An Agile and Ontology-Aided Modeling Environment

Emanuele Laurenzi[1,2,3], Knut Hinkelmann[1,3], and Alta van der Merwe[3]

[1] FHNW University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, 4600 Olten, Switzerland
emanuele.laurenzi@fhnw.ch, knut.hinkelmann@fhnw.ch,
[2] University of Applied Sciences St. Gallen, IPM-FHSG, Institute of Information and Process Management, Rosenbergstrasse, 59, 9001 St. Gallen, Switzerland
[3] University of Pretoria, Department of Informatics, Lynnwood Rd, 0083 Pretoria, South Africa
alta.vdm@up.ac.za

Abstract. Enterprise knowledge is currently subject to ever-changing, complex and domain-specific modeling requirements. Assimilating these requirements in modeling languages brings the benefits associated to both domain-specific modeling languages (DSMLs) and a baseline of well-established concepts. However, there are two problems that hamper the speed and efficiency of this activity: (1) the separation between the two key expertise: language engineering and domain knowledge, and (2) the sequential modeling language engineering life-cycles. In this work, we tackle these two challenges by introducing an Agile and Ontology-Aided approach implemented in our Modeling Environment - the AOAME. The approach seamlessly integrates meta-modeling and modeling in the same modeling environment, thus cooperation between language engineers and domain experts is fostered. Sequential engineering phases are avoided as the adaptation of the language is done on-the-fly. To this end, a modeling language is grounded with an ontology language providing a clear, unambiguous and machine-interpretable semantics. Mechanisms implemented in the AOAME ensure the propagation of changes from the modeling environment to the graph-based database containing the ontology.

Keywords: agile and ontology-aided modeling environment, domain-specific adaptation, domain-specific modeling language

1 Introduction

Today's enterprises are subject to a continuous digital business evolution. Many different complex aspects of enterprises are affected such as processes, organization and product structures, IT-systems as well as different degree of domain specificity, i.e. an entire industry or an application area or a single case in an enterprise. Complex and domain-specific aspects take place within an increasingly challenging environment for enterprises characterized by high competition, cross-organization cooperation, and continuous and unexpected change [1,2]. As a consequence, modern enterprises should have the ability to continuously, quickly and

efficiently capture relevant ever-changing, complex and domain-specific aspects and represent them in enterprise knowledge. Burlton at al. [3] call this ability "Business Agility", which creates competitive advantage and enables to thrive in innovative environments.

Enterprise models' main purpose is to capture enterprise knowledge. Due to the frequent change of the latter, models have to be re-designed continuously as they become outdated. Ideally, modeling approaches and tools should enable modelers to continuously adapt enterprise models by accommodating new modeling requirements quickly and efficiently. Assimilating modeling requirements directly in modeling languages has many benefits that are typically associated to domain-specific modeling languages (DSMLs) built by adapting existing modeling languages. Conversely to DSMLs built from scratch, a set of concepts and well-established semantics already foster the dissemination of a DSML within the modeling community. Also, the latter provide a baseline of lessons learned in the field and a set of well-known concepts that can be borrowed. This "domain-specific adaptation" of one or more modeling languages has the ultimate goal of facilitating modeler's task in creating meaningful models as well as increasing understanding of models by domain experts. This goal should be achieved by the language engineer (developer and adapter of the modeling language), who is required to continuously, quickly and efficiently adapt modeling languages. This activity, in contrast, is a time-consuming engineering effort. This is mainly due to the lack of agility in the domain-specific adaptation life-cycles and in the way these are implemented in modeling tools.

This paper elaborates an agile and ontology-aided approach for a domain-specific adaptation of modeling languages. The approach is implemented in the modeling environment called AOAME (Agile and Ontology-Aided Modeling Environment). Section 2 presents the theoretical background upon which AOAME is built. Next, section 3 emphasizes the two main challenges addressed in this work and introduces the related works that strive to address them. Section 4 describes our AOAME solution, including the architecture, the ontologies and the mechanisms that support the solution. The paper ends with section 5, where a validation of the AOAME is presented with respect to a use case derived from a research project.

2 Background

This section introduces the theoretical foundations of this work. In the following sub-sections we first define a modeling language. Next, notions on the modeling language developing technique "meta-modeling" are provided. Finally, the term domain-specific adaptation is introduced.

2.1 Modeling Language Specification

A modeling language is specified by notation, abstract syntax and semantics [4]. Abstract syntax refers to the class hierarchy of modeling elements together with their relations and attributes, through which the language terminology is defined. Modeling constructs are typically expressed through graphical or textual

notation (also known as concrete syntax), which should be cognitively adequate to ensure users' understanding of models [5]. The semantics define the meaning of the syntactic elements of a modeling language. Harel and Rumpe [6] claim that the semantics of a modeling language is described in two parts: the semantic domain and the semantic mapping. "The semantic domain can be defined independently of the syntax: in fact, we can use completely different languages for describing the same kinds of systems, so that these languages might all use the same semantic domain" [6]. Whilst concrete syntax (e.g. graphical notation) is used to create models, abstract syntax specifies what kind of knowledge a model is allowed to contain. Hence, the semantic mapping takes place from concepts in the abstract syntax to the domain semantics.

In some cases, however, not all semantics can be expressed through this mapping [7]. In order to govern how the language constructs can be combined to produce valid models, constraints (or rules or restrictions) should be inserted over concepts. Thus, the semantics of a modeling language can be defined by (a) abstract syntax, including constraints or restrictions on concepts, (b) domain semantics and (c) the mapping between concepts from the abstract syntax to those of the domain. The mapping can be seen as a relation between the linguistic view and the domain of discourse view. The differentiation between the two views is consistent with the work in [8], which regards fundamental to have both the linguistic definition and the domain definition in a modeling language.

Semantics can be expressed formally (i.e. through mathematics or ontologies) or informally (i.e. through natural language). Formality of the semantics depends on the formalism of the abstract syntax, the domain semantics and the semantic mapping. The latter should be made explicit (according to Harel and Rumpe [6]) as it is not satisfactory to define the semantic mapping by examples, as it does not allow analysis through which insights can be gained. Therefore the semantic mapping also must be formally defined. Section 4.2 describes how the semantic language, including the semantic mapping, is made explicit and formal.

2.2 Meta-modeling for Enterprise Modeling Languages

Enterprise Modeling Languages (EML) such as ArchiMate[4] and BPMN[5] are typically specified in UML class diagrams. Concepts of an EML reside at Level 2 of the meta-modeling hierarchy introduced by Strahringer [9]. Abstract syntax and constraints of an EML are specified in the meta-model and elements of the abstract syntax are furnished with graphical notations such that modelers can then create models in the lower level, i.e. Level 1. That means graphical notations are not just shaped boxes that rely on the human-interpretation. Instead, each notation is instantiated from a higher abstraction concept with explicitly defined semantics, which is based on a concept taxonomy and descriptive properties. This leads to the definition of the language terminology, thus a vocabulary for modeling constructs in the considered domain is made available. As a consequence, the semantics of a modeling language emerges as a tangible artifact that may be exchanged, inspected, and discussed. Hence, an understanding of the problem domain increases together with a better comprehensibility of the modeling language. As an example, concepts like *Sequence Flow* or *Task* in BPMN are part of a taxonomy that reflects the semantics of the language. These concepts are associated with correspondent graphical notations in order for the modeler to use them.

[4] http://pubs.opengroup.org/architecture/archimate3-doc/
[5] https://www.omg.org/spec/BPMN/2.0/About-BPMN/

Karagiannis at al. [10] introduce the notion of *domain-specificity degree*, where an higher specificity degree means assimilating concepts in the meta-model that target a more specific domain. Meta-models of EMLs capture aspects that target a particular degree of a domain. BPMN targets process modeling, whereas Archimate targets enterprise architecture modeling. Their particular degree of a domain can serve as a baseline to build modeling languages with higher domain-specificity degree [10,11,12], which in literature are also known as domain-specific modeling languages (DSMLs) [7]. The baseline provides many advantages. For instance, established experience, lessons learned and best practices can be taken into account. Also, EMLs provide concepts with well-known notations and a widely accepted semantic, which foster the dissemination of the DSML within the modeling community [13].

2.3 Domain-Specific Adaptation

The activity of adapting a modeling language to add more domain-specificity degree is commonly known as modeling language adaptation or extensibility [14,15,16]. Jablonski at al. [14] define the latter as an "extension or extensibility so that domain specific requirements can be integrated or domain specific semantics are better reflected". However, the term extensibility is limited only to add new concepts or restricting value types or values, e.g. profiling mechanisms from UML like stereotype and tagged values [17]. Additionally, the term modeling language adaptation is commonly limited to one modeling language, excluding the integration of different modeling languages [13].

Another emerging term for this activity is "domain-specific adaptation" [13,5]. As shown in [11], domain-specific adaptation also includes integration of different modeling languages and simplification of these by removing unnecessary concepts. In this work, we define a "domain-specific adaptation" as the adaptation of one or more modeling languages. It can comprise the following actions: (1) removal of unnecessary concepts, (2) specialization of concepts, (3) integration of concepts from different modeling languages, (4) restrictions on attribute types and values.

3 Problems and Related Work

Most of current meta-modeling approaches and the way these are implemented in modeling workbenches (e.g. EMF, ADOxx, TextEdit, Eugenia, MetaEdit, Kaos, ATL) address different expertise. Namely, the conceptualization and implementation of the meta-model (Level 2) target language engineers, whereas the domain expert (Level 1) would use the concrete syntax to create models. Noteworthy is the fact that the most significant feedback and amendments for the language originate at the early stage as soon as the first version of the language is being used. Pitfalls related to inappropriate constraints, abstraction issues, or ambiguity of modeling constructs are likely to arise [7]. Also, decisions on whether to promote productivity at the expenses of re-usability (or vice versa) of the modeling language are subject to continuous changes. Unless domain experts have language engineering skills, new requirements cannot be accommodated by modelers or domain experts. Instead they have to be properly communicated to the language engineer, who adapts the modeling language at the meta-level. In turn, changes should be propagated in the modeling tool, which implements the new language specifications. If feedback or amendments

are not properly communicated (e.g. due to a lack of cooperation), misinterpretations can arise, which hamper the adaptation process and the quality of the released DSML [18].

Ideally, conditions to foster the cooperation between language engineers and domain experts should be created. This sets the *first challenge* of this work. It goes in line with the recent research agenda of Enterprise Modeling proposed in [19], where with the slogan "modeling for the masses" emphasizes the need to welcome non-experts in the field of modeling for an inter-disciplinary benefit. Research work addressing this challenge, however, is still in its infancy. Izquierdo at al. [18] were first to propose a collaborative approach to create DSMLs. Namely, an example-driven and collaborative supported tool was developed to engage end-users in the construction of DSMLs. With a similar end, Barisic at al. [20] propose the USE-ME as a methodological approach that covers the language development process, along which domain experts are involved in the assessment through user interface experimental techniques. While on one hand these solutions improve the quality of the final DSML, on the other hand they do not solve the problem of the time-consuming engineering effort. This is mainly due to the sequential engineering approach that characterizes the life-cycle of domain-specific adaptations.

Avoiding this sequential life-cycles sets the *second challenge* addressed in this work. Typically, such a life-cycle follows the iterative phases of (1) first eliciting relevant domain knowledge. Then, (2) the language engineer conceptualizes the meta-model. Subsequently, (3) the meta-model is implemented in a meta-modeling tool, allowing the modeler or domain expert to use the intermediate modeling language, and thus (4) evaluating it. The latter generates feedback and determines language amendments. Hence, the process iterates until a stable enough version of the language is achieved. Some examples of such life-cycle can be found in [20,21,22]. To foster agility, the AMME framework is proposed in [23] and instantiated by the OMiLab Lifecycle. The latter foresees feedback channels along different engineering phases to promote agility in the evolution of modeling requirements.

In domain-specific adaptation, however, each time a new modeling requirement is to be embedded in the modeling language, it has to go through the all above-mentioned engineering phases, sequentially [24]. This does not just result in a time-consuming engineering effort as in the case of a lack of cooperation. A sequential approach also becomes problematic with the long duration of each phase as the longer they take the higher is the risk of having outdated requirements. This would lead to a mismatch between the created DSML and the actual needs of end-users.

4 The Agile and Ontology-Aided Modeling Environment

The Agile and Ontology-Aided Modeling Environment (AOAME) was conceived through the design science research (DSR) approach [25], which provided guidance throughout the construction of the artifact. In particular, the awareness of problem was initially raised by use cases and lessons learned from three research projects targeting three different domains: (1) a patient discharging process among sites of care within the health-care sector [11,26]; (2) business process requirements and cloud service specification for the Business-IT matchmaking in the Cloud [27,28]; (3) workplace learning in public administrations [29]. Each project presented a model-driven solution and domain-specific adaptation activities were performed adopting current engineering life-cycles. As a result, the

main challenges introduced in the previous section raised together with the first set of requirements for the conceptualization of AOAME (listed in [24]).

Differently from the work described in [24], this paper elaborates on the AOAME architecture. To this end, we first describe the main idea for addressing the two above-mentioned problems (section 4.1). Next, section 4.2 introduces the AOAME architecture that builds upon an ontology-based approach. Thus, the ontology architecture is also presented together with the motivation of the adopted ontology language. This section ends with mechanisms that allows the automatic propagation of domain-specific adaptations (section 4.3).

4.1 Seamless Integration of Meta-modeling and Modeling

To address the two main challenges introduced in section 3, we found inspiration in UML mechanisms such as *stereotype* and *tagged values*, which allow customizing modeling constructs on-the-fly. These mechanisms are typically implemented in modeling tools and both the customization and modeling take place in the same modeling environment (e.g. Visual Paradigm[6]).

Similarly, we conceptualized a unique modeling environment that seamlessly integrates meta-modeling and modeling for an on-the-fly domain-specific adaptation. That means, the sequential engineering phases are avoided as adaptations can (a) occur on-the-spot as they are needed and (b) be tested right away in the same modeling environment. Thus, both expertise language engineering and domain knowledge can be employed at the same time. This enables a tight and synchronized cooperation between the language engineer and the domain expert. Obviously, in case where someone has expertise in both fields would anyway benefit from the agile approach.

However, on-the-fly adaptations may lead to new modeling constructs for which semantics need to be made explicit. If not, the meaning of the new modeling constructs may be ambiguous or not understood by the users. To overcome this issue our solution makes use of ontologies. Namely, abstract syntax and additional semantics of a modeling language are made explicit by grounding them with an ontology formalism.

Making use of ontologies for a formal representation of models or modeling language constructs is an established practice within the research community, e.g. [30,29,31]. An ontology has not just the benefit of providing clear and unambiguous understanding of the meaning of language constructs and model concepts. Also, ontologies are interpretable by and interchangeable among machines and enable automated reasoning. Additionally, compared to approaches adopting standard data-bases, the ontology-based ones are more powerful in terms of query results. However, current ontology-based approaches mainly refer to semantic annotation, where ontology concepts (i.e. machine-interpretable concepts) are annotated to concepts of the meta-model or models (i.e. human-interpretable models). This approach has the drawback of the manual or semi-automatic alignment between meta-model concepts and ontology concepts, which can be error-prone and time-consuming. On one hand, mechanisms for the automatic generation of ontologies from models overcome this drawback. Examples for such mechanisms range from the creation of knowledge graphs [30] to more expressive ontologies (e.g. OWL - Ontology Web Language[7]) like in [29]. On the other hand, this solution may cause inconsistency issues that originate from the separation between human-interpretable models and machine-interpretable concepts [32]. For

[6] https://www.visual-paradigm.com/

[7] https://www.w3.org/OWL/

instance, if a change occurs in the ontology, the human-interpretable model has to be adapted, accordingly. Also, if changes occur in the meta-model, transformation patterns for the ontology generation might need to be adapted. This is the case in [29], where XSLT[8] templates are created for the automatic generation of ontology instances from XML models.

To avoid these problems we build upon the ontology-based meta-modeling approach introduced in [5]. This approach foresees an ontology as a meta-model, where ontology concepts are furnished with human-interpretable modeling constructs, i.e. graphical notations. This approach is implemented in our solution so that changes that occur in ontology concepts (e.g. adding a data type property to a class) are automatically reflected in the human-interpretable modeling constructs. Additionally, we took a step further by implementing the automatic propagation of changes that occur in the human-interpretable modeling constructs back to the ontology. This enables language engineers who are non-ontology experts to adapt a modeling language while the reflecting ontology is adapted automatically. Hence, a symbiosis is achieved between human-interpretale modeling constructs and related machine-interpretable concepts. To this end, a new architecture was introduced and is described in the next section.

4.2 AOAME Architecture

The AOAME architecture (see Fig. 1) foresees the support not just of the propagation from ontology concepts to modeling constructs, which ensures that all the displayed graphical notations in the modeling environment are grounded by ontology concepts (see arrow 1 and 2 in Fig. 1) but also, as mentioned above, it supports the propagation of changes (on modeling constructs) from the modeling environment back to the ontology (see arrows 3 and 4 in Fig. 1).

The architecture consists of three main components: a web-based modeling environment (*ME*) developed in AngularJS[9] (see left-hand side of Fig. 1). This has two sub-components: (a) the "Palette" where graphical notations of modeling constructs are displayed and (b) the "Canvas" where models are designed. In this work we focus on the "Palette" sub-component as it is the one enabling a domain-specific adaptation. The second main component is the Java-based web-service (*WS*) - at the center of Fig. 1 - which implements algorithms and mechanisms for the automatic propagation of changes from and to the Palette sub-component. The third component is a graph-based database (a.k.a. triple store - *TS*) implemented in Apache Jena Fuseki[10], which contains the three main ontologies of AOAME: the Palette Ontology (*PO*), the Modeling Language Ontology (*MLO*) and the Domain Ontology (*DO*).

Ontology Architecture. Fig. 2 shows the three main ontologies of AOAME. Namely, the *MLO* reflects the abstract syntax, while the *DO* reflects the semantic domain. Concepts from the *MLO* are mapped with concepts of the *DO* (see section 2.1 for the theoretical foundation that motivates the mapping). Concepts in the *PO* reflect the graphical notations of the language, and are directly linked to the concepts in the abstract syntax.

In more detail, the *PO* contains concepts and relations about graphical notations of the modeling language as well as knowledge for positioning the graphical

[8] https://www.w3.org/TR/xslt-10/

[9] https://angularjs.org/

[10] https://jena.apache.org/documentation/fuseki2/

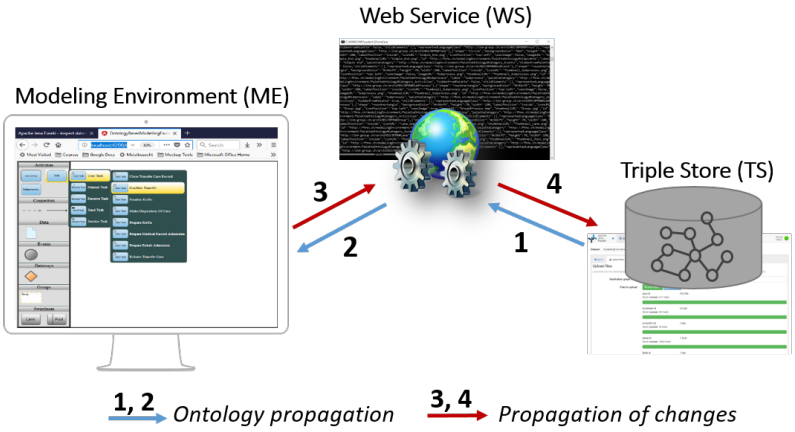

Fig. 1. AOAME Architecture

notations over the palette. Thus, the palette in the *ME* is populated by the *PO* concepts. In particular, the class *po: PaletteConnector* contains instances reflecting connectors of one or more modeling languages (e.g. message flow and sequence flow for BPMN), while the class *po: PaletteElement* contains instances reflecting modeling elements of one or more modeling languages (e.g. task, data object for BPMN). Instances from both classes are meant to contain knowledge regarding the graphical notation, e.g. the name of the image extension, the size, whether it should be visible or hidden from the palette. These are all in the form of data type properties. Also, they contain object properties, where the most relevant are *po: hasParent* and *po: isRelatedTo*. The former determine the hierarchy among modeling constructs that will then be shown in the Palette sub-component. The latter specifies which class of *MLO* the instance relates to. This relation reflects the link that connects notation with (abstract) syntax as described in [4]. In terms of ontology architecture, this implies that the *PO* includes the *MLO* (see Fig. 2).

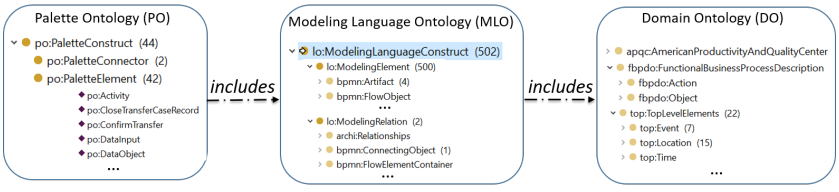

Fig. 2. Ontology Architecture

The *MLO* contains classes and properties describing the abstract syntax elements of a modeling language, i.e. modeling elements and modeling relations with respective taxonomy and object properties. MLO includes one or more modeling languages, which are separate from each other or integrated. Each

MLO concept gets the prefix of the language it belongs to, e.g. Task in BPMN is shown as a class bpmn:Task. The object property *lo:is Mapped With* reflects the formal explication of the semantic mapping introduced in section 2.1. This connects concepts from the *MLO* to those from the *DO*. Hence, the former includes the latter in the architecture shown in Fig. 2. In case there is the need to use a concept of a modeling language, the related ontologies (*MLO* and *PO*) need to be loaded in the *TS*.

The *DO* contains classes and properties that describe the semantic domain. As introduced in section 2.1, the latter is independent from the abstract syntax of a language and describes a domain of discourse. The *DO* also consists of existing ontologies that are loaded in the *TS* to further specify a language construct. An example would be an ontology reflecting the standard American Productivity and Quality Center (APQC). This will be elaborated in the use case introduced in section 5.

Ontology Language. The choice of a ontology language typically depends on the purpose the ontology, i.e. types of facts that are important to deduce, represent and/or retrieve [33]. In our work we adopt the Resource Description Framework Schema (RDFS) 1.1[11]. This lightweight ontology language fits the actions of the domain-specific adaptation as defined in section 2.3, e.g. create a sub-class, attribute, relations etc. Moreover, it allows to have classes as instances of other classes, on the contrary to the more expressive above-mentioned OWL. The latter is limited to the knowledge representation of two levels: the TBox (i.e. classes) and the ABox (i.e. instances). This representation makes OWL unable to support a multi-layer representation that characterizes meta-model representations [34].

Instead, by adopting RDFS in AOAME we are able to further instantiate modeling elements from the *PO* to create models. Also, multilayer representation at design phase is supported, e.g., by modeling execution data as instances of process activities. Semantic rules (e.g. SPIN[12]) and the SPARQL[13] query language can be performed against the ontologies. The former are used to infer new knowledge while the latter provides powerful query constructions. The research works described in [28,29,35] show the validity of this approach.

An initial set of ontologies reflecting modeling languages should be provided by an ontology engineer. Next, actions allowed by the user in the *ME* are such that the expressivity power of the ontology does not increase. Hence, the risk of entering axioms that might be contradicting is avoided, and the ontology quality is not harmed over time. A way to allow expressive statements (e.g. in BPMN, a start event is not allowed to have an incoming sequence flow) while keeping the current ontology expressivness is by adopting the recent W3C recommendation language constraint SHACL[14]. This topic is, however, out of scope in this work but currently under investigation for its user-friendly applicability on the AOAME.

4.3 Mechanisms for an On-the-Fly Domain-Specific Adaptation

To enable the on-the-fly domain-specific adaptation from the modeling environment, we first derived operators from possible action on the ontologies. The

[11] https://www.w3.org/TR/rdf-schema/
[12] http://spinrdf.org/
[13] https://www.w3.org/TR/rdf-sparql-query/
[14] https://www.w3.org/TR/shacl/

list of operators is introduced in [24] and was implemented in the Palette subcomponent. These, mainly consists of creating, updating and deleting of (i) subclass relations, (ii) object properties and (iii) data type properties. Concrete types and values can also be assigned to the latter. To ensure the automatic propagation of changes from *ME* to the *TS*, mechanisms were conceptualized in terms of semantic rules and subsequently implemented for each operator. Semantic rules always impact both the PO and the MLO. Depending on the user actions over the operator semantic rules might also impact the DO. This is following explained through the description of mechanisms that are applied to a concrete operator: the "create sub-class". This leads to two possible results: (1) integration of modeling constructs from different modeling languages, i.e. a modeling construct is extended with a modeling construct of another modeling language; (2) extension of a modeling construct with a new modeling construct. Both generate different semantic rules. The former leads to the following semantic rules that are described in natural language:

1. Create a relation *po:hasParent* in the *PO* between the instance that is being extended and the selected instance.
2. Create a relation *rdfs:subClassOf* in the *MLO*, between the class that relates to the selected instance and the class that relates to the instance that is being extended.

The second possible result generates the following semantic rules:

1. Create a new class and a new relation *rdfs:subClassOf* in the *MLO*, where the new class is sub-class of the modeling construct that is being extended.
2. Create a new instance in the *PO*, containing two new relations *po: isRelatedTo* pointing to the class created in rule 1, *po:hasParent* pointing to the instance of parent class of the class created in rule 1.
3. If concepts from the *DO* are selected, create as many relations *po: isMappedWith* as the number of the selected concepts pointing to the selected concepts. In case concepts from the *DO* are not selected, this semantic rule is not generated.
4. If attributes are inserted, create as many data type properties as the number of the attributes pointing to the specified type, e.g. string, integer, boolean etc. In case attributes are not inserted, this semantic rule is not generated.

Generating the semantic rules and subsequently firing them against the *TS* enable the insertion of the new data from the modeling environment (*ME*) to the ontology. Semantic rules are created in the *WS* component and algorithms are implemented to feed them automatically with data coming from the *ME*.

5 Validation

The validation of the approach is based on a use case extracted from the European research project CloudSocket[15]. For space reasons we only show one implemented scenario of the use case. This is motivated by the need for a cloud broker (i.e. domain expert) of specifying predefined functional requirements of parts of a business process (BP) that reflects a send invoice scenario. The aim is to facilitate the requirement annotation of BP models. This enables the ontology-based business matchmaking between BP requirements and specifications of existing BPs residing in a cloud marketplace. Hence, the most suitable BPs are retrieved

[15] https://site.cloudsocket.eu/

for the given BP requirements by means of a SPARQL query. In this scenario we describe the actions needed to adapt the BPMN language such that a requirement annotation of BPs is made possible. Language adaptations would then be propagated to the ontologies. Also, models built with such language are grounded with an ontology language. Thus, the SPARQL query for the matchmaking can be performed straight without intermediate transformations (e.g. from models to ontologies) or semantic annotation steps. For a comprehensive description on how a SPARQL query and semantic rules are implemented for the matchmaking we suggest to have a look at the research work in [28].

We assume that the cloud broker cooperates with a language engineer to analyse the problem and sketch an appropriate solution. The latter includes a predefined requirement annotation regarding the activities of customer relationship management. For this, the BPMN modeling construct "Group" should be extended with the new "Managing Customer Relationship" construct. The latter should be further specified with semantic domain elements such as predefined values of APQC category, an Action and an Object. For instance, APQC category: *3.5.2.4 Manage Customer Relationship*, Action: *Manage*, and Object: *Customer* would suite the annotation of an activity called *Manage Customer Relationship* of a BP. The language engineer selects the BPMN construct to extend (from the *ME*), which in this case is "Group". This leads to the pop-up window pointed by arrow 1, shown in Fig. 3. The tab shown in the pop-up

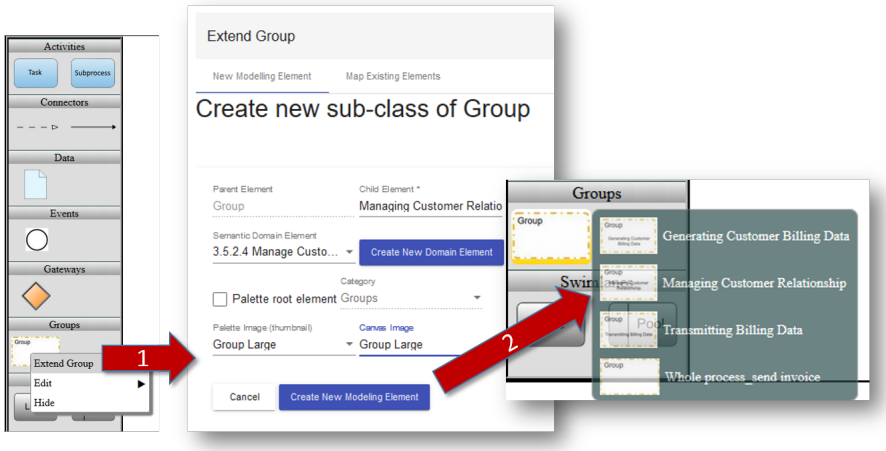

Fig. 3. Extending the BPMN construct "Group" in AOAME

enables the extension of modeling constructs with new ones (the second-right tab enables the integration of existing constructs but it is out of scope in this use case). Hence, information related to the abstract syntax concept of the new modeling construct are to be added. These will be the name of the new concept, the semantic relation pointing to the Domain Ontology concepts and the graphical notations to associate them with. The latter should be uploaded beforehand on a dedicated folder before it can be selected. Additional information like attributes and further relations are omitted for space reasons. In this use

case we assume that concrete values for *3.5.2.4 Manage Customer Relationship* (as APQC category), *Manage* (as an action) and *Customer* (as an object) are already loaded in the *TS* as part of the *DO*. Thus, the language engineer can select them from the drop-down menu entitled Semantic Domain Element (see it in Fig. 3 with the value already selected). In case a semantic element does not exist yet, the language engineer can create it on the spot. This is then inserted in the *TS* as a *DO* concept.

After saving the new modeling construct *bpmn: ManagingCustomerRelationship*, a SPARQL INSERT DATA query is generated, and Fig. 4 shows an excerpt of it. This reflects the implementation of the second set of semantic rules introduced in 4.3. Namely, the new class Managing Customer Relationship is inserted as a sub-class of the BPMN Group construct and labelled with the inserted name. Also, the class is mapped to the three semantic elements (see rows with relations *po:is Mapped With* shown in Fig. 4). A new instance *po:Managing Customer Relationship* is created in the class *po:Palette Element* (see third last row of the figure). An object property *po:hasParent* is added pointing from the created instance *po:Managing Customer Relationship* to the instance that corresponds to the parent class of the created class, i.e. *po:Group*. An object property *po:is Related To* is created pointing from the created instance *po:Managing Customer Relationship* to the created class *bpmn:Managing Customer Relationship* (see last row of Fig. 4).

The SPARQL INSERT DATA in this case would contain additional data types and object properties, e.g., the *po: hasGraphicalNotation* containing the URI of the graphical notation of *po: ManagingCustomerRelationship* as well as the value for the data type property *po: hiddenfromPalette*, which is set to false by default to show the related graphical notation in the palette. This value then can be changed from the palette (see the Hide feature in the small window on the top-left quadrant of Fig. 3). If additional attributes or relations would be added by the user, these would be attached to the SPARQL INSERT DATA in the form of data types and object properties, respectively. Next, algorithms implemented in the *WS* fetch the new changes made in the ontology and propagate them back in the palette sub-component of the *ME*. Thus, the new graphical notation of the modeling construct "Managing Customer Relationship" will be displayed in the palette as a sub-element of "Group" (see far right screen-shot pointed to by the second arrow in Fig. 3). Hence, the new modeling construct is now ready to be used to annotate existing BP models by the domain expert. In case he or she needs to modify the modeling construct, the change can be performed on-the-fly with the help of the language engineer.

6 Conclusion

This paper introduces the Agile and Ontology-Aided Modeling Environment (AOAME) with two main objectives: (1) fostering cooperation between language engineers and domain experts while performing a domain-specific adaptation on modeling languages, and (2) avoiding sequential engineering phases in the modeling language engineering life-cycle. These are pillars of agile approaches. To achieve these two goals an architecture has been conceptualized that is built upon the ontology-based meta-modeling. Hence, while language engineers and domain experts cooperate in adapting one or more modeling languages, mechanisms in the background allow the propagation of changes to the ontologies. Also the propagation of ontologies to the modeling environment are made possible. Namely, algorithms implement the ontology-based meta-modeling approach by showing the graphical notations of ontology concepts in the palette of the

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX bpmn: <http://ikm-group.ch/archiMEO/BPMN#>
PREFIX apqc: <http://ikm-group.ch/archimeo/apqc#>
PREFIX fbpdo: <http://ikm-group.ch/archimeo/fbpdo#>
PREFIX po: <http://fhnw.ch/modelingEnvironment/PaletteOntology#>
INSERT DATA {
bpmn:ManagingCustomerRelationship rdf:type rdfs:Class .
bpmn:ManagingCustomerRelationship rdfs:subClassOf bpmn:Group .
bpmn:ManagingCustomerRelationship rdfs:label "Managing Customer Relationship" .
bpmn:ManagingCustomerRelationship po:isMappedWith apqc:3.5.2.4_Manage_Customer_Relationship .
bpmn:ManagingCustomerRelationship po:isMappedWith fbpdo:Customer .
bpmn:ManagingCustomerRelationship po:isMappedWith fbpdo:Manage .
po:ManagingCustomerRelationsip rdf:type po:PaletteElement .
po:ManagingCustomerRelationsip po:hasParent po:Group .
po:ManagingCustomerRelationsip po:isRelatedTo bpmn:ManagingCustomerRelationsip .
}
```

Fig. 4. Excerpt of the automatic generated SPARQL INSERT DATA

modeling environment. The ontology architecture supporting the AOAME is also introduced, with three main ontologies: the Palette Ontology, Modeling Language Ontology and the Domain Ontology. These reflect the notation, abstract syntax and semantics of a modeling language, respectively. The AOAME also implements the concept of semantic mapping between the abstract syntax and language-independent concepts. The latter can be contextualized within the LinkedData, which increasingly contain world-wide standards and vocabularies. Thus, we regard the AOAME as a concrete step to address the challenge of not just making semantics of enterprise modeling languages explicit but also linking them to the bottom-up web of semantically annotated data, as introduced in the research agenda for Enterprise Modeling [19]. Future work goes in the direction of enhancing the AOAME prototype based on projects addressing different industry applications. Also, a user-friendly way to insert constraints among modeling concepts (e.g. prohibiting the connection between two specific modeling constructs) is being investigated.

References

1. Horkoff, J., Jeusfeld, M.A., Ralyté, J., Karagiannis, D.: Enterprise Modeling for Business Agility. Business & Information Systems Engineering **60**(1) (feb 2018) 1–2
2. Hinkelmann, K., Gerber, A., Karagiannis, D., Thoenssen, B., van der Merwe, A., Woitsch, R.: A new paradigm for the continuous alignment of business and IT: Combining enterprise architecture modelling and enterprise ontology. Computers in Industry **79** (2016) 77–86
3. Burlton, R.T., Ross, R.G., Zachman, J.A.: The Business Agility Manifesto. https://busagilitymanifesto.org (2013)
4. Karagiannis, D., Kühn, H.: Metamodelling Platforms. In: In Proceedings of the 3rd International Conference EC-Web 2002 – Dexa 2002, Aix-en-Provence, France, 2002, LNCS 2455, Springer-Verlag (2002) 182
5. Hinkelmann, K., Laurenzi, E., Martin, A., Thönssen, B.: Ontology-Based Metamodeling. In Dornberger R., ed.: Business Information Systems and Technology 4.0. Studies in Systems, Decision and Control. Springer, Cham (2018) 177–194

6. Harel, D., Rumpe, B.: Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff. Technical report (2000)
7. Frank, U.: Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In: Domain Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2013) 133–157
8. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE Software **20**(5) (2003) 36–41
9. Strahringer, S.: Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysenmethoden. Publications of Darmstadt Technical University, Institute for Business Studies (BWL) (1996)
10. Karagiannis, D., Buchmann, R.A., Burzynski, P., Reimer, U., Walch, M.: Fundamental Conceptual Modeling Languages in OMiLAB. In: Domain-Specific Conceptual Modeling. Springer International Publishing, Cham (2016) 3–30
11. Laurenzi, E., Hinkelmann, K., Reimer, U., van der Merwe, A., Sibold, P., Endl, R.: DSML4PTM - A domain-specific modelling language for patient transferal management. In: ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems, Porto, Portugal. (2017) 520–531
12. Braun, R.: Extensibility of Enterprise Modelling Languages. PhD thesis, Technischen Universit at Dresden (2016)
13. Braun, R.: Towards the State of the Art of Extending Enterprise Modeling Languages. In: 3rd International Conference on Model-Driven Engineering and Software Development, Angers, France, IEEE (2015)
14. Jablonski, S., Volz, B., Dornstauder, S.: A Meta Modeling Framework for Domain Specific Process Management. In: 2008 32nd Annual IEEE International Computer Software and Applications Conference, IEEE (2008) 1011–1016
15. Chiprianov, V., Kermarrec, Y., Rouvrais, S.: Extending Enterprise Architecture Modeling Languages. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12, New York, New York, USA, ACM Press (2012) 1661
16. Atkinson, C., Gerbig, R., Fritzsche, M.: Modeling Language Extension in the Enterprise Systems Domain. In: 2013 17th IEEE International Enterprise Distributed Object Computing Conference, IEEE (sep 2013) 49–58
17. Lidia Fuentes, Antonio Vallecillo: An Introduction to UML Profiles. UPGRADE, The European Journal for the Informatics Professional **5**(2) (2004) 5–13
18. Izquierdo, J.L.C., Cabot, J., López-Fernández, J.J., Cuadrado, J.S., Guerra, E., de Lara, J.: Engaging End-Users in the Collaborative Development of Domain-Specific Modelling Languages. In: Cooperative Design, Visualization, and Engineering. Springer, Berlin, Heidelberg (2013) 101–110
19. Sandkuhl, K., Fill, H.G., Hoppenbrouwers, S., Krogstie, J., Matthes, F., Opdahl, A., Schwabe, G., Uludag, Ö., Winter, R.: From Expert Discipline to Common Practice: A Vision and Research Agenda for Extending the Reach of Enterprise Modeling. Business & Information Systems Engineering **60**(1) (feb 2018) 69–80
20. Barišić, A., Amaral, V., Goulão, M.: Usability driven DSL development with USE-ME. Computer Languages, Systems & Structures **51** (jan 2018) 118–157
21. Cho, H., Gray, J., Syriani, E.: Creating visual Domain-Specific Modeling Languages from end-user demonstration. In: 2012 4th International Workshop on Modeling in Software Engineering (MISE), IEEE (jun 2012) 22–28
22. Ceh, I., Crepinsek, M., Kosar, T., Mernik, M.: Ontology driven development of domain-specific languages. Computer Science and Information Systems **8**(2) (2011) 317–342

23. Karagiannis, D.: Conceptual Modelling Methods: The AMME Agile Engineering Approach. In: Informatics in Economy. Springer, Cham (jun 2018) 3–19
24. Laurenzi, E., Hinkelmann, K., Izzo, S., Reimer, U., van der Merwe, A.: Towards an Agile and Ontology-Aided Modeling Environment for DSML Adaptation. In Raimundas Matuleviius, R.D., ed.: Advanced Information Systems Engineering Workshops. Springer, Cham (jun 2018) 222–234
25. Vaishnavi, V., Kuechler, B.: Design Science Research in Information Systems. Journal MIS Quarterly **28**(1) (2004) 75–105
26. Reimer, U., Laurenzi, E.: Creating and maintaining a collaboration platform via domain-specific reference modelling. In: EChallenges e-2014 Conference : 29-30 October 2014, Belfast, Ireland., IEEE (2014) 1–9
27. Hinkelmann, K., Laurenzi, E., Lammel, B., Kurjakovic, S., Woitsch, R.: A Semantically-Enhanced Modelling Environment for Business Process as a Service. In: 2016 4th International Conference on Enterprise Systems (ES), IEEE (2016) 143–152
28. Kritikos, K., Laurenzi, E., Hinkelmann, K.: Towards Business-to-IT Alignment in the Cloud. In: Advances in Service-Oriented and Cloud Computing. Springer, Cham (sep 2017) 35–52
29. Emmenegger, S., Hinkelmann, K., Laurenzi, E., Martin, A., Thönssen, B., Witschel, H.F., Zhang, C.: An Ontology-Based and Case-Based Reasoning Supported Workplace Learning Approach. In: Communications in Computer and Information Science. Springer, Cham (feb 2017) 333–354
30. Karagiannis, D., Buchmann, R.A.: A proposal for deploying hybrid knowledge bases: the adoxx-to-graphdb interoperability case. In: Proceedings of HICSS 2018, University of Hawaii. (jan 2018) 4055–4064
31. Fill, H.G., Schremser, D., Karagiannis, D.: A Generic Approach for the Semantic Annotation of Conceptual Models Using a Service-Oriented Architecture. International Journal of Knowledge Management **9**(1) (2013) 76–88
32. Hinkelmann, K.: Business Process Flexibility and Decision-Aware ModelingThe Knowledge Work Designer. In: Domain-Specific Conceptual Modeling. Springer International Publishing, Cham (2016) 397–414
33. Brachman, R.J.: The Basics of Knowledge Representation and Reasoning. AT&T Technical Journal **67**(1) (jan 1988) 7–24
34. Fanesi, D., Cacciagrano, D.R., Hinkelmann, K.: Semantic Business Process Representation to Enhance the Degree of BPM Mechanization - An Ontology. In: 2015 International Conference on Enterprise Systems (ES), IEEE (oct 2015) 21–32
35. Emmenegger, S., Hinkelmann, K., Laurenzi, E., Thönssen, B.: Towards a Procedure for Assessing Supply Chain Risks Using Semantic Technologies. In Fred, A., Dietz, J., Liu, K., Filipe, J., eds.: Knowledge Discovery, Knowledge Engineering and Knowledge Management. Volume 415 of Communications in Computer and Information Science. Springer Berlin Heidelberg (2013) 393–409