quadrant in Fig. 19 shows an excerpt of the Domain Ontology containing concepts of the ICF Standard.
- The Palette Ontology. The two classes *po:Palette Connector* and *po:Palette Element* contains the instances for displaying the graphical notations of BPMN. Such instances are already linked to the respective classes in the Meta-Model Ontology. The blue quadrant Fig. 19 shows an excerpt of the Palette Ontology, which contains two palette elements *po:Data Document* and *po:Data Objects*. As the figure shows, the two instances are linked with the homonym classes, which belong to the modelling languages DKMM and BPMN, respectively.

This set of ontologies is uploaded to the triple-store in order to populate the palette. The selected modelling language for the palette is DSML4PTM (i. e., Domain Specific Modelling Language for Patient Transferal Management), and the selected modelling view is "Document and Knowledge Modelling View".

Fig. 20 depicts the user actions that are performed to implement Use Case 3, where step 12 shows the final result.

The query result shown in Fig. 21 proves that the machine-interpretable representation of the modelling language is consistent with the graphical representation. Therefore, the functionalities of Feature 1, "Creating New Bridging Connectors" and "Creating New Datatype Properties", are validated.

6.3 Evaluation of the Generality of the Artifact

The generality evaluation criteria refers to the ability of the proposed agile and ontology-based meta-modelling approach to be applied in different application domains. For this, the approach is implemented to support the innovation process of Design Thinking (Emanuele Laurenzi et al. 2020). The approach is also extended with ontology-based case-based reasoning for the retrieval of successful business model canvases, where the ontology-based canvases would be modelled in AOAME (Peter et al. 2020). In Mancuso and Laurenzi (2023), the approach has been used to

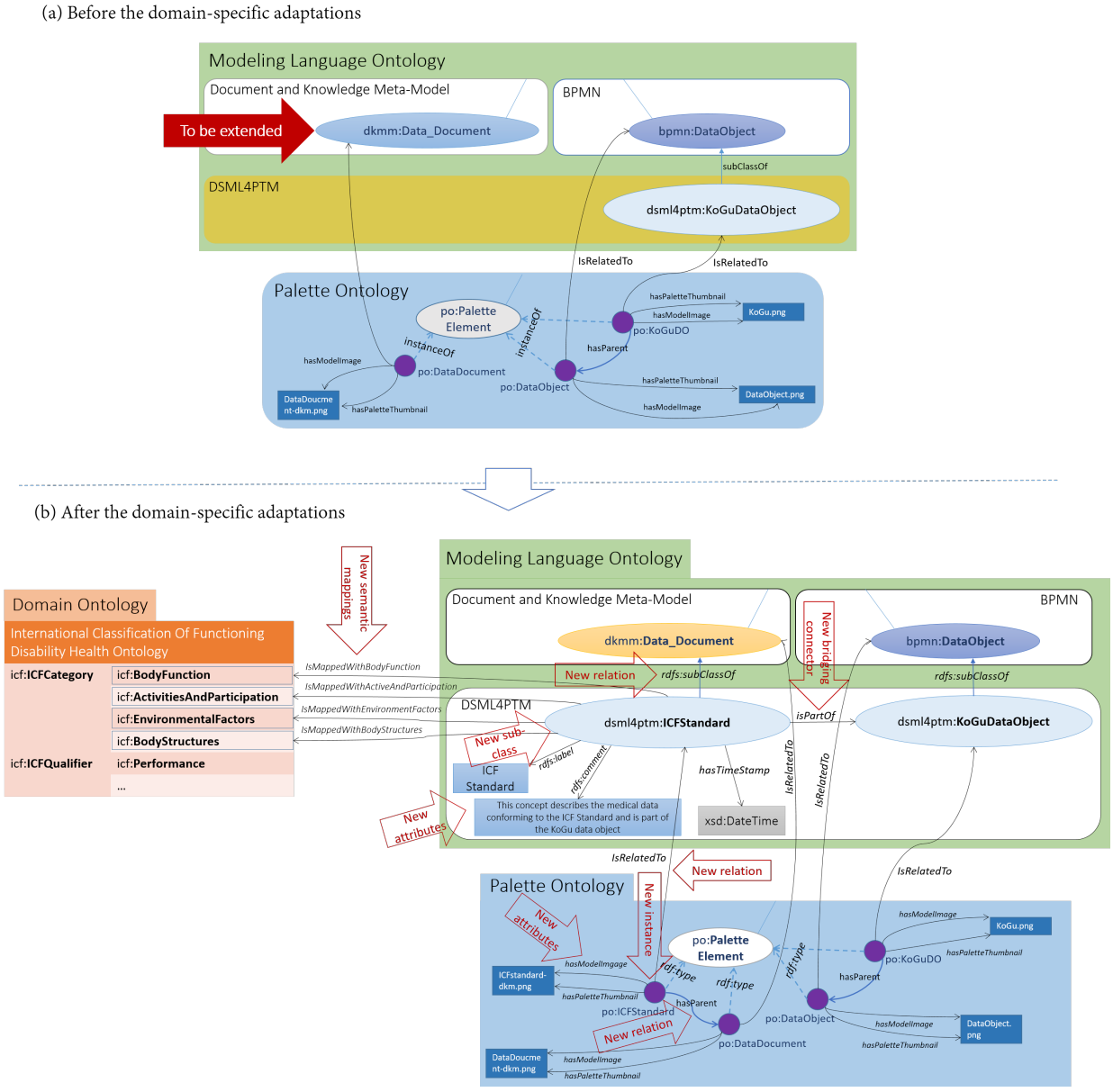

Figure 18: Conceptual solution (a) before and (b) after adding and referring ICF Standard document

create ontology-based user stories to support the Scrum methodology.

7 Limitations of the approach

An open issue of the presented research approach is the possible inconsistencies that meta-modelling operators can cause to existing models. This might occur when an adaptation of a DSML deletes modelling elements that are already used in existing models. Currently, we deal with it in two ways: (1) a functionality export both a model and related ontologies at any time; (b) unless a class is not a terminal node of a meta-model (i. e., the last sub-classes in the taxonomy) it cannot be deleted from the GUI.

Another limitation concerns those projects where a new ontology-based meta-model needs

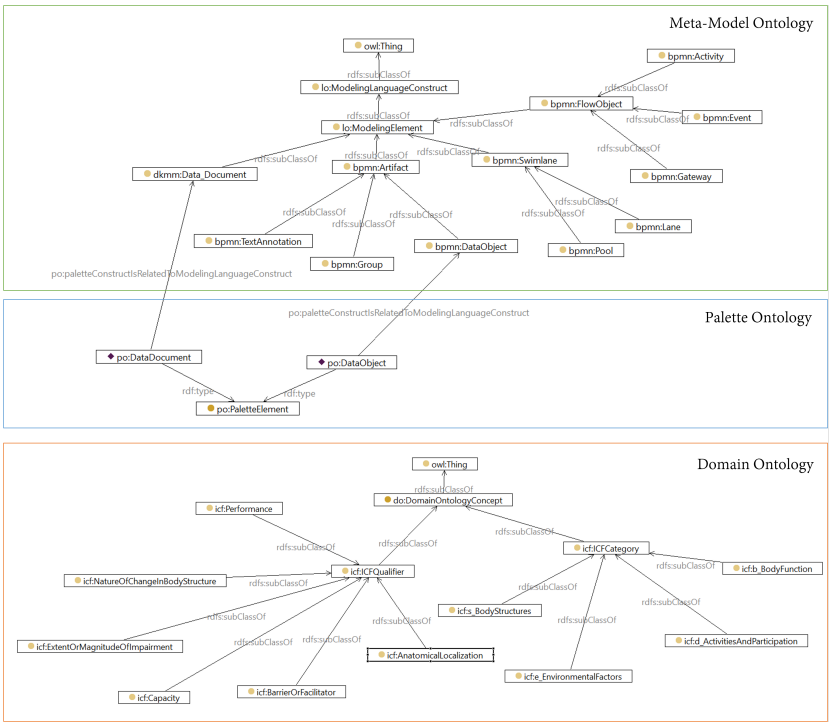

Figure 19: Excerpt of the Palette Ontology, Meta-Model Ontology and Domain Ontology related to the modelling languages DKMM, BPMN and ICF

to be created, which requires some ontology engineering effort.

8 Conclusion

This research work started by stressing the current struggle to include domain experts in the design and maintenance of enterprise knowledge graphs (EKGs) schemas. For the resolution of this problem, it was investigated the adoption of a meta-modelling technique and semantic lifting, which are common in the Enterprise Modelling discipline. Findings from literature, expert interviews and case analysis pointed (1) to the need to inject agility into the current meta-modelling technique and (2) to a shift of the paradigm from semantic lifting to an ontology-based meta-modelling. The latter replaces a meta-model with an ontology, thus specifying an abstract syntax of a modelling language in an ontology language. The proposed approach built on this paradigm by proposing operators for the domain-specific adaptations of modelling languages, which are able to ensure the specification of a domain-specific modelling language in an ontology language, even while adapting the language. In this sense, the continuous alignment between human- and machine-interpretable knowledge is achieved. Since concepts in a DSML can address a specific enterprise domain, they can be equivalent to an EKG schema. Therefore, the domain-specific adaptations allow the design and maintenance of EKGs schemata by adding and removing concepts, properties of and relationships among concepts. Because modelling languages have an established syntax, semantics and graphical notation, they can be easily understood by domain experts. The domain-specific adaptations and their immediate test in the single integrated meta-modelling and modelling component enable the tight collaboration between language engineers and domain experts. A domain expert with language engineering know-how can also attempt

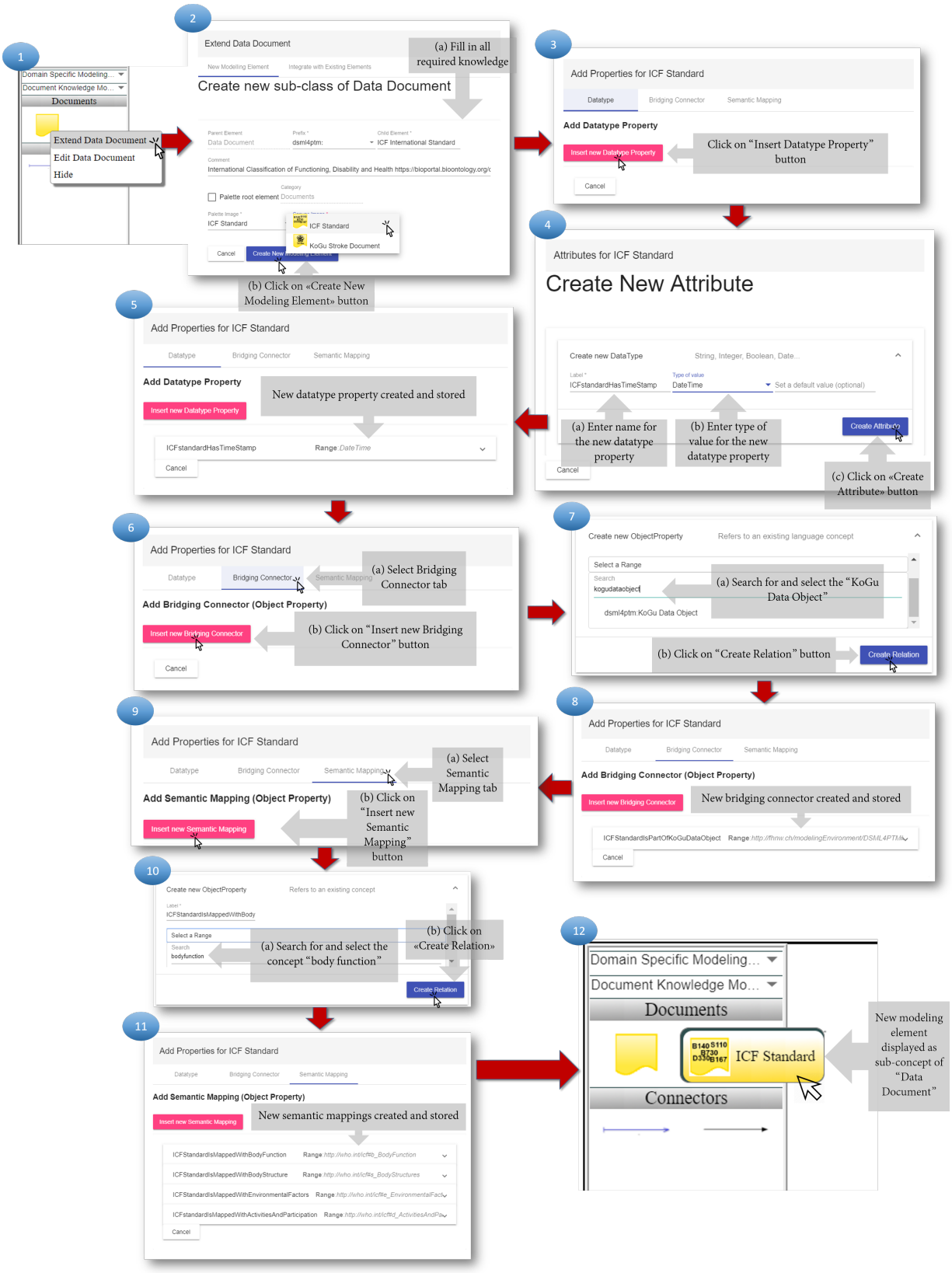

Figure 20: Steps to extend the modelling element "Data Document" with "ICF Standard"

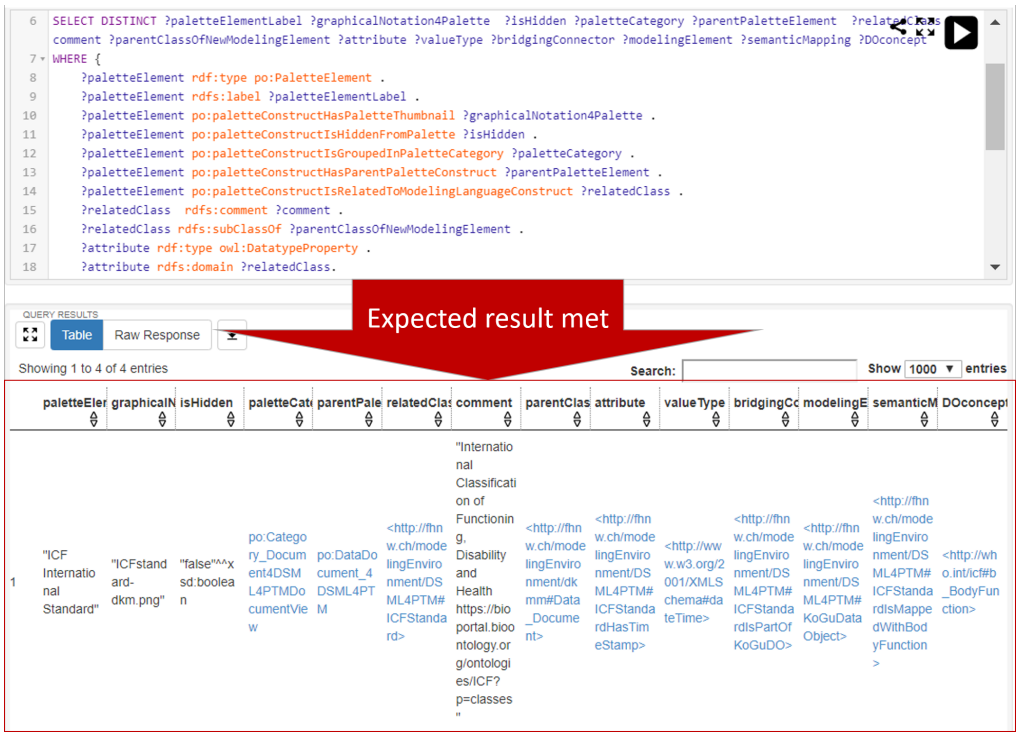

Figure 21: Query results after domain-specific adaptations in Use Case 3: Adding and Referring ICF Standard document

the engineering activity. The conceived agile and ontology-based meta-modelling approach was instantiated in the prototypical tool AOAME, which integrates meta-modelling with modelling activities in a single component. Modelling activities were not described as out of scope in this paper. The implementation of a real-world scenario in AOAME proved the operationality of the approach. Namely, on the one hand, it was shown that from a domain-specific adaptation, the visual approach supports the activity and the result is shown in the palette of AOAME. On the other hand, it was shown that the resulting action manifests also in the knowledge graph schema, which is specified in the ontology language RDF(S). The approach was also proven to be generalizable, as it has been used in other application domains, such as innovation processes and in the agile software development methodology Scrum. The correct design of the artifact was proven by mapping all the requirements with the implemented functionalities in AOAME.

Current research focuses on how to specify visual constraints in the agile and ontology-based meta-modelling approach so as to specify them in the W3C standard SHACL W3C (2017). Such constraints will then be used to validate RDF graph-based models that will also be created in AOAME by instantiating the modelling constructs. For example, one application regards the automatic validation of Enterprise Architecture (EA) principles on EA models (Montecchiari and Hinkelmann 2022).

References

Abu-Salih B. (2021) Domain-specific knowledge graphs: A survey. In: Journal of Network and Computer Applications 185, p. 103076

Allemang D., Hendler J. (2011) Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. Elsevier

Atkinson C., Kuhne T. (2003) Model-driven Development: A Metamodeling Foundation. In: IEEE Software 20(5), pp. 36–41

Azzini A., Braghin C., Damiani E., Zavatarelli F. (2013) Using Semantic Lifting for Improving Process Mining: a Data Loss Prevention System Case Study. In: Proceedings of the 3rd International Symposium on Data-driven Process Discovery and Analysis. CEUR-WS.org, pp. 62–73

Bachhofner S., Kiesling E., Revoredo K., Waibel P., Polleres A. (2022) Automated Process Knowledge Graph Construction from BPMN Models In: Database and Expert Systems Applications. DEXA 2022. Vol. 13426 Lecture Notes in Computer Science Springer, pp. 32–47

Barišić A., Amaral V., Goulão M. (2018) Usability driven DSL development with USE-ME. In: Computer Languages, Systems Structures 51, pp. 118–157

Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fower M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R., Schwaber K., Sutherlan J., Thomas D. (2001) Manifesto for Agile Software Development

Benevides A. B., Guizzardi G. (2009) A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML. In: Enterprise Information Systems. Springer, pp. 528–538

Benyon D., Bental D., Green T. (1999) Conceptual Modeling for User Interface Development. Springer, p. 187

Bertolazzi P., Krusich C., Missikoff M., Manzoni V. (2001) An Approach to the Definition of a Core Enterprise Ontology : CEO. In: International Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations - OES-SEO 2001, pp. 104–115

Bräuer M., Lochmann H. (2007) Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. In: 4th International Workshop on Software Language Engineering (ATEM 2007) at the 10th IEEE/ACM International Conference on Model-Driven Engineering Languages and Systems (MODELS 2007). Springer

Bridgeland D. M., Zahavi R. (2009) Business Modeling: A Practical Guide to Realizing Business Value. Morgan Kaufmann/Elsevier, p. 387

Buchmann R. A., Karagiannis D. (2016) Enriching Linked Data with Semantics from Domain-Specific Diagrammatic Models en. In: Business & Information Systems Engineering 58(5), pp. 341–353

Buchmann R. A. (2022) The Purpose-Specificity Framework for Domain-Specific Conceptual Modeling In: 2nd Springer, pp. 67–92

Burlton R. T., Ross R. G., Zachman J. A. (2017) The Business Agility Manifesto Building for Change. In:

Chaudhri V. K., Baru C., Chittar N., Dong X. L., Genesereth M., Hendler J., Kalyanpur A., Lenat D. B., Sequeda J., Vrandečić D., Wang K., Diego S. (2022) Knowledge graphs: Introduction, history, and perspectives. In: Vol. 43. John Wiley Sons, Ltd, pp. 17–29

Chavula C., Maria Keet C. (2015) An Orchestration Framework for Linguistic Task Ontologies. In: Metadata and Semantics Research. MTSR 2015. Communications in Computer and Information Science. 544th ed. Springer, pp. 3–14

Chiprianov V., Kermarrec Y., Rouvrais S., Simonin J. (2014) Extending Enterprise Architecture Modeling Languages for Domain Specificity and Collaboration: Application to Telecommunication Service Design. In: Software Systems Modeling 13(3), pp. 963–974

Cho H., Gray J., Syriani E. (2012) Creating Visual Domain-Specific Modeling Languages from End-User Demonstration. In: 4th International Workshop on Modeling in Software Engineering (MISE). IEEE, pp. 22–28

Dietz J. L. G. (2006) Enterprise Ontology: Theory and Methodology. Springer, p. 243

Efendioglu N., Woitsch R., Utz W., Falcioni D. (2017) ADOxx Modelling Method Conceptualization Environment. In: Astesj

Ehrlinger L., Wöss W. (2016) Towards a Definition of Knowledge Graphs. In: CEUR-WS.org

Emmenegger S., Hinkelmann K., Laurenzi E., Thönssen B., Witschel H., Zhang C. (2016) Workplace Learning-Providing Recommendations of Experts and Learning Resources in a Context-Sensitive and Personalized Manner: An Approach for Ontology Supported Workplace Learning. In: MODELSWARD 2016 - Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. IEEE

Emmenegger S., Hinkelmann K., Laurenzi E., Thönssen B. (2013) Towards a Procedure for Assessing Supply Chain Risks Using Semantic Technologies. In: Knowledge Discovery, Knowledge Engineering and Knowledge Management. Springer, pp. 393–409

Fill H.-G. (2011) Using Semantically Annotated Models for Supporting Business Process Benchmarking. In: Perspectives in Business Informatics Research. Springer Berlin Heidelberg, pp. 29–43

Fill H.-G., Karagiannis D. (2013) On the Conceptualisation of Modelling Methods using the ADOxx Meta Modelling Platform. In: Enterprise Modelling and Information Systems Architectures (EMISAJ) 8(1), pp. 4–25

Fox M. S., Barbuceanu M., Grüninger M. (1996) An Organisation Ontology for Enterprise Modeling: Preliminary Concepts for Linking Structure and Behaviour. In: Computers in Industry 29(1-2), pp. 123–134

Fox M. S., Gruninger M. (1998) Enterprise Modeling. In: AI Magazine 19(3), p. 109

Frank U. (2010) Outline of a Method for Designing Domain-Specific Modelling Languages. 42

Frank U. (2013) Domain-Specific Modeling Languages: Requirements analysis and design guidelines. In: Domain Engineering. Springer, pp. 133–157

Frank U. (2014) Enterprise Modelling: The Next Steps. In: Enterprise Modelling and Information Systems Architectures (EMISAJ) 9 (1), pp. 22–37

Gailly F., Alkhaldi N., Casteleyn S., Verbeke W. (2017) Recommendation-Based Conceptual Modeling and Ontology Evolution Framework (CMOE+). In: Business & Information Systems Engineering 59(4), pp. 235–250

Giachetti R. E. (2010) Design of Enterprise Systems: Theory, Architecture, and Methods. CRC Press, p. 429

Gomez-Perez J. M., Pan J. Z., Vetere G., Wu H. (2017) Enterprise knowledge graph: An introduction. In: Exploiting Linked Data and Knowledge Graphs in Large Organisations, pp. 1–14

Green P., Rosemann M. (2000) Integrated Process Modeling: An Ontological Evaluation. In: Information Systems 25(2), pp. 73–87

Green P., Rosemann M., Indulska M., Manning C. (2007) Candidate Interoperability Standards: An ontological Overlap Analysis. In: Data Knowledge Engineering 62(2), pp. 274–291

Guizzardi G. (2005) Ontological Foundations for Structural Conceptual Models. PhD, CTIT, Centre for Telematics and Information Technology, p. 441

Haase P. (2019) Hybrid Enterprise Knowledge Graphs. In: CEUR-WS

Harel D., Rumpe B. (2004) Meaningful modeling: what's the semantics of "semantics"? In: Computer 37(10), pp. 64–72

Harel D., Rumpe B. (2000) Modeling languages: Syntax, Semantics and All that Stuff.. Citeseer

Harzallah M., Berio G., Opdahl A. L. (2012) New Perspectives in Ontological Analysis: Guidelines and Rules for Incorporating Modelling Languages into UEML. In: Information Systems 37(5), pp. 484–507

Hevner A. R., March S. T., Park J., Ram S. (2004) Design Science in Information Systems Research. In: MIS Quarterly 28(1), pp. 75–105

Hinkelmann K., Gerber A., Karagiannis D., Thoenssen B., van der Merwe A., Woitsch R. (2016a) A New Paradigm for the Continuous Alignment of Business and IT: Combining Enterprise Architecture Modelling and Enterprise ontology en. In: Computers in Industry Special Issue on Future Perspectives On Next Generation Enterprise Information Systems 79, pp. 77–86

Hinkelmann K., Laurenzi E., Lammel B., Kurjakovic S., Woitsch R. (2016b) A Semantically-Enhanced Modelling Environment for Business Process as a Service. In: ES. IEEE, pp. 143–152

Hinkelmann K., Laurenzi E., Martin A., Montecchiari D., Spahic M., Thönssen B. (2020) ArchiMEO: A Standardized Enterprise Ontology based on the ArchiMate Conceptual Model. In: Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development. SciTePress, pp. 417–424

Hinkelmann K., Laurenzi E., Martin A., Thönssen B. (2018) Ontology-Based Metamodeling, en. In: Business Information Systems and Technology 4.0: New Trends in the Age of Digital Change. Studies in Systems, Decision and Control. Springer, pp. 177–194

Hrgovcic V., Karagiannis D., Woitsch R. (2013) Conceptual Modeling of the Organisational Aspects for Distributed Applications: The Semantic Lifting Approach. In: 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops, pp. 145–150

Huang L., Duan Y., Sun X., Lin Z., Zhu C. (2016) Enhancing UML Class Diagram Abstraction with Knowledge Graph In: Vol. 9937 LNCS Springer, pp. 606–616

Izquierdo J. L. C., Cabot J., López-Fernández J. J., Cuadrado J. S., Guerra E., de Lara J. (2013) Engaging End-Users in the Collaborative Development of Domain-Specific Modelling Languages. In: Cooperative Design, Visualization, and Engineering. Springer, pp. 101–110

Kappel G., Kapsammer E., Kargl H., Kramler G., Reiter T., Retschitzegger W., Schwinger W., Wimmer M. (2006) Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In: International Conference on Model Driven Engineering Languages and Systems. Springer, pp. 528–542

Karagiannis D., Woitsch R. (2015) Knowledge Engineering in Business Process Management. In: Handbook on Business Process Management 2. Springer

Karagiannis D. (2018) Conceptual Modelling Methods: The AMME Agile Engineering Approach. In: Informatics in Economy. Springer, pp. 3–19

Karagiannis D., Buchmann R. A. (2018) A Proposal for Deploying Hybrid Knowledge Bases: the ADOxx-to-GraphDB Interoperability Case. In: Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS), pp. 4055–4064

Karagiannis D., Buchmann R. A., Burzynski P., Reimer U., Walch M. (2016) Fundamental Conceptual Modeling Languages in OMiLAB. In: Domain-Specific Conceptual Modeling. Springer, pp. 3–30

Karagiannis D., Kühn H. (2002) Metamodelling Platforms. In: Proceedings of the Third International Conference EC-Web at DEXA 2002. Springer

Karagiannis D., Lee M., Hinkelmann K., Utz W. (2022) Domain-Specific Conceptual Modeling. Springer

Kejriwal M. (2019) Domain-Specific Knowledge Graph Construction. Springer

Kejriwal M., Shao R., Szekely P. (2019) Expert-Guided Entity Extraction Using Expressive Rules. In: Association for Computing Machinery, Inc, pp. 1353–1356

Kleppe A. G. (2008) Software Language Engineering: Creating Domain-Specific Languages using Metamodels. Addison-Wesley Professional, p. 207

Laurenzi E., Hinkelmann K., Reimer U., Van Der Merwe A., Sibold P., Endl R. (2017) DSML4PTM: A Domain-Specific Modelling Language for Patient Transferal Management. In: ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems Vol. 3. SciTePress, pp. 520–531

Laurenzi E. (Feb. 19, 2020) An Agile and Ontology-Aided Approach for Domain-Specific Adaptations of Modelling Languages. PhD Thesis, University of Pretoria

Laurenzi E., Hinkelmann K., Goel M., Montecchiari D. (2020) Agile Visualization in Design Thinking. In: New Trends in Business Information Systems and Technology. Springer

Leppänen M. (2007) A Context-Based Enterprise Ontology. In: Business Information Systems. Springer, pp. 273–286

Li Y., Zakhozhyi V., Zhu D., Salazar L. J. (2020) Domain Specific Knowledge Graphs as a Service to the Public Powering Social-Impact Funding in the US. In: Vol. 20. ACM

Mancuso M., Laurenzi E. (2023) An Approach for Knowledge Graphs-Based User Stories in Agile Methodologies. In: Perspectives in Business Informatics Research. BIR 2023. Lecture Notes in Business Information Processing. Springer, pp. 133–141

Meissner R., Thor A. (2021) Creation and Utilisation of Domain Specific Knowledge Graphs for E-Learning. In: Gesellschaft für Informatik e.V., pp. 271–276

Mernik M., Heering J., Sloane A. M. (2005) When and How to Develop Domain-Specific Languages. In: ACM Computing Surveys 37(4), pp. 316–344

Montecchiari D., Hinkelmann K. (2022) Towards Ontology-Based Validation of EA Principles. In: The Practice of Enterprise Modeling. PoEM 2022.. Lecture Notes in Business Information Processing Vol. 456. Springer, pp. 66–81

Moody D. (2009) The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. In: IEEE Transactions on Software Engineering 35(6), pp. 756–779

OMG (2011) Business Process Model and Notation (BPMN), Version 2.0

Opdahl A. L., Berio G. (2006) Interoperable language and model management using the UEML approach. In: GaMMa '06: Proceedings of the 2006 international workshop on Global integrated model management. Association for Computing Machinery, Inc, pp. 35–41

Opdahl A. L., Henderson-Sellers B. (2002) Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model. In: Software and Systems Modeling 1(1), pp. 43–67

Panich A., Vatanawood W. (2016) Detection of Design Patterns from Class Diagram and Sequence Diagrams Using Ontology. In: IEEE, pp. 1–6

Parreiras F. S. (2012) Semantic Web and Model-Driven Engineering. Wiley-IEEE Press, p. 264

Peng C., Xia F., Naseriparsa M., Osborne F. (2023) Knowledge Graphs: Opportunities and Challenges. In: Artificial Intelligence Review 56(11), pp. 13071–13102

Peter M., Montecchiari D., Hinkelmann K., Gatziu Grivas S. (2020) Ontology-Based Visualization for Business Model Design. In: The Practice of Enterprise Modeling. Springer, pp. 244–258

Pries-Heje J., Baskerville R., Venable J. (2008) Strategies for Design Science Research Evaluation. In: Proceedings of the 16th European Conference on Information Systems, Galway, Ireland, Paper 87. National University of Ireland

R. C., Wood D., Lanthaler M. (2014) RDF 1.1 Concepts and Abstract Syntax https://www.w3.org/TR/rdf11-concepts/

Rohde F. (1995) An Ontological Evaluation of Jackson's System Development Model. In: Australasian Journal of Information Systems 2(2), pp. 77–87

Sandkuhl K., Fill H.-G., Hoppenbrouwers S., Krogstie J., Matthes F., Opdahl A., Schwabe G., Uludag Ö., Winter R. (2018) From Expert Discipline to Common Practice: A Vision and Research Agenda for Extending the Reach of Enterprise Modeling en. In: Business & Information Systems Engineering 60(1), pp. 69–80

Selic B. (2007) A Systematic Approach to Domain-Specific Language Design Using UML. In: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributes Computing (ISORC'07). IEEE, pp. 2–9

Singhal A. (2012) Introducing the Knowledge Graph: things, not strings

Smajevic M., Bork D. (2021) From Conceptual Models to Knowledge Graphs: A Generic Model Transformation Platform. In: IEEE, pp. 610–614

Stahl T., Völter M. (2006) Model-Driven Software Development: Technology, Engineering, Management. John Wiley

Strahringer S. (1996) Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysenmethoden. Publications of Darmstadt Technical University, Institute for Business Studies (BWL). Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL)

Strembeck M., Zdun U. (2009) An Approach for the Systematic Development of Domain-Specific Languages. In: Software - Practice and Experience 39, pp. 1253–1292

Uschold M., King M., Morale S., Zorgios Y. (1998) The Enterprise Ontology. In: The Knowledge Engineering Review 13(01), pp. 31–89

Vaishnavi V. K., Kuechler Jr. W. (2007) Design Science Research Methods and Patterns: Innovating Information and Communication Technology, 1st. Auerbach Publications

Van Harmelen F., Teije A. t. (2019) A Boxology of Design Patterns for Hybrid Learning and Reasoning Systems. In: Journal of Web Engineering 18(1-3), pp. 97–124

Vernadat F. B. (2003) Enterprise Modelling and Integration. In: Springer, pp. 25–33

Voigt K. (2011) Structural Graph-based Metamodel Matching. Dissertation, Technical University of Dresden, p. 199

W3C (2014) RDF Schema 1.1.. https://www.w3.org/TR/rdf-schema/

W3C (2017) Shapes Constraint Language (SHACL)

Wand Y., Storey V. C., Weber R. (1999) An Ontological Analysis of the Relationship Construct in Conceptual Modeling. In: ACM Trans. Database Systems 24(4), pp. 494–528

Wegeler T., Gutzeit F., Destailleur A., Dock B. (2013) Evaluating the Benefits of Using Domain-Specific Modeling Languages. In: Proceedings of the 2013 ACM workshop on Domain-specific modeling - DSM '13. ACM Press, pp. 7–12

Woitsch R., Hinkelmann K., Maria A., Ferrer J., Yuste J. I. (2016) Business Process as a Service (BPaaS): The BPaaS Design Environment. In: Proceedings of the CAiSE 2016 Industry Track co-located with 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016). CEUR-WS.org

Xiaohan Z. (2020) A Survey on Application of Knowledge Graph. In: Journal of Physics: Conference Series 1487 (1), p. 012016

Yin R. K. (2018) Case Study Research Design and Applications, 6th. Sage Publications Ltd., p. 319

Zachman J. A. (1987) A Framework for Information Systems Architecture. In: IBM systems journal 26(3), pp. 276–292

Zečević I., Bjeljac P., Perišić B., Maruna V., Venus D. (2017) Domain-Specific Modeling Environment for Developing Domain Specific Modeling Languages as Lightweight General Purpose Modeling Language Extensions. In: Springer, pp. 872–881

Zhang H., Kishore R., Ramesh R. (2007) Semantics of the MibML Conceptual Modeling Grammar: An Ontological Analysis Using the Bunge-Wang-Weber Framework. In: Journal of Database Management 18(1), pp. 1–19