Formalizing Meta Models with FDMM: The ADOxx Case

Hans-Georg Fill[1], Timothy Redmond[2], and Dimitris Karagiannis[1]

[1] Research Group Knowledge Engineering, University of Vienna, Vienna, Austria
hgf|dk@dke.univie.ac.at,
WWW home page: http://www.dke.univie.ac.at
[2] Stanford Center for Biomedical Informatics Research, Stanford University, Stanford, USA
tredmond@stanford.edu,
WWW home page: http://bmir.stanford.edu

Abstract. This paper contains an extended and improved version of the description of the FDMM formalism presented at ICEIS'2012. FDMM is a formalism to describe how meta models and models are defined in the ADOxx approach as used in the Open Models Initiative. It is based on set theory and first order logic statements. In this way, an exact description of ADOxx meta models and corresponding models can be provided. In the paper at hand we extend the description of the formalism by illustrating how the mathematical statements can be used to support the implementation on the ADOxx platform. For this purpose we show how the FDMM constructs are mapped to statements in the ADOxx Library Language (ALL). As an example of the approach, the formalism and the mapping to ALL are applied to a modeling language from the area of risk management.

Key words: Conceptual Modeling, Meta Modeling, Domain-specific Modeling

1 Introduction

Conceptual models are today used in many areas of enterprise information systems [1, 2, 3]. Examples range from fields such as strategic management, business process and workflow management to enterprise architecture and software engineering. For these purposes a large variety of modeling languages have been developed and successfully applied in academia and industry [4]. When it comes to the sharing of such modeling languages and their corresponding models - as it has been recently promoted by the Open Models Intiative [5, 6] - the exact description of a modeling language and the models is one of the most important tasks. These descriptions not only reflect the design choices made during the implementation of the language. They also permit to compare and learn from different implementations of a modeling language and support the interpretation of the models [7].

In order to describe the building blocks of modeling languages it can be reverted to several types of *meta modeling* approaches [8]. These approaches provide the constructs necessary for describing the abstract and concrete syntax of a modeling language [9, 10]. In this way they also define constraints and correctness criteria for creating valid models based on the definition of a modeling language. In the context of the Open Models Initiative, several projects[1] have reverted to the freely available and industry proven ADOxx[2] meta modeling approach. At its core, the ADOxx approach allows to specify the syntax of a modeling language together with its graphical representation. From these specifications, visual model editors are then created automatically [11].

For the description of ADOxx based modeling languages, it has so far either been reverted to natural language descriptions, e.g. [12, 13], concrete implementations in the form of source code, e.g. [14, 15] or visual representations, e.g. [16, 17]. A formal description of the ADOxx meta modeling approach is so far not available. This is however necessary to analyze and evaluate how the syntax of a certain modeling language has been realized, compare ADOxx meta models and models to other meta modeling approaches such as GME, Ecore or ARIS cf. [8], derive suggestions for its enhancement and finally describe semantics for its further processing [18, 9].

Therefore we propose the FDMM[3] formalism that is capable of describing the core constitutents of the ADOxx approach. FDMM aims to provide an easy to use formalism that does not require specialized mathematical knowledge and that is capable of expressing the implementation of ADOxx meta models and models. The paper is structured as follows: In section 2 we will briefly discuss the foundations of modeling languages, meta models and models, the characteristics of the ADOxx approach and describe a running example for a modeling language from the area of enterprise information systems. Section 3 will describe the formalism including the necessary constraints and correctness criteria. In section 4 the formalism will be applied to the sample modeling language. In section 5 it will be shown how the formalism can be used as a basis for the implementation of the meta models in ADOxx using the ADOxx Library Language (ALL). The results of this implementation are then discussed in section 6. Work related to the approach will be part of section 7. The paper is concluded by an outlook on the future work in section 8.

2 Foundations

In this section we will briefly define the terms *modeling language*, *meta model* and *model* and describe the main characteristics of the ADOxx meta modeling approach. Finally, we will present a running example by using a modeling language from the area of risk management.

[1] See http://www.openmodels.at/web/omi/omp

[2] ADOxx is a registered trademark of BOC AG.

[3] The acronym FDMM stems from: A Formalism for Describing ADOxx Meta Models and Models

2.1 Modeling Language, Meta Model and Model

According to a framework proposed by [10], a modeling language is composed of a *syntax*, *semantics* and *notation*. The syntax specifies the elements and attributes of the language and the semantics assigns meaning to these constructs. In contrast to other frameworks, the notation is treated separately and is used to specify the visual representation of the language. The syntax further consists of an abstract syntax, which is represented by the *meta model*, and the concrete syntax, which is represented by a *model* [19, 9, 20]. The meta model can itself be described by a modeling language, i.e. the *meta modeling language*. Accordingly, the abstract syntax of the meta modeling language is represented by a *meta meta model* and the concrete syntax is represented by a meta model [8].

An example for these relationships is shown in figure 1: in the meta meta model the two entities *element* and *attribute* are defined. Additionally, a relation between the two entities is shown. On the meta model level the E_1 entity is defined as an element and the A_1 and A_2 entities as attributes that can be related to E_1. Finally, on the model level the entities ϵ_1 and ϵ_2 are defined as E_1 elements, the α_1 and α_2 entities as A_1 attributes and the β_1 and β_2 entities as A_2 attributes. The meta meta model thus defines which entities are provided for the specification of the abstract syntax of a modeling language in the form of a meta model. If the specification of the meta meta model is generic enough it can be used to specify a multitude of different modeling languages. A typical use case is then to automatically create model editors based on the static meta meta model specification and the dynamically adaptable meta model specifications.

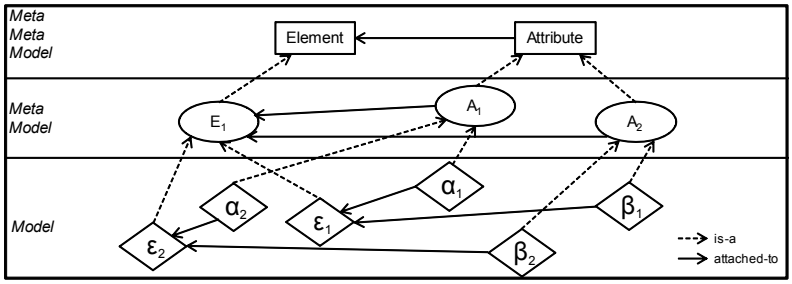

Fig. 1. Example for a Meta Meta Model, a Meta Model and a Model

In addition to *association mechanisms* for defining linkages between entities, meta meta models typically also provide *inheritance* and *containment* mechanisms [20]. By inheritance mechanisms, generalization and specialization relationships between entities of a meta model can be expressed to provide means for effecting polymorphic behaviors at model execution or interpretation time. This is required in particular for the design of algorithms that shall work on multiple, similar modeling languages without the need of particular adaptations: by defining an algorithm on a set of general entities that are shared by different meta models, the algorithm can be later bound automatically to entities that

are inherited from these general entities. Containment mechanisms refer to the inclusion of a set of entities into another entity on the model level. This is typically used to specify model/diagram types or aggregations/nestings that group sets of entities.

2.2 The ADOxx Meta Modeling Approach

The ADOxx meta modeling approach has originally been conceived in the course of the development of the ADONIS business process management toolkit in 1995. Since then it has been successfully used in a large number of academic and commercial projects by more than 1000 customers worldwide [21, 22]. The basic building blocks of its meta meta model are *classes* and *relationclasses* that are complemented with *attributes* [22]. Classes are organized in the form of an *inheritance hierarchy* so that the attributes of a super-class are inherited by its sub-classes in the sense of standard object orientation principles. Relationclasses are defined by their *from-class* and *to-class* attributes to specify valid instances of source and target classes. These relations can be complemented with cardinality constraints to limit the number of participating instances.

For collections of classes and relationclasses a containment mechanism in the form of *model types* is available. Model types define the context for the instantiation of classes and relationclasses in the form of *models*. Therefore, when creating a model, at first a model type has to be selected to specify which classes and relationclasses are valid in a model. Subsequently, these classes and relationclasses are instantiated as part of the model.

Besides the standard data types such as *integer*, *string*, and *double*, *enumeration* attributes are available in ADOxx that contain *pre-defined* values that can only be selected but not modified by a user during modeling. Furthermore, attributes can also be of two special types: attributes of the type *expression* contain strings in a proprietary expression grammar for automatically calculating the value of an attribute. Attributes of the type *interref* allow linking the instance of a class to another class instance of the same or of a different model instance or linking it to the same or a different model instance itself. The graphical representation of the instances of classes and relationclasses is specified via the particular string attribute named *GRAPHREP*. This attribute permits to specify context-dependent graphical representations for the classes and relationclasses, again based on a proprietary grammar - cf. [23]. Although an inherent part of the ADOxx approach, the graphical representation can thus be modified independently from the other entities.

With these characteristics, the following requirements were defined for a formalism that can describe the concepts of the ADOxx meta modeling approach: It should permit a formal description of the approach that is easy to handle and thus suitable for a wide range of users. Therefore, the formalism should focus on the core constituents to enable the description of arbitrary modeling languages that have been implemented using the ADOxx approach. It should however be extensible to allow its further development and future refinement.

2.3 Running Example: The 4R Modeling Language

As a running example we will revert to an existing modeling language in enterprise information systems from the area of governance, risk, and compliance (GRC). This example will first illustrate how meta models and models are typically used in information systems to create domain conceptualizations. In section 4 we will revert to it again for showing the application of the FDMM formalism.

In the last years particular consideration has been given to the management of risks and regulatory compliance together with their effects on returns and corresponding reporting requirements of enterprises. In line with these developments, the framework for *integrated enterprise balancing* has been derived [24, 13, 25]. The goal of this framework is to govern business activities with organization-wide consistent return and risk indicators. As a foundation, it is necessary to provide a common data basis that represents information from the areas of *r*isk, *r*eturn, *r*egulation, and *r*eporting - the so-called 4R information architecture. For acquiring this information, the *4R modeling language* [13] and the *4R situational method* for implementing such solutions in an organization were developed [25]. In its original form, the central parts of the 4R modeling language were illustrated by an extension of a graph based formalism. The corresponding realization using the ADOxx meta modeling approach was however described in natural language.

The meta model of the 4R modeling language, as it was specified in ADOxx, contains the following model types [13]: the *4R portfolio model*, the *4R business process model*, and the *4R organizational model*. Briefly summarized, the portfolio model type is used to describe multi-dimensional aggregations of the risks, returns and correlations of business transactions in regard to their relations to products and customers [24]. The single business transactions in this model can be linked to instances of the 4R business process model type. This second model type extends the process modeling language of business graphs [26] with elements, relations, and attributes for representing events, aggregations of events and their influence on the properties of process activties. The meta model is complemented with a 4R organizational model for representing actors, organizational units, resources, and roles that fulfill tasks in a business process.

For the implementation of the 4R model types on ADOxx the following classes and relationclasses together with several attributes were specified to represent the risk and return figures of business transactions and the underlying risk-affected business processes:

- for the portfolio model type the class *business transaction* and the relationclass *relates business transaction*,
- for the 4R business process model the super class *FlowObject* and as sub classes of this class: *Start*, *Decision*, *SubProcess*, *Activity*, *Parallelity*, *Join*, and *End*. Additionally the classes: *4R risk aggregation*, and *4R event*; the relationclasses: *subsequent*, *4R aggregation relation*, and *4R influences relation*

– for the 4R organizational model the classes: *actor*, *organizational unit*, *resource*, and *role*; and the relationclasses: *uses resouce*, *belongs to unit*, *has role*, and *has resource*

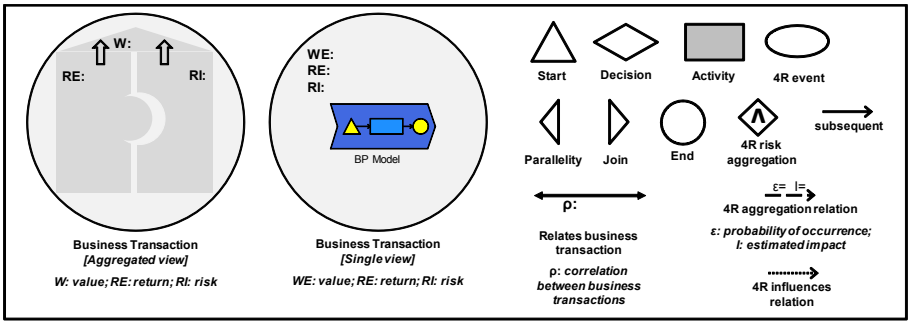

Fig. 2. Symbols of the 4R Portfolio Model Type and the 4R Business Process Model Type

To apply the FDMM formalism we will later regard the 4R portfolio model and the 4R business process model type in more detail. The classes and relationclasses of these model types are represented by the sets of symbols as shown in figure 2.

3 The FDMM Formalism

As stated in the introduction, the goal of this paper is to develop a formalism that is capable of describing the core constituents of ADOxx meta models and models as well as the criteria for valid models based on the meta model definitions. FDMM is therefore not a formalization of all aspects of the ADOxx approach, but a formalism that aims to support users of ADOxx to describe their meta models and models in a formal way. For the development of the formalism we reverted both to literature sources on the ADOxx meta modeling approach as well as existing implementations [2, 23, 11]. During the development we aimed for keeping the formalism as simple as possible while not sacrificing any of the core concepts of the approach.

3.1 Definition of Meta Models

At first we define the basic constituents of meta models **MM** which can then be used to derive model instances **mt**. We define a meta-model to be a tuple of the form

$$\mathbf{MM} = \langle \mathbf{MT}, \preceq, \text{domain}, \text{range}, \text{card} \rangle \tag{1}$$

where

– **MT** is the set of model types. We have

$$\mathbf{MT} = \{\mathbf{MT}_1, \mathbf{MT}_2, \ldots, \mathbf{MT}_m\} \tag{2}$$

where $\mathbf{MT}_i$ in turn is a tuple,

$$\mathbf{MT}_i = \left\langle \mathbf{O}_i^T, \mathbf{D}_i^T, \mathbf{A}_i \right\rangle \tag{3}$$

consisting of the set of object types $\mathbf{O}_i^T$, a set of data types $\mathbf{D}_i^T$, and a set of attributes $\mathbf{A}_i$. In this way ADOxx classes and relationclasses are uniformly represented as object types. As will be described below we will use the attributes $\mathbf{A}_i$ also for expressing associations between object types. When we describe the instantiation of the meta model in section 3.2 in model instances, the attributes will map the instantiation of object types to the instantiation of either object types or data types. This permits us to represent the ADOxx concepts of relationclasses and interref relations in the same way. However, this also means that, if for example directed relations are required, the notions of source and target object types have to be added when specifying a particular meta model. The object types, data types, and attributes are part of their respective total sets:

$$\mathbf{O}^T = \bigcup_j \mathbf{O}_j^T, \mathbf{D}^T = \bigcup \mathbf{D}_i^T, \mathbf{A} = \bigcup \mathbf{A}_i \tag{4}$$

– $\preceq$ is an ordering on the set of object types, $\mathbf{O}^T$. If $o_1^t \preceq o_2^t$ we say that the object type o_1^t is a subtype of the object type o_2^t. Thereby the inheritance hierarchy of ADOxx classes can be expressed. Due to the generic definition, this ordering can also be used for relationclasses: as ADOxx requires a from-class and a to-class attribute for relationclasses, a generic object type with these attributes can be defined and used for the definition of subtypes that inherit these attributes.
– the domain function maps attributes to the power set of all object types, i.e.

$$\text{domain} : \mathbf{A} \rightarrow \mathcal{P}(\bigcup_j \mathbf{O}_j^T) \tag{5}$$

The domain function will constrain what objects an attribute can map in the model instances. It is therefore used to attach attributes to a particular set of object types. In regard to ADOxx this corresponds to the assignment of ADOxx attributes to classes and relationclasses and the definition of an endpoint of an ADOxx relationclass.
– The range function maps an attribute to the power set of all pairs of object types and model types, all data types, and all model types

$$\text{range} : \mathbf{A} \rightarrow \mathcal{P}\left(\bigcup_j (\mathbf{O}_j^T \times \{\mathbf{MT}_j\}) \cup \mathbf{D}^T \cup \mathbf{MT}\right) \tag{6}$$

In the model instances, the range function will constrain what values an attribute can take. For the definition of a meta model it is thus used to specify the type of an attribute. I.e. whether the attribute has the function of specifying the relation of an instance of an object type in a model instance to either: *a.* the instances of object types in a particular model type, *b.* instances of data types or *c.* instances of model types. Case *a.* thus corresponds to both the relationclass and interref concepts in ADOxx that have an instance of a class as a target, case *b.* to the ADOxx attribute concepts except interref attributes, and case *c.* to the ADOxx interref attribute concept that has an instance of a model type as a target.

– The card function maps pairs of object types and attributes to pairs of integers

$$\text{card} : O^T \times A \rightarrow \mathcal{P}(\mathbb{N} \times (\mathbb{N} \cup \{\infty\})) \tag{7}$$

where $\mathbb{N}$ is the set of non-negative integers. In the model instances the card function will constrain how many attribute values a object can have. In regard to the different types of attributes this thus permits to specify how many instances of object types, of data types or of model types an attribute can contain. When comparing this to the ADOxx approach, the card function determines whether the value of an attribute corresponds to either: *a.* a target of a relationclass - that can only be one distinct class, *b.* the target of an interref attribute that can have multiple values or *c.* the instance of a datatype that can also have multiple values, which corresponds to the enumeration attribute type in ADOxx.

In addition we define the following correctness criteria for meta models: The sets of object types, data types, and attributes have to be pairwise disjoint

$$\mathbf{O}^T \cap \mathbf{D}^T = \emptyset,\ \mathbf{O}^T \cap \mathbf{A} = \emptyset,\ \mathbf{D}^T \cap \mathbf{A} = \emptyset \tag{8}$$

This follows from the fact that in mathematical terms we have so far only been defining various sets that could overlap. In addition, for any attribute a that is part of the attribute set $\mathbf{A}_i$ of the i-th model type - see equation 3, the domain function for that attribute must point to any of the object types in that model type

$$a \in \mathbf{A}_i \Rightarrow \text{domain}(a) \subseteq \mathbf{O}_i^T \tag{9}$$

That ensures that attributes that are related by the domain function to a certain object type are part of the same model type definition. This corresponds directly to the ADOxx approach in the way that all concepts that are relevant for the definition of models are grouped within the context of a model type.

3.2 Instantiation of Meta Models

We will now describe the instantiation of a meta model. The instantiation of a meta model essentially describes the mapping of the model types, object types,

and data types to model instances, objects, and data values together with a set of triples. Thus, an instantiation of a metamodel **MM** will be a tuple

$$\left\langle \mu_{\mathbf{mt}}, \mu_{\mathbf{O}}, \mu_{\mathbf{D}}, \mathcal{T}, \beta \right\rangle \tag{10}$$

where

- $\mu_{\mathbf{mt}}$ is a one-to one mapping from model types to the power set of model instances:

$$\mu_{\mathbf{mt}} : \mathbf{MT} \rightarrow \mathcal{P}(\mathbf{mt}) \tag{11}$$

  Thereby it is defined that a model instance must be of one specific model type and that there may be several instances of one model type.
- $\mu_{\mathbf{O}}$ is a function taking the object types in a given model type to collections of objects:

$$\mu_{\mathbf{O}} : \bigcup_j \left(\mathbf{O}_j^T \times \{\mathbf{MT}_j\}\right) \rightarrow \mathcal{P}(\mathbf{O}) \tag{12}$$

  where

$$\mathbf{O} = \bigcup_j \mu_{\mathbf{O}}(\mathbf{O}_j^T \times \{\mathbf{MT}_j\}) \tag{13}$$

  Thereby, the objects are defined as instances of an object type $\mathbf{O}_j^T$ that is part of a particular model type $\mathbf{MT}_j$ - see also equation 3. The addition of the model type is necessary as object types may be part of multiple model types and in the ADOxx approach objects can only occur within a model instance.
  Sometimes it is convenient to create an object type which is meant to be subtyped but which is not meant to be directly instantiated. The purpose of such a type is to capture information that is common to all the subtypes. Such a type is called an *abstract type* and we can define what it means to be an abstract type based on the definitions above. An object type $o^t \in \mathbf{O}^T$ is said to be abstract if for all model types $\mathbf{MT}_i$ which contain the object type o^t ($o^t \in O_i^T$) we have

$$\mu_{\mathbf{O}}(o^t, \mathbf{MT}_i) = \bigcup_{o_1^t \neq o^t, o_1^t \preceq o^t} \mu_{\mathbf{O}}(o_1^t, \mathbf{MT}_i). \tag{14}$$

  That is to say that all the objects that instantiate o^t must instantiate o^t through one of its subtypes. In terms of ADOxx the notion of abstract types corresponds to super classes of which one or more of their sub classes are included in a model type but who cannot be instantiated themselves.
- $\mu_{\mathbf{D}}$ maps data types to a power set of data objects

$$\mu_{\mathbf{D}} : \mathbf{D}^T \rightarrow \mathcal{P}(\mathbf{D}) \tag{15}$$

  The data types are not further constrained. It is thus left to the user of the formalism to ensure the correct content of a type, e.g. whether an 'integer' type contains only integer numbers. The formalism will only ensure that a data object is assigned a type that is valid in a particular context.

- $\mathcal{T} \subseteq \mathbf{O} \times \mathbf{A} \times (\mathbf{D} \cup \mathbf{O} \cup \mathbf{mt})$ is a set of triples. These triples will later be used to describe the contents of model instances.
- $\beta : \mathbf{mt} \to \mathcal{P}(\mathcal{T})$. This map describes how the triples are assigned to the model instances.

We will additionally define a collection of correctness constraints on the instantiation of the meta model. These constraints fall into two categories: disjointness constraints that describe how a model instance is partitioned and domain/range/cardinality constraints that constrain how attributes can map objects to other objects and data values.

The following constraints define the disjointness and partitioning constraints that must be enforced for the various parts of the meta model instantiation:

- The instances of object types and the instances of datatypes must be disjoint, i.e. instances of object types and instances of datatypes cannot be the same.

$$\mu_{\mathbf{O}}(o^t, MT_j) \cap \mu_{\mathbf{D}}(d^t) = \emptyset \tag{16}$$

- The instances of two object types are disjoint if either the object types are disjoint or if their model types to which they belong are disjoint, i.e. the formalism does not permit the instantiation from multiple object types nor a 'reuse' of objects of the same object type for different model instances:

$$i \neq j \vee o_1^t \neq o_2^t \Rightarrow \mu_{\mathbf{O}}(o_1^t, \mathbf{MT}_i) \cap \mu_{\mathbf{O}}(o_2^t, \mathbf{MT}_j) = \emptyset \tag{17}$$

- For two different model types $\mathbf{MT}_i$ and $\mathbf{MT}_j$ also the corresponding model instances must be disjoint, i.e. also for model instances no instantiations from multiple model types are allowed:

$$\mathbf{MT}_i \neq \mathbf{MT}_j \Rightarrow \mu_{\mathbf{mt}}(\mathbf{MT}_i) \cap \mu_{\mathbf{mt}}(\mathbf{MT}_j) = \emptyset \tag{18}$$

- Every element of the set of model instances $\mathbf{mt}$ has to be derived from a model type, i.e. there cannot be model instances without a corresponding model type:

$$\mathbf{mt} = \bigcup \mu_{\mathbf{mt}}(\mathbf{MT}_j) \tag{19}$$

- For two different data types it must follow that also their instances are disjoint, i.e. also for data types it is not allowed that instances can be derived from multiple types:

$$d_1^t \neq d_2^t \Rightarrow \mu_{\mathbf{D}}(d_1^t) \cap \mu_{\mathbf{D}}(d_2^t) = \emptyset \tag{20}$$

- $\mathcal{T}$ is the disjoint union of $\beta(mt_i)$ where $mt_i \in \mathbf{mt}$. More colloquially every triple is contained in exactly one model instance.

The following constraints define the inheritance, domain, range and cardinality constraints that the meta model instantiation must satisfy:

– if the object type $o_1^t \in \mathbf{O}_j^T$ is a subtype of the object type $o_2^t \in \mathbf{O}_j^T$ ($o_1^t \preceq o_2^t$) then we have

$$\mu_{\mathbf{O}}(o_1^t, \mathbf{MT}_j) \subseteq \mu_{\mathbf{O}}(o_2^t, \mathbf{MT}_j). \tag{21}$$

– Sibling object types are disjoint. More specifically if $o_1^t, o_2^t, o_3^t \in \mathbf{O}_j^T$ are object types such that

$$o_2^t \preceq o_1^t, o_3^t \preceq o_1^t, o_2^t \not\preceq o_3^t, o_3^t \not\preceq o_2^t \tag{22}$$

then

$$\mu_{\mathbf{O}}(o_2^t, \mathbf{MT}_j) \cap \mu_{\mathbf{O}}(o_3^t, \mathbf{MT}_j) = \emptyset. \tag{23}$$

– If the value y of a statement is an object, i.e. there is a mapping from an object type to an object for a concrete model type $\mathbf{MT}_j$, then the pair of an object type and a model type have to be part of the range definition in the meta model:

$$(x\ a\ y) \in \mathcal{T} \wedge y \in \mathbf{O} \Rightarrow \exists o^t, MT_j, (y \in \mu_{\mathbf{O}}(o^t, MT_j) \wedge (o^t, MT_j) \in range(a)) \tag{24}$$

The second equation further defines that if y points to an object, then there must exist an object type o^t and a model type MT_j that are part of an $\mu_{\mathbf{O}}$ mapping for y.

– If the value y of a statement is a data object then there must exist a datatype that is part of the range definition of the attribute in the meta model and there must be a mapping between the data type and the data object:

$$(x\ a\ y) \in \mathcal{T} \wedge y \in \mathbf{D} \Rightarrow \exists d^t \in \mathbf{D}^T\ d^t \in \text{range}(a) \wedge y \in \mu_{\mathbf{D}}(d^t) \tag{25}$$

– If the value y of a statement is a model instance $\mathbf{mt}$, then a model type $\mathbf{MT}_j$ must be part of the range definition and the y value must correspond to the mapping of that model type to the model instance:

$$(x\ a\ y) \in \mathcal{T} \wedge y \in \mathbf{mt} \Rightarrow \exists \mathbf{MT}_j \in \text{range}(a), y \in \mu_{\mathbf{mt}}(\mathbf{MT}_j) \tag{26}$$

– For each statement the attribute a of that statement must be part of the same model type from which the object x has been mapped:

$$(x\ a\ y) \in \mathcal{T} \Rightarrow \exists j\ a \in \mathbf{A}_j \wedge \exists o^t \in \text{domain}(a), x \in \mu_{\mathbf{O}}(o^t, \mathbf{MT}_j) \tag{27}$$

– If the value y of a statement is a data object then the data type must be part of the same model type as the attribute:

$$(x\ a\ y) \in \mathcal{T}, a \in \mathbf{A}_i, y \in \mathbf{D} \Rightarrow \exists d^t \in \mathbf{D}_i^T, y \in \mu_{\mathbf{D}}(d^t) \tag{28}$$

– And for the cardinality constraints: if $x \in \mu_{\mathbf{O}}(o^t, MT_j)$, $a \in \mathbf{A}_i$ where $\langle m, n \rangle = card(o^t, a)$ then $m \leq |\{y\ :\ (x\ a\ y) \in \mathcal{T} \wedge y \in (\mathbf{O} \cup \mathbf{D} \cup \mathbf{mt})\}| \leq n$.

4 Application of FDMM to the 4R Modeling Language

To illustrate the usage of the FDMM formalism we will show in the following how the modeling language from the running example in section 2.3 and instances of this modeling language can be formally described. We start by defining the model types that will represent the *4R portfolio models* and *4R business process models* by $\mathbf{MT}_{PO}$ and $\mathbf{MT}_{BP}$:

$$\mathbf{MT}_{\mathrm{PO}} = \left\langle \mathbf{O}^T_{\mathrm{PO}}, \mathbf{D}^T_{\mathrm{PO}}, \mathbf{A}_{\mathrm{PO}} \right\rangle, \mathbf{MT}_{\mathrm{BP}} = \left\langle \mathbf{O}^T_{\mathrm{BP}}, \mathbf{D}^T_{\mathrm{BP}}, \mathbf{A}_{\mathrm{BP}} \right\rangle \tag{29}$$

Next, we detail the sets of object types $\mathbf{O}^T_{\mathrm{PO}}$ and $\mathbf{O}^T_{\mathrm{BP}}$ for expressing what corresponds to the classes and relationclasses in ADOxx by:

$$\begin{aligned}\mathbf{O}^T_{PO} =& \{Business\text{-}transaction, relates\text{-}business\text{-}transaction\}\\ \mathbf{O}^T_{BP} =& \{FlowObject, Start, Decision, Activity, Parallelity, Join, End,\\ & 4R\text{-}event, 4R\text{-}risk\text{-}aggregation, subsequent, aggregation\\ & influences\}\end{aligned} \tag{30}$$

Thereby, the object type *FlowObject* is defined as an abstract type that has to be instantiated through one of its sub types. In addition the following subtype relationships hold between the following object types:

$$\begin{aligned}Start \preceq FlowObject, Decision \preceq FlowObject, Activity \preceq FlowObject,\\ Parallelity \preceq FlowObject, Join \preceq FlowObject, End \preceq FlowObject\end{aligned} \tag{31}$$

The same is applied for detailing the sets of data types $\mathbf{D}^T_{PO}$ and $\mathbf{D}^T_{BP}$. Thereby, the $\mathbf{Enum}_{view}$ and $\mathbf{Enum}_{influence}$ types are used to represent the ADOxx enumeration attribute types with pre-defined values:

$$\begin{aligned}\mathbf{D}^T_{PO} =& \{String, Float, \mathbf{Enum}_{view}\}\\ \mathbf{Enum}_{view} =& \{Aggregated, Single\}\\ \mathbf{D}^T_{BP} =& \{String, Float, \mathbf{Enum}_{influence}\}\\ \mathbf{Enum}_{influence} =& \{Time\text{-}influence, Cost\text{-}influence,\\ & Return\text{-}influence, Quality\text{-}influence\}\end{aligned} \tag{32}$$

We continue by detailing the sets of attributes $\mathbf{A}_{PO}$ and $\mathbf{A}_{BP}$:

$$\begin{aligned}\mathbf{A}_{PO} =& \{ID, W, RE, RI, WE, \rho, relates\text{-}from, relates\text{-}to, Process, View\}\\ \mathbf{A}_{BP} =& \{Name, \epsilon, I, Time, Cost, Return, Quality, Influence\text{-}type,\\ & subsequent\text{-}from, subsequent\text{-}to, aggregation\text{-}from,\\ & aggregation\text{-}to, influences\text{-}from, influences\text{-}to\}\end{aligned} \tag{33}$$

For attaching the attributes to the object types and defining their value range, we add according domain and range definitions. This can be done for example by attaching the *Name* attribute to the required object type and then defining

its range to be of the data type *String*. By using the *FlowObject* abstract type we can do this for all object types that are defined as its subtypes:

$$\begin{aligned} \text{domain}(Name) &= \{FlowObject, 4R\text{-}risk\text{-}aggregation, 4R\text{-}event\} \\ \text{range}(Name) &= \{String\} \end{aligned} \tag{34}$$

We then add the cardinality definitions for each of the object types and their attributes as shown here exemplarily for the name attribute:

$$\begin{aligned} &\text{card}(FlowObject, Name) = \langle 1,1\rangle, \text{card}(4R\text{-}risk\text{-}aggregation, Name) = \langle 1,1\rangle, \\ &\text{card}(4R\text{-}event, Name) = \langle 1,1\rangle \end{aligned} \tag{35}$$

As it has been done for the attributes of the data type *String* we can similarly define the domain, range, and cardinality functions for an attribute of the type *Float*. As already mentioned above, the FDMM formalism does not further specify the data types so that we would have for example:

$$\begin{aligned} &\text{domain}(W) = \{Business\text{-}transaction\}, \text{range}(W) = \{Float\} \\ &\text{card}(Business\text{-}transaction, W) = \langle 0,1\rangle \end{aligned} \tag{36}$$

In the same way, ADOxx attributes with pre-defined values can be represented in FDMM as shown in the following by inserting the data type set $\mathbf{Enum}_{view}$ in the range definition to specify the type of view that is used for a business transaction:

$$\begin{aligned} &\text{domain}(View) = \{Business\text{-}transaction\}, \text{range}(View) = \{\mathbf{Enum}_{view}\} \\ &\text{card}(Business\text{-}transaction, View) = \langle 1,1\rangle \end{aligned} \tag{37}$$

To permit references from one object to another model instance, e.g. to reference business transactions to corresponding 4R business process models, the following domain and range definitions are needed:

$$\begin{aligned} &\text{domain}(Process) = \{Business\text{-}transaction\}, \text{range}(Process) = \{\mathbf{MT}_{\text{BP}}\} \\ &\text{card}(Business\text{-}transaction, Process) = \langle 0,1\rangle \end{aligned} \tag{38}$$

Finally, we also give an example for defininig the equivalent of a relationclass based on an object type that connects to two other object types via "to" and "from" attributes:

$$\begin{aligned} \text{domain}(influences\text{-}from) &= \{influences\} \\ \text{range}(influences\text{-}from) &= \{(4R\text{-}risk\text{-}aggregation, \mathbf{MT}_{\text{BP}})\} \\ \text{card}(influences, influences\text{-}from) &= \langle 1,1\rangle \\ \text{domain}(influences\text{-}to) &= \{influences\} \\ \text{range}(influences\text{-}to) &= \{(Activity, \mathbf{MT}_{\text{BP}})\} \\ \text{card}(influences, influences\text{-}to) &= \langle 1,1\rangle \end{aligned} \tag{39}$$

Also for such an object type, that corresponds to a relationclass, attributes can be added in the same way as shown above:

$$\begin{aligned} \text{domain}(Influence\text{-}type) &= \{influences\} \\ \text{range}(Influence\text{-}type) &= \{\mathbf{Enum}_{influence}\} \\ \text{card}(influences, Influence\text{-}type) &= \langle 1,1 \rangle \end{aligned} \tag{40}$$

When defining relationclasses that can be used to connect multiple classes, the definition can be simplified by reverting to a supertype class as for example the *FlowObject* class and the *subsequent* relationclass:

$$\begin{aligned} \text{domain}(subsequent\text{-}from) &= \{subsequent\} \\ \text{range}(subsequent\text{-}from) &= \{(FlowObject, \mathbf{MT}_{\text{BP}})\} \\ \text{card}(subsequent, subsequent\text{-}from) &= \langle 1,1 \rangle \\ \text{domain}(subsequent\text{-}to) &= \{subsequent\} \\ \text{range}(subsequent\text{-}to) &= \{(FlowObject, \mathbf{MT}_{\text{BP}})\} \\ \text{card}(subsequent, subsequent\text{-}to) &= \langle 1,1 \rangle \end{aligned} \tag{41}$$

Based on these definitions for the model type we can describe the instantiation of a concrete model. As an example we use two models that have been described in [13] - see figures 3 and 4. They represent sample instances of a 4R portfolio model type and a 4R business process model type that shows how 4R events and 4R risk aggregations are used to represent the influence of risks on the accomplishment of activities. We will use these models to describe some of its contents by using the FDMM formalism.

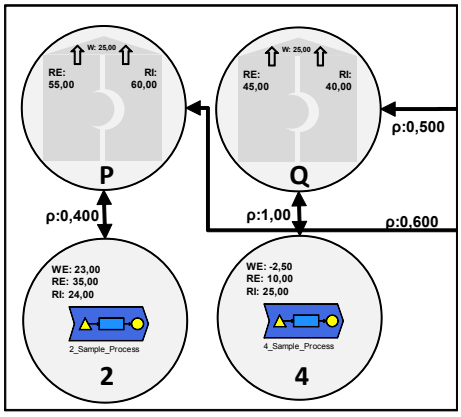

Fig. 3. Excerpt of a 4R Portfolio Model [13]

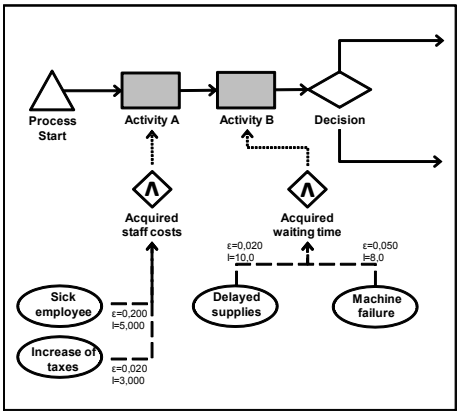

Fig. 4. Excerpt of a 4R Business Process Model [13]

First we instantiate concrete models for the model types $\mathbf{MT}_{PO}$ and $\mathbf{MT}_{BP}$ based on a mapping from the meta model definition:

$$\mu_{\mathbf{MT}}(\mathbf{MT}_{PO}) = \{\mathbf{mt}_{po1}\}, \mu_{\mathbf{MT}}(\mathbf{MT}_{BP}) = \{\mathbf{mt}_{bp1}\} \tag{42}$$

Next, we instantiate objects based on mappings from the object types. We show this exemplarily for some instances of the business transaction, activity, 4R event, and 4R aggregation object types:

$$\begin{aligned} \mu_{\mathbf{O}}(\textit{Business-transaction}, \mathbf{MT}_{PO}) &= \{BT_P, BT_Q\} \\ \mu_{\mathbf{O}}(\textit{Activity}, \mathbf{MT}_{BP}) &= \{\textit{Activity}_A, \textit{Activity}_B\} \\ \mu_{\mathbf{O}}(\textit{4R-event}, \mathbf{MT}_{BP}) &= \{\textit{Machine-failure}, \textit{Delayed-supplies}\} \\ \mu_{\mathbf{O}}(\textit{4R-aggregation}, \mathbf{MT}_{BP}) &= \{\textit{Acquired-waiting-time}\} \end{aligned} \tag{43}$$

Similarly we can instantiate object types that can later act as relations such as the influences type:

$$\mu_{\mathbf{O}}(influences, \mathbf{MT}_{BP}) = \{influences_1, influences_2\} \tag{44}$$

Subsequently, we also show the mappings of some data types to data objects in order to later assign them as values for attributes:

$$\begin{aligned} \mu_{\mathbf{D}}(\textit{String}) &= \{'\textit{ActivityA}', '\textit{ActivityB}', '\textit{DelayedSupplies}', \\ &\quad '\textit{Machinefailure}', \textit{Acquired-waiting-time}\} \\ \mu_{\mathbf{D}}(\textit{Float}) &= \{25.00, 55.00, 60.00\} \\ \mu_{\mathbf{D}}(\textit{Time-influence}) &= \{'\textit{time-influence}'\} \end{aligned} \tag{45}$$

And finally we can define the relationships between the objects and the data object by using the attributes in the form of triple statements, e.g. to express the names of concrete objects and the values of attributes and defining relations:

$$\begin{aligned} (\textit{Activity}_A \;\; \textit{Name} \;\; '\textit{ActivityA}') \in \beta(\mathbf{mt}_{bp1}), (BT_P \;\; W \;\; 25.00) \in \beta(\mathbf{mt}_{po1}), \\ (influences_1 \;\; \textit{influences-from} \;\; \textit{Acquired-waiting-time}) \in \beta(\mathbf{mt}_{bp1}), \\ (influences_1 \;\; \textit{influences-to} \;\; \textit{Activity}_B) \in \beta(\mathbf{mt}_{bp1}) \end{aligned} \tag{46}$$

And for detailing the type of influence by specifying the attribute value that is available based on a pre-defined data type:

$$(influences_1 \;\; \textit{Influence-type} \;\; '\textit{time-influence}') \in \beta(\mathbf{mt}_{bp1}) \tag{47}$$

5 FDMM for Implementations on the ADOxx Platform

The ADOxx meta modeling approach as it has been described in section 2.2 has been implemented in the form of an industry-ready, client-server based software platform. One core functionality of the platform is, that it permits to specify meta models in a proprietary definition language and automatically generates

according model editors. Both meta models and models are thereby automatically stored in a relational database. In addition, the platform provides a number of extension functionalities such as a proprietary scripting language for defining mechanisms and algorithms on models, a generic query language for models, documentation functionalities for generating reports about models or a web service interface for establishing couplings to third party tools. These functionalities are complemented by a multi-user rights management for controlling the accessibility to the different platform components and the models.

In the following we will illustrate how the specifications using the FDMM formalism can be used to implement meta models on the ADOxx platform. We start by showing how the model type for 4R business process models as defined in equation 29 is translated into the ADOxx Library Language (ALL). These ALL definitions are then used by the ADOxx platform to create the meta model and automatically provide according model editors for the thus specified model types:

Example Definition of the 4R Business Process Model Type in ALL

```
BUSINESS PROCESS LIBRARY <4R-IEB-Library>
ATTRIBUTE <Version number>
VALUE ""
...
ATTRIBUTE <Modi>
VALUE "MODELTYPE \"4R Business process model\" from:none
plural:\"4R Business process models\"
INCL \"Start\"
INCL \"Decision\"
INCL \"Activity\"
INCL \"Parallelity\"
INCL \"Join\"
INCL \"End\"
INCL \"Subsequent\"
INCL \"4R risk aggregation\"
INCL \"4R event\"
INCL \"aggregation\"
INCL \"influences\"
MODE \"All modelling objects\" from:all
MODE \"Documentation\" from:all no-modeling"
...
```

The 4R business process model type is thereby defined to contain a number of *classes* that are not abstract and *relationclasses* and can therefore be instantiated to object instances in a model editor. When mapping the definitions in FDMM to ALL, it has to be decided, which object types in FDMM become classes and which become relationclasses in ALL. As relationclasses in ALL can only represent binary relations between object types of the cardinality $< 1, 1 >$, only object types that satisfy this restriction can be mapped to a relationclass.

For other relations between object types and between object types and model types, ALL offers the *interref* attribut type. As a rule of thumb, FDMM object types that correspond to edges in graph-like model types can be mapped to relationclasses.

The definition of classes in ALL is accomplished using the 'CLASS' keyword. As shown in the example code below, classes are arranged in a superclass/subclass hierarchy using a colon followed by the name of the super class. Thereby, every class in ALL has to be a direct or indirect subclass of the predefined "__BP-construct__" class. These hierarchy definitions correspond to the definition of equation 31 as shown for the FlowObject class and the Start class:

Example Definition of the Start Class in ALL

```
CLASS <FlowObject> : <__BP-construct__>
...
CLASS <Start> : <FlowObject>
...
```

For the classes also the corresponding attributes as defined in FDMM by the domain, range, and cardinality statements can be added in ALL. Thereby, the name attribute is automatically added by ADOxx as it constitutes the unique identifier of instances of a class. For every attribute in ALL, the type and potential additional information have to be specified. We show this in the following example for the definition of the Business transaction class as specified by the equations 36, 37:

Example Definition of the Business Transaction Class and the W and View Attributes in ALL

```
CLASS <Business transaction> : <__BP-construct__>
...
ATTRIBUTE <W>
TYPE DOUBLE
VALUE 0
...
ATTRIBUTE <View>
TYPE ENUMERATION
FACET <EnumerationDomain>
VALUE "Aggregated@Single"
...
```

As has already been mentioned above, relations between object types that do not satisfy the cardinality restrictions of relationclasses in ALL or that relate object types and model types, have to be specified as interref attributes in ALL. Also, in case a model type should not contain visible relations between class instances, this attribute type can be chosen. We show this for the Process attribute of the business transaction model type that relates an instance of a

Business Transaction class to a 4R business process model type as specified in equation 38:

Example Definition of the Process Interref Attribute in ALL

```
...
ATTRIBUTE <Process>
TYPE INTERREF
...
FACET <AttributeInterRefDomain>
VALUE "REFDOMAIN MODREF
     mt:\"4R Business process model\"
     max:1"
...
```

The definition of relationclasses in ALL requires the definition of a start class, indicated by the FROM keyword, and a target class, indicated by the TO keyword. We show this by the example of the influences relationclass that is based on the specifications in equation 39 and the definition of the Influence type attribute in equation 40:

Example Definition of the Influences Relationclass in ALL

```
RELATIONCLASS <influences>
FROM <4R risk aggregation>
TO <Activity>
...
ATTRIBUTE <Influence type>
TYPE ENUMERATION
FACET <EnumerationDomain>
VALUE "Time influence@Cost influence@Return influence@
Quality influence"
...
```

In order to permit the automatic generation of model editors from the ALL definitions, two more aspects have to be added. First, it has to be defined, which attributes of a class or a relationclass can be edited by a user in the model editor. This is defined via the AttrRep attribute as shown below for the example of the Influences relationclass:

Example Definition of the AttrRep Attribute of the Influences Relationclass in ALL

```
...
ATTRIBUTE <AttrRep>
TYPE STRING
VALUE "NOTEBOOK
ATTR \"Influence type\""
...
```

Second, also a graphical representation has to be added if the classes and relationclasses should be represented in a graphical way in the model editor. This is defined via the GraphRep class attribute that contains a proprietary grammar for defining the visual representation of a class or a relationclass. It can either be coded by hand or defined using a visual editor as provided by the Open Models Initiative[4]:

Example Definition of the GraphRep Classattribute of the 4R Event Class in ALL

```
CLASS <4R event> : <_4R-Superclass_>
...
CLASSATTRIBUTE <GraphRep>
VALUE "GRAPHREP
PEN w:0.07cm
ELLIPSE x:0cm y:0cm rx:2cm ry:1cm
ATTR \"Name\" h:c w:c"
```

Finally, when all ALL definitions are in place, the ADOxx platform automatically generates model editors for the defined model types. As an example, an editor for the 4R business process models is shown in figure 5.

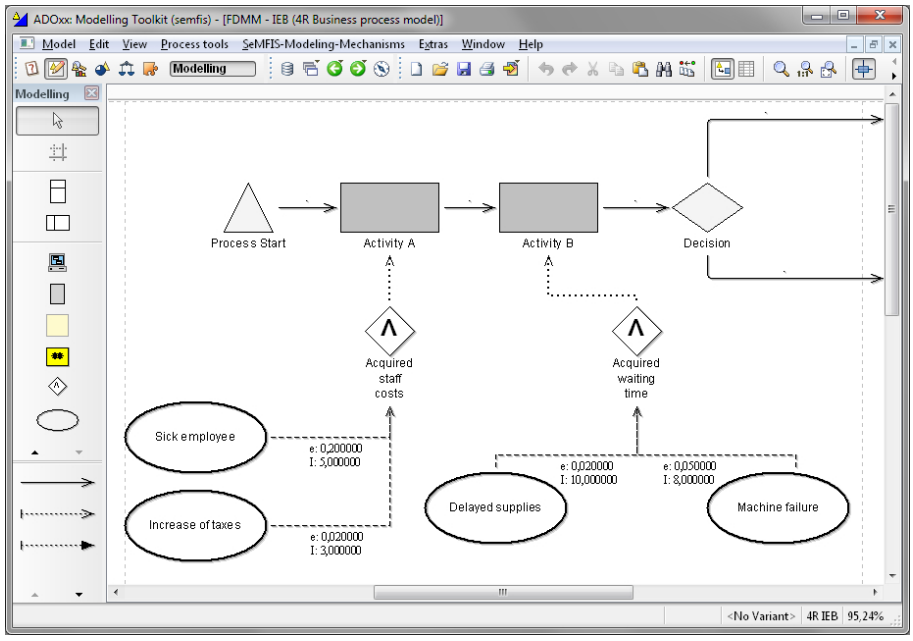

Fig. 5. Screenshot of the Model Editor for 4R Business Process Models on ADOxx

[4] See `http://www.openmodels.at/web/omi/services`

6 Discussion

Based on the constructs of the FDMM formalism it is possible to specify ADOxx meta models and models in a mathematically rigorous way that builds only upon set theory and first order logic statements. With these mathematical foundations, the ADOxx meta modeling approach can be compared to other approaches and used as a basis for further formal specifications or implementations. As has been discussed in [27], the FDMM formalism differs in several aspects from the ADOxx Library Language. In the paper at hand this could be shown in particular for the translation of object types into classes and relationclasses in ALL. As ALL requires the definition of relationclasses for the visualization of edges, not all object types specified in FDMM can be used for this purpose. Therefore, at this point it has to be decided by the developer of the ALL definition, how object types should be translated into ALL classes and relationclasses. In addition, also graphical representations and the definitions for editable attributes have to be added during this translation as they are not contained in FDMM. Regarding the graphical representations, it could thereby also reverted to the approach of semantic visualization that permits to automatically assign graphical representations based on semantic annotations [23].

7 Related Work

When comparing FDMM and ADOxx to similar approaches in the literature, two directions can be taken. The first is the comparison to other meta modeling approaches and the second is the comparison to other kinds of formalizations for meta modeling approaches.

Based on the classification proposed by [20], the FDMM and the ADOxx meta modeling approach directly compare to *domain-specific modeling approaches* that view meta models as language specifications. This is in contrast to approaches that treat meta models as *software structure specifications*, which is the typical use case for approaches such as EMOF [28], EMF [29] or KM3 [30]. A common aspect of domain-specific modeling approaches is the creation of visual model editors from meta models that are based on one pre-defined meta meta model and that use a graphical representation for the concrete syntax of the defined language. [8] also denote this direction as *heavyweight approaches* of language definition and distinguish it from *lightweight approaches* that adapt a generic meta model with domain-specific concepts. An example for the latter direction would be the use of the profile package in UML, e.g. to extend existing meta classes with the stereotyping mechanism [31].

FDMM and ADOxx can be directly compared to the approaches analyzed in [8]: thereby a core feature of ADOxx and FDMM that is shared with the GME and ARIS meta modeling approaches is the use of model types for defining the grouping of object types and their instances. In contrast to all approaches compared by Kern et al. and ADOxx, FDMM does not use any relation concept as a first class concept. Neither ADOxx nor FDMM use explicit *role type* concepts

that provide further mechanisms for specifying relationships such as semantic dependencies between object types. However, in ADOxx such concepts can be expressed using the ADO*script* language and enforced during modeling.

For the meta modeling approaches mentioned above, formalizations have been discussed for EMOF e.g. [32, 33] and KM3 [30]. However, they differ from ADOxx in regard to their focus on the specification of software structures. Another approach that shows some similarities to the way the FDMM formalism has been conceived can also be found in the specification of the Object Constraint Language (OCL) [34][Annex A]. However, the main difference is that FDMM is directed towards supporting the representation of meta models and models. The OCL specification does not describe a meta modeling approach but rather an approach to formalize one particular modeling language, i.e. UML together with constraints.

Furthermore, the domain, range, and card functions and the associated constraints described for them have a similarity with the notions of domain, range and cardinality restrictions used in description logics [35]. In contrast to the description logic case, our work is not intended to give a semantics for some formal language. Instead it is intended to provide a formal description of an existing system that has been effectively used in several application domains.

8 Conclusion and Outlook

In this paper we presented a formalism to describe the core constituents of the ADOxx meta modeling approach and showed its application to a concrete modeling language as well as how it can act as a basis for an implementation. It is the first formal definition for ADOxx meta modeling concepts and is therefore expected to be of benefit also for other projects using the ADOxx approach. Future work will therefore include the application to further modeling languages and the evaluation of the usability of the formalism. This concerns in particular the definition of algorithms, e.g. for describing analyses and simulations of models. Finally, it will also be investigated how the formalism can be represented visually to enhance the interaction with it and enable the easy re-use of formal meta model and model statements.

Acknowledgement

Parts of the work on this paper have been funded by the Austrian Science Fund (FWF) in the course of an Erwin-Schrödinger fellowship project number J3028-N23.

References

1. Kaschek, R.: On the evolution of conceptual modeling. In: Dagstuhl Seminar Proceedings. Volume 08181. (2008)

2. Karagiannis, D., Fill, H.G., Hoefferer, P., Nemetz, M.: Metamodeling: Some application areas in information systems. In Kaschek, R., et al., eds.: UNISCON. Springer (2008) 175–188
3. Wand, Y., Weber, R.: Research commentary: Information systems and conceptual modeling - a research agenda. Information Systems Research **13**(4) (2002) 363–376
4. Borgida, A., Chaudhri, V., Giorgini, P., Yu, E.: Conceptual Modeling: Foundations and Applications. Springer (2009)
5. Koch, S., Strecker, S., Frank, U.: Conceptual Modelling as a New Entry in the Bazaar: The Open Model Approach. In: Open Source Systems. Volume 203/2006. IFIP International Federation for Information Processing (2006) 9–20
6. Karagiannis, D., Grossmann, W., Hoefferer, P.: Open model initiative - a feasibility study (04-04-2010 2008) http://cms.dke.univie.ac.at/uploads/media/Open_Models_Feasibility_Study_SEPT_2008.pdf.
7. Hinkelmann, K., Nikles, S., Wache, H., Wolff, D.: An enterprise architecture framework to organize model repositories. In Woitsch, R., Micsik, A., eds.: OKM Open Knowledge Models, Workshop W3 at EKAW 2010. (2010)
8. Kern, H., Hummel, A., Kuehne, S.: Towards a comparative analysis of metametamodels. In: The 11th Workshop on Domain-Specific Modeling, Portland, USA (2011) http://www.dsmforum.org/events/DSM11/Papers/ kern.pdf (last access 05-01-2012).
9. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? IEEE Computer **October 2004** (2004) 64–72
10. Karagiannis, D., Kuehn, H.: Metamodeling platforms. In Bauknecht, K., Min Tjoa, A., Quirchmayr, G., eds.: EC-Web 2002 at Dexa 2002. Springer, Aix-en-Provence, France (2002) 182 Full version available at: http://www.dke.univie.ac.at/mmp/FullVersion_MMP _DexaECWeb2002.pdf.
11. Kuehn, H.: The ADOxx Metamodelling Platform. In: Workshop on Methods as Plug-Ins for Meta-Modelling, Klagenfurt, Austria (2010) http://www.openmodel.at/c/document_library/get_ file?uuid=7516b7c5-a525-4d92-929e-6c11e5da9d39&groupId=10122.
12. Bork, D., Sinz, E.: Design of a SOM Business Process Modelling Tool based on the ADOxx meta-modelling Platform. In De Lara, J., Varro, D., eds.: Proceedings of the Fourth International Workshop on Graph-Based Tools, EASST (2010)
13. Fill, H.G., Gericke, A., Karagiannis, D., Winter, R.: Modellierung fuer Integrated Enterprise Balancing (German: Modeling for Integrated Enterprise Balancing). Wirtschaftsinformatik **06/2007** (2007) 419–429
14. Schwab, M., Karagiannis, D., Bergmayr, A.: i* on ADOxx(R): A Case Study. In: Proceedings of the 4th International i* Workshop - iStar10 - CAiSE Workshop Proceedings, Springer (2010) 92–97
15. Nemetz, M.: A meta-model for intellectual capital reports. In Karagiannis, D., Reimer, U., eds.: Proceedings of the 6th International Conference on Practical Aspects of Knowledge Management, Springer (2006)
16. Hofer, S.: Instances over algorithms: A different approach to business process modeling. In Johannesson, P., Krogstie, J., Opdahl, A., eds.: The Practice of Enterprise Modeling. 4th IFIP WG 8.1 Working Conference, Oslo, Springer (2011)
17. Braun, C., Winter, R.: A comprehensive enterprise architecture metamodel and its implementation using a metamodeling platform. In Desel, J., Frank, U., eds.: Enterprise Modelling and Information Systems Architectures. Gesellschaft fuer Informatik, Bonn (2005) 64–79

18. Demirkan, H., Kauffman, R.J., Vayghan, J.A., Fill, H.G., Karagiannis, D., Maglio, P.: Service-oriented technology and management: Perspectives on research and practice for the coming decade. Electronic Commerce Research and Applications **7**(4) (2008) 356–376
19. Harel, D., Rumpe, B.: Modeling languages: Syntax, semantics and all that stuff - part i: The basic stuff. Technical Report MCS00-16, The Weizmann Institute of Science (August 22, 2000 2000)
20. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Metamodelling - state of the art and research challenges. In Giese, H. et al. , ed.: MBEERTS. Volume LNCS 6100. Springer (2010) 57–76
21. Harmon, P.: The BPTrends 2010 BPM Software Tools Report on BOC's Adonis Version 4.0 (2010) http://www.bptrends.com/publicationfiles/2010%20BPM%20Tools%20Report-BOCph.pdf last accessed 10-10-2011.
22. Junginger, S., Kuehn, H., Strobl, R., Karagiannis, D.: Ein Geschaeftsprozessmanagement-Werkzeug der naechsten Generation - ADONIS: Konzeption und Anwendungen (German: ADONIS: A next generation business process management tool - Concepts and Applications). Wirtschaftsinformatik **42**(5) (2000) 392–401
23. Fill, H.G.: Visualisation for Semantic Information Systems. Gabler (2009)
24. Faisst, U., Buhl, H.: Integrated Enterprise Balancing mit integrierten Ertrags- und Risikodatenbanken (German: Integrated Enterprise Balancing with integrated Return and Risk Databases). Wirtschaftsinformatik **47**(6) (2005) 403–412
25. Gericke, A., Fill, H.G., Karagiannis, D., Winter, R.: Situational Method Engineering for Governance, Risk and Compliance Information Systems. In: DESRIST, ACM (2009)
26. Karagiannis, D., Junginger, S., Strobl, R.: Introduction to Business Process Management Systems Concepts. In Scholz-Reiter, B., Stickel, E., eds.: Business Process Modelling. Springer, Berlin et al. (1996) 81–106
27. Fill, H.G., Redmond, T., Karagiannis, D.: FDMM: A Formalism for Describing ADOxx Meta Models and Models. In Maciaszek, L., Cuzzocrea, A., Cordeiro, J., eds.: Proceedings of ICEIS 2012, Wroclaw, Poland. Volume 3. (2012) 133–144
28. Object Management Group: Omg meta object facility (mof) core specification version 2.4.1 (2011) http://www.omg.org/spec/MOF/2.4.1/PDF/.
29. McNeill, K.: Metamodeling with EMF: Generating concrete, reusable Java snippets (2008) http://www.ibm.com/developerworks/library/os-eclipse-emfmetamodel/index.html?S_TACT=105AG X44&S_CMP=EDU.
30. Jouault, F., Bezivin, J.: KM3: a DSL for Metamodel Specification. In: Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Springer (2006) 171–185
31. OMG, O.M.G.: Unified modeling language (uml) specification: Infrastructure version 2.0. Technical report (2004) http://www.omg.org/docs/ptc/04-10-14.pdf accessed 12-03-2006.
32. Poernomo, I.: The Meta-Object Facility Typed. In: SAC'06, Dijon, France, ACM (2006) 1845–1849
33. Favre, L.M.: Formalization of MOF-Based Metamodels. In Favre, L.M., ed.: Model Driven Architecture for Reverse Engineering Technologies. Information Resources Management Association (2010)
34. OMG, O.M.G.: Object constraint language specification version 2.2. Technical report (2010) http://www.omg.org/spec/OCL/2.2 accessed 11-01-2012.
35. Baader, F.: The description logic handbook: theory, implementation, and applications. Cambridge Univ Pr (2003)