Fill, Hans-Georg, Hickl, Susan, Karagiannis, Dimitris, Oberweis, Andreas, Schoknecht, Andreas (2013)

A Formal Specification of the Horus Modeling Language Using FDMM

Final Version
Accepted for WI 2013, Leipzig

A Formal Specification of the Horus Modeling Language Using FDMM

Hans-Georg Fill[1], Susan Hickl[2], Dimitris Karagiannis[1], Andreas Oberweis[2], and Andreas Schoknecht[2]

[1] University of Vienna, Research Group Knowledge Engineering, Vienna, Austria
{hans-georg.fill,dimitris.karagiannis}@dke.univie.ac.at
[2] Karlsruhe Institute of Technology, Institute of Applied Informatics and Formal Description Methods
{susan.hickl, andreas.oberweis, andreas.schoknecht}@kit.edu

Abstract. In this paper we show how a modeling language from the area of business process engineering can be formally specified using meta modeling concepts. This serves as a basis for the implementation on an industry-scale meta modeling platform. For this purpose we revert to the Horus modeling method and the FDMM formalism that has recently been introduced to formally describe meta models and models. Subsequently we report on the implementation of the modeling language on the ADOxx meta modeling platform and discuss the lessons learned by the application of this approach.

Keywords: Modeling, Meta Modeling, Business Process Engineering.

1 Introduction

For many years the field of business informatics has developed a multitude of modeling methods and tools to support the representation and analysis of complex business-IT relationships [1]. When implementing such modeling methods in the form of modeling tools, meta modeling concepts and platforms today greatly facilitate the implementation of the contained modeling languages [2,3,4,5]. Based on formal specifications of the model, object and data types together with their respective attributes, a modeling language can thus be realized very efficiently with no or little programming effort. In the paper at hand we describe how the modeling language of the Horus modeling method is implemented using such meta modeling concepts [3]. For this purpose we created a formal specification of the Horus modeling language for exactly describing meta models and models [6]. This formal specification is then used for the implementation on an industry-scale meta modeling platform. Subsequently, we discuss our experience of using this approach for the realization of modeling methods.

11th International Conference on Wirtschaftsinformatik,
27th February – 01st March 2013, Leipzig, Germany

The remainder of the paper is structured as follows: In section 2 the foundations for our approach will be outlined. These will encompass a characterization of the Horus modeling method for business process engineering, the FDMM formalism and the ADOxx meta modeling platform. In section 3 it will be shown how the FDMM formalism has been applied to Horus to establish the basis for the implementation on the ADOxx meta modeling platform. The lessons learned from this application will then be discussed in section 4. In section 5 we will discuss related work concerning similar modeling methods and related meta modeling approaches. The paper is concluded with an outlook on the future work in section 6.

2 Foundations

In this section we will briefly describe the foundations used for our approach. This includes at first the foundations of the Horus modeling method, then the FDMM formalism and finally the ADOxx meta modeling platform.

2.1 The Horus Modeling Method

The Horus modeling method for business process engineering comprises steps for an integrated modeling of business processes and for the improvement and further use of the created models [8]. The application of this method always considers a business process in terms of its organizational environment. This is realized by using a set of interrelated models describing different aspects of the business process which are part of the Horus modeling method.

The Horus modeling method classifies the business process engineering into four phases. In phase 0 the engineering project has to be prepared. During the phases 1 and 2 the integrated Horus process model is created. In phase 1 the strategic aspects and the description of the enterprise and system architecture are represented by the model types: *goal model, context model, supply- & services model, SWOT model, strategy model, risk model, key figure model, object model, rule model, business unit model, business process architecture model* and *system architecture model*. Based on this set of models, the business process analysis has to be done in phase 2. Besides improved models of phase 1, the results of this phase are a *resource model*, an *organization model* and a *procedure model*. The models created in phase 2 describe the business process in more detail and from a technical point of view. In phase 3 the use of the integrated process model for implementation and further activities follows.

A special feature of the Horus modeling method is the use of XML nets for process descriptions. XML nets [9] are high-level Petri nets where tokens represent identifiable objects. The places are typed by an XML schema whereby places can be interpreted as containers for XML documents describing relevant process objects. The edges are labeled with filter schemas that can be expressed by an XQuery expression, which describes the relevant process objects for the following transitions. The flow of an XML document is defined by the occurrences of transitions. A transition is activated

if every place in the pre-set of the transition contains a valid XML document, which observes both conditions specified in the filter-schema of the adjacent edge and in the logical expression of the transition. Then the transition may fire and the respective XML documents will be assigned to the post-set of the transition. Using XML nets enables the integrated modeling of structured business objects and object flows as well as process simulation.

2.2 The FDMM Formalism

According to a framework introduced in [3], modeling methods are comprised of a modeling technique and mechanisms and algorithms. The modeling technique can then be further detailed by a modeling language and a modeling procedure that defines the way how to apply the modeling language together with the mechanisms and algorithms to achieve results. In the following we will focus on the aspects of the modeling language and in particular the description of its syntax by using meta models and models.

In order to exactly describe meta models and models independently of their actual implementation, the FDMM formalism has been developed [6]. It can be used to describe meta models and models in a mathematical way. In this way, the resulting formal descriptions can serve as input for the implementation of the meta models and models in an IT environment. In FDMM a meta model MM contains the following parts:

$$\mathrm{MM} = \langle \mathrm{MT}, \preccurlyeq, \mathrm{domain}, \mathrm{range}, \mathrm{card} \rangle \quad (1)$$

Thereby the set MT comprises the set of model types specified for this meta model:

$$\mathrm{MT} = \{\mathrm{MT}_1, \mathrm{MT}_2, \dots, \mathrm{MT}_m\} \quad (2)$$

Each model type MT_i is a tuple of a set of object types O_i^T, a set of data types D_i^T and a set of attributes A_i:

$$\mathrm{MT}_i = \langle \mathrm{O}_i^T, D_i^T, A_i \rangle \quad (3)$$

All object types, data types, and attributes of the model types are parts of the sets O^T, D^T and A whereby object types may also exist independently of model types:

$$O^T = \bigcup_j O_j^T, D^T = \bigcup D_i^T, A = \bigcup A_i \quad (4)$$

$\preccurlyeq$ is an ordering on the set of object types, O^T, i.e. if $o_1^t \preccurlyeq o_2^t$ then we denote object type o_1^t as a 'subtype' of object type o_2^t.

For assigning attributes to object types the domain function maps attributes to the power set of object types:

$$\mathrm{domain} : A \rightarrow \mathcal{P}\left(\bigcup_j O_j^T\right) \quad (5)$$

Similarly, the range function maps attributes to the power set of all pairs of object types and model types, to data types, and to model types. It thus constrains what values an attribute can take in the model instances. Apart from the assignment of data

types, e.g. for strings and integers, this mapping also permits to link to object types of the same or other model types and to model types:

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

The card function constrains how many attribute values an object may have:

$$\text{card: } O^T \times A \to \mathcal{P}\left(\mathbb{N}_0 \times (\mathbb{N}_0 \cup \{\infty\})\right) \tag{7}$$

The sets O^T, D^T, A are pairwise disjoint. For any attribute it is defined that a corresponding domain function must point to an object type of the same model type:

$$a \in A_i \Longrightarrow \text{domain}(a) \subseteq O_i^T \tag{8}$$

The instantiation of a meta model MM is a tuple:

$$\langle \mu_{\text{mt}}, \mu_{\text{O}}, \mu_{\text{D}}, \tau, \beta \rangle \tag{9}$$

where μ_{mt} is a mapping from model types MT to the power set of model instances mt

$$\mu_{\text{mt}}\text{:MT} \to \mathcal{P}(\text{mt}) \tag{10}$$

with the set mt being the union of all mappings of model types to model instances so that every element of the set of model instances mt has to be derived from a model type:

$$\text{mt} = \bigcup \mu_{mt}(\text{MT}_j) \tag{11}$$

The function μ_{O} maps object types of a particular model type to the power set of object instances O:

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

where O is the union of all object instances so that there is no object instance without a mapping to an object type and a model type :

$$O = \bigcup_j \mu_{\text{O}}\left(O_j^T \times \{\text{MT}_j\}\right) \tag{13}$$

An object type $o_t \in O^T$ can be defined as an abstract type. This means that for all model types MT_i which contain the object type o^t ($o^t \in O_i^T$) the object type can only be instantiated through one of its subtypes:

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

The function μ_D maps the data types to the power set of data objects. The data objects themselves are not further defined or constrained. The FDMM formalism thus leaves it to the user to further specify the nature and valid content of a data type:

$$\mu_{\text{D}}\text{: } D^T \to \mathcal{P}(D) \tag{15}$$

For describing the model instances FDMM uses triple statements τ defined as:

$$\tau \subseteq O \times A \times (D \cup O \cup \text{mt}) \tag{16}$$

The function β is then used to map the set of model instances mt to the power set of these triple statements and thus assigns triple statements to the model instances:

$$\beta\text{:mt} \to \mathcal{P}(\tau) \tag{17}$$

In this context, a correctness constraint is defined by FDMM so that τ is the disjoint union of $\beta(\mathrm{mt}_i)$ with $\mathrm{mt}_i \in \mathrm{mt}$, meaning that every triple is contained in exactly one model instance.

Additionally, FDMM defines a number of disjointness and partitioning constraints for the instantiation of meta models. We will briefly describe these in the following in textual form, for the formal definitions we refer to [6]: in FDMM the instances of object types and data types must be disjoint; instantiations from multiple object types, from multiple model types or from multiple data types are not permitted.

For the correct application of the inheritance, domain, range, and cardinality constructs the following constraints must be satisfied [6]: the instantiation of an object type that is a subtype of another object type via the function μ_O is a subset of the instantiation of that other object type; instantiations of sibling object types are disjoint; the value part in the triple statements τ has to match the corresponding range definition for the used attribute and the object type – e.g. if the range for an attribute is to be defined as a 'string', then only a 'string' value can be used in a triple statement for that attribute; for triple statements consisting of an object instance, an attribute, and a value it must hold that the attribute is part of a domain function for the object type used for the instance; for data objects it must hold that their corresponding data type must be defined for the same model type as the attribute; and finally, the cardinality definitions constrain the minimum and maximum number of triple statements for a pair of object types and attributes.

2.3 The ADOxx Meta Modeling Platform

The ADOxx meta modeling platform is currently being used for several projects within the Open Models Initiative[1] and has been successfully deployed in industry for a wide range of business-IT scenarios [7]. In contrast to other meta modeling approaches, the ADOxx platform does not require programming skills for the implementation of meta models and model editors and thus supports a fast and efficient definition of modeling languages.

The ADOxx meta modeling approach builds upon three layers: On the top layer stands the ADOxx meta-meta model that defines the constructs that can be used for the definition of meta models. The basic constructs are classes and relationclasses that can be grouped by using model types. Both classes and relationclasses may contain attributes. On the second layer, meta models can be specified to define the abstract syntax of a modeling language. This abstract syntax is then instantiated on the third and bottom layer to multiple models. For the representation of the concrete syntax, a dynamic graphical notation is assigned to classes and relationclasses. This representation can be automatically adapted during run-time based on the current state of attributes.

[1] Karagiannis, D., Grossmann, W., Hoefferer, P.: Open Model Initiative - A Feasibility Study: http://cms.dke.univie.ac.at/uploads/media/Open_Models_Feasibility_Study_SEPT_2008.pdf

At its core the ADOxx meta modeling platform is a client-server based software application that implements the ADOxx meta modeling approach. It provides a graphical user interface for the definition of classes, relationclasses, and attributes and stores both meta models and models in a relational database. From the specifications in the meta models, the platform automatically generates graphical model editors, provides generic XML import and export formats and handles the persistency of the models and meta models. Based on its client-server architecture it also supports multi-user environments with fine-grained access controls. Furthermore, it offers additional components such as the AdoScript language for implementing algorithms on top of meta models and models, a simulation component for applying process-based simulation algorithms, an analysis component for querying models using the AQL query language and a documentation component for the automatic generation of model documentations in various export formats.

3 Formal Specification of Meta Models for Horus

Based on the foundations presented in the previous section, we can now apply the FDMM formalism to the Horus modeling language. This will provide us with a specification for the implementation on the ADOxx platform. The formal specification presented in the following does, however, not comprise all aspects of the language but only a selected set of constructs. This will illustrate some of the core aspects of applying meta modeling concepts to the Horus modeling language with FDMM. Subsequently we will discuss the implementation on the ADOxx platform.

3.1 FDMM Specification of the Meta Model

As outlined in section 2.1 Horus is composed of a large number of model types that are all tightly interconnected. For illustrating the application of the FDMM formalism we selected four core model types that are used in Horus to describe XML nets. The model types we will discuss in the following are: the procedure model MT_{PM}, the employee pool model MT_{EM}, the role pool model MT_{RM}, and the object model MT_{OM}:

$$\mathrm{MT}_{PM} = \langle O_{PM}^{T}, D_{PM}^{T}, A_{PM} \rangle \tag{18}$$

$$\mathrm{MT}_{EM} = \langle O_{EM}^{T}, D_{EM}^{T}, A_{EM} \rangle \tag{19}$$

$$\mathrm{MT}_{RM} = \langle O_{RM}^{T}, D_{RM}^{T}, A_{RM} \rangle \tag{20}$$

$$\mathrm{MT}_{OM} = \langle O_{OM}^{T}, D_{OM}^{T}, A_{OM} \rangle \tag{21}$$

To illustrate how a model type is detailed by its object types, data types and attributes, we show this for the procedure model type. The object types of the procedure model are defined as:

$$O_{PM}^{T} = \{\text{Abstract-Procedure-Class, Object-store, Activity, Connection, Human-resource-requirements}\} \quad (22)$$

We can then define inheritance relationships between the object types by:

$$\begin{aligned} \text{Object-Store} &\preccurlyeq \text{Abstract-Procedure-Class} \\ \text{Activity} &\preccurlyeq \text{Abstract-Procedure-Class} \end{aligned} \quad (23)$$

Thereby, the object types 'Object-Store' and 'Activity' are defined as subtypes of the object type 'Abstract-Procedure-Class' that is defined as abstract. This makes it easier to assign attributes to the sub types and also simplifies the specification of relationships between all subtypes of an object type.

Next, we define the data types for the procedure model type by:

$$D_{PM}^{T} = \{\textit{String, Integer, Float, File, } \mathbf{Enum_{WF}}, \mathbf{Enum_{AT}}\} \quad (24)$$

As FDMM does not further define the data types that can be used, we are free to use either common types such as *String*, *Integer* or *Float* or custom ones such as *File* that have to be specified during the implementation based on the used implementation platform. By using another set as a data type as it is shown by the $\mathbf{Enum}_{WF}$ and $\mathbf{Enum}_{AT}$ types for example, we can also express data types with pre-defined, fixed values:

$$\mathbf{Enum}_{WF} = \{\text{yes, no}\} \quad (25)$$

$$\mathbf{Enum}_{AT} = \{\text{executing, checking, responsible, informing}\} \quad (26)$$

Subsequently, we can define the attributes necessary for the object types. In the set A_{PM} we only present an excerpt of the attributes that were actually defined for this model type for reasons of brevity. For example, the 'Activity' object type also requires a number of additional attributes for specifying time properties and simulation parameters. The attribute set also comprises elements that will later be used to specify the start- and endpoint of relation-classes, e.g. the connection-from and the connection-to attributes:

$$A_{PM} = \{\text{Name, Object-type, Object-number, Documents, connection-to, connection-from, sub-diagram, HR-req, Role-ref, Assignment-type, Quantity, Percentage, XQuery-transition-condition}\} \quad (27)$$

In the current formalization of the Horus modeling language, the employee pool model and the role pool model only contain a small number of object types, data types and attributes, i.e. for the employee pool model:

$$O_{EM}^{T} = \{\text{Employee}\} \quad (28)$$

$$D_{EM}^{T} = \{\textit{String, File, Calendar}\} \quad (29)$$

$$A_{EM} = \{\text{Name, Availability,Documents}\} \quad (30)$$

And similarly the definitions for the role pool model:

$$O_{RM}^{T} = \{\text{Role, Employees}\} \quad (31)$$

$$D_{RM}^{T} = \{\textit{String, Float, File}\} \quad (32)$$

$$A_{RM} = \{\text{Name, Quality, Assigned-employees, Documents, Employee, Intensity}\} \quad (33)$$

In contrast to these simple model types, the definition of the object model type requires more constructs as it constitutes a way of representing actual XML schemas. The object types of this model type are therefore defined as follows:

$$O_{OM}^{T} = \{\text{Object, Object-copy, Object-Aggregation, Collective-Constraint, Relationship, Inheritance, Constraint-connection, Keys, Attributes, Constraints}\} \quad (34)$$

For the data types of this model type we again use the possibility of referring to predefined sets of attribute values:

$$D_{OM}^{T} = \{\textit{String, Float, File,} \mathbf{Enum}_{DT}, \mathbf{Enum}_{opt}, \mathbf{Enum}_{CT}, \mathbf{Enum}_{CARD}\} \quad (35)$$

$$\mathbf{Enum}_{DT} = \{\text{Unspecified, String, Integer, Float, Date, Enumeration}\} \quad (36)$$

$$\mathbf{Enum}_{opt} = \{\text{Yes, No}\} \quad (37)$$

$$\mathbf{Enum}_{CT} = \{\text{XOR, OR, SIM}\} \quad (38)$$

$$\mathbf{Enum}_{CARD} = \{<0..n>, <1..1>, <1..n>\} \quad (39)$$

The set of attributes for the object model type are then defined as follows:

$$A_{OM} = \{\text{Name, Is-Root, XML-Schema, Key-attributes, Attributes, Constraints, Constraint-type, Relationship-from, Relationship-to, Inheritance-from, Inheritance-to, Constraint-connection-from, Constraint-connection-to, is-inside}\} \quad (40)$$

Finally, we can conclude the formal specification by adding domain, range, and cardinality definitions for the attributes. Again we selected some of the attributes and object types defined above to illustrate this. By assigning an attribute to a super-type, all subtypes automatically inherit the attribute definition as e.g. shown for the Abstract-procedure-class object type and the name attribute:

$$\begin{aligned} \text{domain(Name)} &= \{\text{Abstract-procedure-class}\} \\ \text{range(Name)} &= \{\textit{String}\} \\ \text{card(Name)} &= \langle 1,1 \rangle \end{aligned} \quad (41)$$

For the specification of references between object instances and object instances and model instances it can be chosen from two directions in FDMM: the first is to directly reference another object type or model type and the second is to use an intermediary object type that has references to the object and/or model types that shall be connected. The references are in all cases expressed by attributes whose range contains other object types or model types. At first we illustrate a direct reference to other object types by the example of assigning an object type to an attribute of 'Object-store':

$$\begin{aligned} \text{domain}(\text{Object-type}) &= \{\text{Object-store}\} \\ \text{range}(\text{Object-type}) &= \{\text{Object, Object-Aggregation}\} \\ \text{card}(\text{Object-type}) &= \langle 0,1 \rangle \end{aligned} \quad (42)$$

In this way a core feature of Horus to represent XML nets is specified. The attribute ‘Object-type’ that is attached to the ‘Object-store’ object type can thus be used to reference object instances of the types ‘Object’ and ‘Object-Aggregation’ that are part of the object model type. As this reference points directly to another object type no further information can be assigned to the reference itself. We show in the following how to resolve this.

To enable more complex references where the reference itself can be further specified, an intermediary object type has to be used. We illustrate this in the following for the specification of edges in the procedure model type, which are denoted as ‘Connections’ in the Horus modeling language. For this purpose the two attributes ‘connection-to’ and ‘connection-from’ are assigned to the ‘Connection’ object type with the range definition pointing to the ‘Abstract-procedure-class’. The cardinalities of the from and to attributes are set to <1,1> as edges can only occur with exactly two object instances attached to them:

$$\begin{aligned} \text{domain}(\text{connection-to}) &= \{\text{Connection}\} \\ \text{range}(\text{connection-to}) &= \{\text{Abstract-procedure-class}\} \\ \text{card}(\text{connection-to}) &= \langle 1,1 \rangle \end{aligned} \tag{43}$$

$$\begin{aligned} \text{domain}(\text{connection-from}) &= \{\text{Connection}\} \\ \text{range}(\text{connection-from}) &= \{\text{Abstract-procedure-class}\} \\ \text{card}(\text{connection-from}) &= \langle 1,1 \rangle \end{aligned} \tag{44}$$

When instantiating the object type ‘Connection’ it becomes possible to connect these instances to instances of the type ‘Abstract-procedure-class’ and to treat this relation separately from the objects it connects to. At the same time, further attributes may be assigned to ‘Connection’. This is for example necessary to define transition conditions in XML nets that can be specified via XQuery strings:

$$\begin{aligned} \text{domain}(\text{XQuery-transition-condition}) &= \{\text{Connection}\} \\ \text{range}(\text{XQuery-transition-condition}) &= \{\textit{String}\} \\ \text{card}(\text{XQuery-transition-condition}) &= \langle 1,1 \rangle \end{aligned} \tag{45}$$

When a separate treatment of the relationship between object types is not required but the relation should still be detailed by additional attributes we can express this in FDMM in the following way. The ‘Activity’ objects in Horus have to be detailed by their requirements in terms of human resources. Therefore they are linked to ‘Role’ objects in the role pool model. However, it should be possible to detail for each role, if the role is just executing or checking the activity or is responsible for it or has to inform someone. At first we specify the attribute ‘HR-req’ that can point to any number of ‘Human-resource-requirements’ objects.

$$\begin{aligned} \text{domain}(\text{HR-req}) &= \{\text{Activity}\} \\ \text{range}(\text{HR-req}) &= \{\text{Human-resource-requirements}\} \\ \text{card}(\text{HR-req}) &= \langle 0, \infty \rangle \end{aligned} \tag{46}$$

Then we specify the attribute 'Role-ref' for the object type 'Human-resource-requirements':

$$\begin{aligned} \text{domain}(\text{Role-ref}) &= \{\text{Human-resource-requirements}\} \\ \text{range}(\text{Role-ref}) &= \{\text{Role}\} \\ \text{card}(\text{Role-ref}) &= \langle 1,1 \rangle \end{aligned} \tag{47}$$

As the target reference is now also an object type, we can add further attributes to detail it - for example by reverting to the previously defined $\mathbf{Enum}_{AT}$ data type:

$$\begin{aligned} \text{domain}(\text{Assignment-type}) &= \{\text{Human-resource-requirements}\} \\ \text{range}(\text{Assignment-type}) &= \{\mathbf{Enum}_{AT}\} \\ \text{card}(\text{Assignment-type}) &= \langle 1,1 \rangle \end{aligned} \tag{48}$$

Another important feature in Horus procedure models is the use of refinements for individual activities [8]. Thereby it is controlled how many details should be displayed on each level. In FDMM these refinements can be expressed by references to other model instances of the same type. We show this for the refinement attribute 'sub-diagram' whose range encompasses model instances of the type MT_{PM}:

$$\begin{aligned} \text{domain}(\text{sub-diagram}) &= \{\text{Activity}\} \\ \text{range}(\text{sub-diagram}) &= \{\text{MT}_{PM}\} \\ \text{card}(\text{sub-diagram}) &= \langle 0,1 \rangle \end{aligned} \tag{49}$$

3.2 Examples for Model Instances in FDMM

As FDMM not only permits to formally specify meta models but also the instantiation of meta models, we will illustrate in the following how the definitions of the previous section can be applied. We show the instantiation by using a procedure model and an object model. In Figure 1 on the next page a sample procedure model for an order process and in Figure 2 a sample object model are depicted as they have been later implemented. The instantiation of these model types is specified by:

$$\begin{aligned} \mu_{MT}(\text{MT}_{PM}) &= \{\text{mt}_{pm1}\} \\ \mu_{MT}(\text{MT}_{OM}) &= \{\text{mt}_{om1}\} \end{aligned} \tag{50}$$

In the next step we illustrate the instantiation of object types for these two model types. We show this for the object stores that are represented by circles and the activities that are represented by rectangles in figure 1:

$$\begin{aligned} \mu_O(\text{Object-store}, \text{MT}_{PM}) &= \{\text{os}_1, \text{os}_2, \dots, \text{os}_7\} \\ \mu_O(\text{Activity}, \text{MT}_{PM}) &= \{\text{a}_1, \text{a}_2, \dots, \text{a}_5\} \end{aligned} \tag{51}$$

For the assignment of textual information – as shown in the form of labels for the elements in figure 1 – we first have to instantiate attribute values of the corresponding data types, for example by:

$$\mu_D(\textit{String}) = \{\text{'Website opened', 'Buy products', ...}\} \tag{52}$$

These data objects can then be used in triple statements to assign them to the object instances via their attributes, e.g.:

$$\begin{gathered}(\mathrm{os}_1 \ \text{Name 'Website opened'}) \in \beta\left(\mathrm{mt}_{pm1}\right) \\ (\mathrm{a}_1 \ \text{Name 'Buy products'}) \in \beta\left(\mathrm{mt}_{pm1}\right)\end{gathered} \tag{53}$$

For the specification of the edges in the procedure model we first have to instantiate an object of type 'Connection' in (54) and can then use two triple statements for defining the start and endpoint in (55):

$$\mu_O(\text{Connection}, \mathrm{MT}_{PM}) = \{\mathrm{c}_1, \mathrm{c}_2, \ldots, \mathrm{c}_{11}\} \tag{54}$$

$$\begin{gathered}\left(\mathrm{c}_1 \ \text{connection-from} \ \mathrm{os}_1\right) \in \beta\left(\mathrm{mt}_{pm1}\right) \\ \left(\mathrm{c}_1 \ \text{connection-to} \ \mathrm{a}_1\right) \in \beta\left(\mathrm{mt}_{pm1}\right)\end{gathered} \tag{55}$$

To illustrate how references to other object instances are specified we also take into account the object model shown in Figure 2. For XML nets the object stores in the procedure model are typed by referencing object elements in an object model. Therefore an instance of an object type has to be made available by: $\mu_O(\text{Object}) = \{\mathrm{obj}_1\}$. Then we can reference this object in a triple statement:

$$(\mathrm{os}_1 \ \text{Object-type} \ \mathrm{obj}_1) \tag{56}$$

3.3 Horus on ADOxx: A Prototypical Implementation

For the implementation of the formal specification in ADOxx some differences between the FDMM formalism and ADOxx had to be taken into account [6]. An important characteristic is that in ADOxx it is distinguished between classes and relationclasses that are used to connect classes. In contrast, FDMM does not make such a distinction but only refers to object types. Therefore, it had to be decided for which object types in FDMM classes should be defined and for which object types relationclasses. In addition, also the way how references between object types and object types and model types are used in FDMM differs from ADOxx. ADOxx provides the attribute type 'interref' that is used to specify references from class instances to either other class instances or model instances. However, when such a reference should possess additional attributes, so-called 'record classes' have to be used in ADOxx that permit the combination of several attribute types in the form of a record, i.e. a table-based structure. This directly corresponds to the use of an additional object type in FDMM.

For the realization of the Horus modeling language also additional constraints for the application of constructs on the model level had to be taken into account, apart from cardinality constraints. As FDMM currently does not provide constructs to express such constraints, these had to be added in ADOxx by using the AdoScript language. It is envisaged to add constructs for such constraint specifications to FDMM in the future. This concerned for example the requirement in procedure models that 'connections' can only connect instances of different classes, i.e. that an object store can only be followed by an activity and vice-versa.

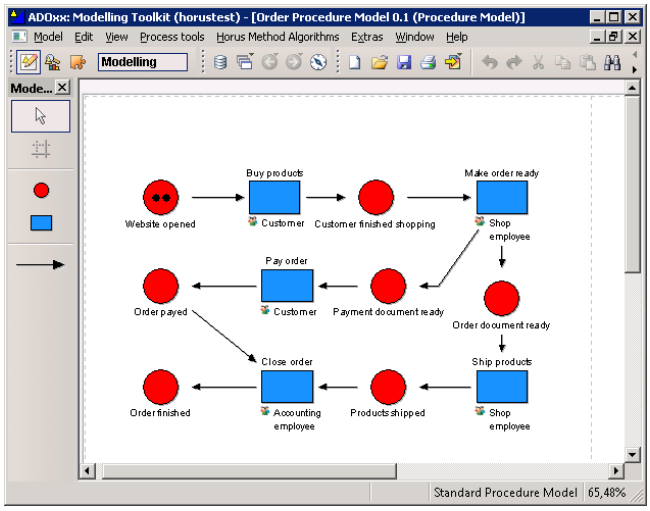

Fig. 1. Example for a Procedure Model in ADOxx

For the implementation of the Horus modeling language on ADOxx it was finally decided to create a total number of 17 model types that contain 44 classes, 9 relation-classes, and 11 record classes. From these classes 18 were defined as abstract classes. In addition, 161 attributes were created for the classes and 26 attributes for the relationclasses. Furthermore, for each non-abstract class and each relationclass a graphical representation was specified in the proprietary ADOxx-GraphRep grammar.

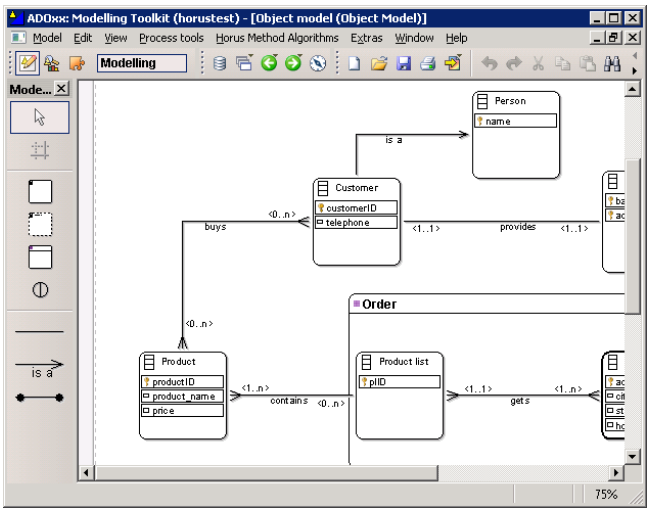

Fig. 2. Example for a Horus Object Model in ADOxx

All these tasks could be done using the graphical user interface of ADOxx for defining meta models and thus did not involve any programming effort. Overall, the implementation required about 60 person hours, not including testing, which is still being conducted at the time of writing this paper. The implementation will be made freely available in the context of the Open Models Initiative[2].

4 Lessons learned

As modeling methods constitute an important means in the domain of business informatics, their exact specification is an important step to ensure a common understanding of the contained concepts between the designers of a modeling language and the developers of the modeling tool. In particular, the implementation of a modeling language on a meta modeling platform greatly benefits from a meta modeling oriented formal specification as it can be determined exactly how the language needs to be implemented. In addition, comparisons between the actual implementation and the specification can be made to ensure the quality of the implementation.

With the formal specification of the Horus modeling language by using the FDMM formalism, we could show how such a specification can be done in practice. Thereby, the FDMM formalism was directly applicable to the concepts of the Horus modeling language. However, the specification using FDMM also has some limitations. For example, in its current form it is not possible to formally specify the modeling procedure and the mechanisms and algorithms of a modeling method, i.e. the dynamic aspects. Also, the graphical representation of the elements and relations that constitute an indispensable part of many modeling methods in business informatics is currently not supported by FDMM. In regard to the implementation on the ADOxx platform, the differences between FDMM and the constructs in ADOxx had to be observed, e.g. the absence of explicit constructs for relations in FDMM that are available in ADOxx. Therefore, the use of FDMM specifications for the later implementation on ADOxx requires good knowledge of the platform functionalities. This also concerns the decisions on how to graphically represent the modeling language so that it is both user friendly and at the same time encompasses all required information for effectively creating models.

For the development of modeling tools, a formal description of the underlying meta model leads in particular to two main advantages: firstly, it has to be made explicit in an unambiguous representation, which concepts of a modeling language, including its elements, their relationships and the according attributes are provided to the users of a modeling language. Secondly, these descriptions can directly support the implementation of a modeling tool by providing a reference for the requirements of the implementation that can be exactly verified against the actual software implementation.

[2] http://www.openmodels.at/web/adoxx-horus-method/download

As a result of the application of FDMM to the Horus modeling method, we can derive the following requirements for a further extension of FDMM. To close the gap between the formalization in FDMM and the implementation on a meta modeling platform, a formal mapping between FDMM and the programming constructs of meta modeling platforms should be established. This mapping can then also be realized as a software component for the automatic generation of meta modeling platform specific code from the formal specifications. Furthermore, the representation of dynamic aspects of modeling methods, such as the modeling procedure and mechanisms and algorithms that take into account the behavior of the modeling method, should be added to FDMM. Also these extensions can then be incorporated in the platform specific mapping to eliminate the step of manually translating the specifications into concrete code.

5 Related Work

In conclusion, two relevant areas can be distinguished to which the paper at hand can be related. First, the approach is compared to approaches in software development. Second, ADOxx and FDMM are compared to other meta modeling approaches.

The formal specification of meta models and models can be compared to formal specifications of software systems, e.g. by using graph transformations [15,16]. Similarly to our approach formal specifications in this domain are also used for making the consistency of the dimensions of a system explicit.

The FDMM formalism and ADOxx can be compared to other meta modeling approaches and formalizations of meta models and models. According to [14] FDMM and ADOxx can be directly compared to domain-specific modeling approaches which view meta models as *language specifications*. In this context, meta-meta models enable the automatic creation of graphical model editors from modeling language specifications. Kern et al. [4] provide an overview of such meta-meta models, which include for instance the GOPPRR/MetaEdit+ or the ARIS framework. Another group of meta modeling approaches uses meta models as *software structure specifications* (e.g. EMOF [13] or EMF [12]) and their formalisms consequently concentrate on the specification of software structures (see e.g. [11] for EMOF). In comparison, ADOxx does not focus on the generation of source code as e.g. intended by MetaEdit+ and neither requires programming skills. In comparison to the approach in [16], FDMM currently does not permit to express the dynamic behavior of modeling methods but rather focuses on the formalization of the abstract and the concrete syntax.

6 Outlook

Based on the insights gained by the application of FDMM to Horus and the implementation on ADOxx we will continue to work on facilitating the implementation of modeling methods. Possible next steps therefore concern the combination of the

FDMM formalism with existing techniques for the automatic assignment of graphical representations [10] as well as the specification of model algorithms in a similar, non-programming oriented way. This will also be of further benefit for the implementation of the Horus method in order to specify additional graphical representations, e.g. to use domain-specific visualizations of Petri-nets, as well as for the realization of simulation functionalities for selected sets of model types.

References

1. Koch, S., Strecker, S., Frank, U.: Conceptual Modelling as a New Entry in the Bazaar: The Open Model Approach. Open Source Systems, vol. 203/2006, pp. 9-20. IFIP (2006)
2. Karagiannis, D., Fill, H.-G., Hoefferer, P., Nemetz, M.: Metamodeling: Some Application Areas in Information Systems. In: Kaschek, R., et al. (eds.) UNISCON, pp. 175-188. Springer (2008)
3. Karagiannis, D., Kuehn, H.: Metamodeling Platforms. In: Bauknecht, K., Min Tjoa, A., Quirchmayr, G. (eds.) EC-Web 2002 – Dexa 2002, pp. 182. Springer (2002)
4. Kern, H., Hummel, A., Kuehne, S.: Towards a Comparative Analysis of Meta-Metamodels. The 11th Workshop on Domain-Specific Modeling, Portland, USA (2011)
5. Gulden, J. and Frank, U.: MEMOCenterNG - A full-featured modeling environment for organization modeling and model-driven software development, in: Proper, E., Soffer, P.: Proceedings of the CAiSE Forum 2010, CAiSE, Hammamet (2010)
6. Fill, H.-G., Redmond, T., Karagiannis, D.: FDMM: A Formalism for Describing ADOxx Meta Models and Models. In: Maciaszek, L., et al. (eds.) ICEIS 2012, Wroclaw, Poland, vol. 3, pp. 133-144 (2012)
7. BPTrends: The 2005 EA, Process Modeling and Simulation Tools Report - Adonis Version 3.81. Business Process Trends, (2005)
8. Schönthaler, F., Vossen, G., Oberweis, A., Karle, T.: Business Processes for Business Communities - Modeling Languages, Methods, Tools. Springer (2012)
9. K. Lenz and A. Oberweis: Inter-organizational Business Process Management with XML Nets. In: Petri Net Technology for Communication-Based Systems, Advances in Petri Nets, LNCS 2472, pp. 243-263. Springer (2003)
10. Fill, H.-G.: Visualisation for Semantic Information Systems. Gabler (2009)
11. Favre, L.: A Formal Foundation for Metamodeling. In: Kordon, F. and Kermarrec, Y. (eds.) Reliable Software Technologies – Ada-Europe 2009. pp. 177-191. Springer (2009)
12. McNeill, K.: Metamodeling with EMF: Generating concrete, reusable Java snippets, http://www.ibm.com/developerworks/library/os-eclipse-emfmetamodel/index.html (2008).
13. Object Management Group: Omg meta object facility (mof) core specification version 2.4.1, http://www.omg.org/spec/MOF/2.4.1/PDF/ (2011)
14. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Metamodelling: state of the art and research challenges. Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems. pp. 57-76. Springer (2010)
15. Engels, G., Heckel, R.: Graph Transformation as a Conceptual and Formal Framework for System Modeling and Model Evolution. In: Montanari, U. et al. (ed.) ICALP 2000, Springer (2000)
16. Engels, G., Soltenborn, C., Wehrheim, H.: Analysis of UML Activities Using Dynamic Meta Modeling. Proceedings of FMOODS 2007, Springer, 76-90 (2007).