Experiences from the Implementation of a Structured-Entity-Relationship Modeling Method in a Student Project

Thilo Maximilian Glässner, Florian Heumann, Luca Keßler, Felix Härer, Andreas Steffan, Hans-Georg Fill[1]

Information Systems - System Development and Database Application Group
An der Weberei 5, 96049 Bamberg, Germany
`seda-info.wiai@uni-bamberg.de`,
WWW home page: `https://www.uni-bamberg.de/seda`

Abstract. For attaining the vision of a wide-spread use of enterprise modeling in everyday business practices, the underlying modeling methods need to be easily accessible. Therefore, open implementations of these modeling methods are required that can be modified and extended as needed. At the same time, such open implementations must provide a sufficiently advanced level to be adequate for practical usage. In this paper, we report on experiences gained from the implementation of the SERM modeling approach on the ADOxx meta modeling platform in a student project. We derive guidelines for teachers and students who want to engage in similar projects and discuss implications for future research.

Key words: data modeling, meta modeling, modeling tool, SQL

1 Introduction

One of the foundations for establishing enterprise modeling as a common practice in organizations is the availability of modeling tools that permit the effective use of the underlying modeling methods and the exchange of model data between different parties and systems [1, 2, 3]. Conceptual enterprise modeling methods are used today for many purposes and domains, ranging from the strategic layer, over the layer of business processes and workflows to the technical and infrastructure layer [4, 5]. An important domain in enterprise modeling is the modeling of data structures and their implementation in database systems [6, 7]. Well-known examples of modeling methods belonging to this domain are the Entity-Relationship-Model by Chen [8] and the Unified Modeling Language (UML). However, also several modeling methods have been conceptualized that target specific ways of modeling and using data structures, e.g. the method for adaptable database design by Roussopoulos and Yeh [9] or the higher order entity-relationship modeling (HERM) language by Thalheim [10].

A further approach that is part of the domain of data modeling is the Structured-Entity-Relationship-Model (SERM) by Sinz [11]. SERM is specifically directed towards the structuring of large data models, the visualization of existence dependencies, the avoidance of inconsistencies and the easy transition to database schemata. For the details of the SERM method we refer to [11, 12].

In the past, several modeling tools have been developed for SERM. However, these tools were either developed based on some legacy systems that are not operational in modern IT environments any more. Or, some central aspects of SERM such as its particular suitability for generating database schemata had not been realized. Therefore, it was decided to realize a new and openly accessible version of a SERM tool in a student project by reverting to the ADOxx meta modeling platform [2]. In this way, the resulting tool should be shared via the Open Models Initiative to provide a foundation for further developments based on SERM [13].

In this paper we will report on the experiences gained from this project, both from the perspective of the students as well as the teachers in the course. The goal is to provide recommendations for similar projects, thereby contributing to the realization of open enterprise modeling. The remainder of the paper is structured as follows. In Section 2 we will describe the set-up of the student project and the positioning within the curriculum. This will be followed by a description of the project phases in Section 3. Finally, we will report on the experiences in Section 4. The paper will conclude with an outlook on further research in Section 5.

2 Project Set-Up and Positioning in the Curriculum

The project was set up in the course of a master seminar for students of Information Systems at the University of Bamberg. Although the students were familiar with various conceptual modeling methods including SERM as well as with the implementation of software applications, none of them had previously implemented a modeling tool. They neither had any knowledge about the ADOxx meta modeling platform nor about the transformation of models into code, e.g. by using XML and XQuery. The goal that was set for the seminar was to implement a SERM modeling tool that should provide as much aid to the users as possible in creating valid SERM models. In addition, the transformation of the models into SQL schemata that are executable on the PostgreSQL database server was defined as a target.

3 Project Phases

The project was structured in the following phases: preliminary phase, foundations of modeling methods conceptualizations, formalization using FDMM, and iterative design and implementation using ADOxx and XML processing. These phases will be described in detail in the following sub sections.

3.1 Preliminary Phase

In the preliminary phase the seminar participants were introduced to the goals of the project and the main properties of SERM were presented as a re-cap from previous courses. In addition, the current knowledge level of the participants in regard to the implementation of modeling methods in general and the familiarity with ADOxx were assessed. This also included the provision of information regarding the availability of corresponding literature sources, the ADOxx platform, and support tools that are helpful in this context, e.g. the GraphRep generator or the syntax highlighting definitions for ADOscript in the Notepad++ editor. With these tools, the students were walked through the design of a minimal language. This included a meta model and a visual language consisting of simple modeling elements similar to a "Hello World" program in software development.

3.2 Foundations of Modeling Method Conceptualization

The theoretical underpinning of the project is based on a framework proposed by Karagiannis and Kühn [14]. It serves as a basis to establish a common understanding regarding the components of modeling methods, especially in the context of developing modeling tools with a meta modeling platform such as ADOxx. According to Karagiannis and Kühn, a modeling method is comprised of modelling technique, including a modeling language and a modeling procedure, as well as mechanisms and algorithms. The modeling language is defined through its syntax, semantics and notation. For this project, these components were of particular interest for the implementation.

When conceptualizing a tool based modeling method, the implementation of the components defined by Karagiannis and Kühn needs to be aligned with three facets. Central to the conceptualization is facet a., the *Domain Concept and Grammar* of the modeling method to define core syntax, semantics and notation of the language. Facet b. is the user's *Handling and Interaction* of the Domain Concept and Grammar, e.g. through the choice of available notation elements and their graphical representation, optimized for usability. Facet c. concerns the modeling platform's *Management and Storage* of the Domain Concept and Grammar in data structures and algorithms, optimized for processing inside the platform. Facets b. and c. conflict each other and need to be balanced. For example, in a language with directed and undirected edges, it might be algorithmically easier to process one notational element which has an attribute to differentiate directed and undirected edges. For a user, however, two separate notational elements available next to each other in a toolbar might be preferable. During implementation, any technical decision has to consider b. as well as c..

The ADOxx meta modeling platform allows designing the abstract syntax of a modeling language through the construction of a meta model as well as an according concrete syntax for the visual notation. The ADOxx meta model [2] complies with the approach of Karagiannis and Kühn by providing model types with Class and Relation Class elements at its core to allow for instantiating models with model elements and relations. Class and Relation Class may contain multiple instances of Attribute, which may be user defined or, in case of the notation, pre-defined as a GraphRep attribute. Through GraphRep, the notation of Class and Relation Class elements is defined.

3.3 Formalization Using FDMM

At this point in the project the student team had a thorough understanding of the SERM modeling method. The team also had a basic understanding of the necessary technical steps for the conceptualization of a meta model on the ADOxx platform (e.g. definition of model types, classes, relation classes and their graphical representation). While SERM meta models have been published in the past, e.g. [11], [15, p. 171], they are incomplete (e.g. missing SERM's generalization constructs), have author-specific notations and are not tailored towards the development of modeling tools. Therefore, the conceptualization of the SERM meta model in the ADOxx platform is not straightforward. However, the conceptualization and formal description is of critical importance for all upcoming development tasks [16].

Instead of proceeding directly with the implementation of the SERM modeling method on the ADOxx platform, it was therefore decided to conduct an additional formalization phase using the FDMM formalism [17]. FDMM allows for a formal, yet concise, technology-independent specification of the meta model to be implemented.

As this phase's first step, a 45 minute introductory lecture on FDMM was given. The lecture, including an exemplary application of FDMM on the 4R modeling language, was based on [17]. It has to be noted that the participating research assistants did have prior experience with the ADOxx platform, but neither the student team, nor the research assistants had any prior experience with FDMM.

Subsequent to the lecture, a discussion about possible design choices regarding the SERM meta model formalization followed. An emphasis has been on when to use attributes and when to use modeling classes and relations of their own for SERM elements (e.g. individual classes for (0,1)-, (0,*)- and (1,*)-relations vs. one class with an attribute describing its complexity). Another point which emerged during the discussion was that previously published SERM meta models are incomplete, especially regarding SERM's generalization. As a result, the students has been aware of the upcoming design choices and to which extent existing literature could be leveraged.

Next, the student team presented a first draft of their SERM formalization using FDMM. The SERM meta model was formally described as follows. As a starting point, single model type MT_{SERM} is defined:

$$MT_{SERM} = \langle O^T_{SERM}, D^T_{SERM}, A_{SERM} \rangle \tag{1}$$

Then, the set of object types O^T_{SERM} was described, which corresponds to classes and relationclasses in ADOxx:

$$\begin{aligned} O^T_{SERM} = \{&DOT, EType, ERType, RType, DOTRelation,\\ &Triangle, DotToTriangle, TriangleToER\\ &Record_{Attributes}, Record_{Inherited\_Attributes}\} \end{aligned} \tag{2}$$

The type DOT (data object type) acts as a super type for $EType$, $ERType$ and $RType$:

$$\begin{aligned} EType &\preceq DOT\\ ERType &\preceq DOT\\ RType &\preceq DOT \end{aligned} \tag{3}$$

The types $Triangle$, $TriangleToER$ and $DotToTriangle$ were introduced for SERM's generalization mechanism. To represent ADOxx's record type attributes, i.e. table-valued attributes, the two object types $Record_{Attributes}$ and $Record_{Inherited\_Attributes}$ were introduced.

Relations between data object types were realized using a single type $DOTRelation$, while the relation's cardinality was represented by an attribute. The SERM meta model from [15, p. 171] uses individual relation types for (0,1)-, (0,*)-, (1,1)- and (1,*)-relations, which are subtypes of a generic relation type. However, as ADOxx does not support subtyping for relation classes, the student team avoided using subtyping for the $DOTRelation$ type. In doing so, the formalization process resulted in a meta model with less model types, making it simpler to implement in ADOxx later. Hence, in order to make an FDMM formalization useful for the upcoming development phases, it was important to establish a basic understanding of the functionality of the target meta modeling platform among all developers before using FDMM. Thereby it can be made sure that the resulting formalization can later be mapped to the target meta modeling platform.

Due to space limitations, only an excerpt of the definition of attributes is described in the following. But before that, the set of data types D^T_{SERM} and the set of attributes A_{SERM} needed to be defined. An excerpt of the set of data types is shown in the following:

$$\begin{aligned} D^T_{SERM} &= \{Integer, String, Enum_{SQLDatatype}, \dots\}\\ Enum_{SQLDatatype} &= \{smallint, int, bigint, date, \dots\} \end{aligned} \tag{4}$$

Enum data types, such as $Enum_{SQLDatatype}$, are used to represent ADOxx Enumeration types with predefined values. The set of attributes A_{SERM} contains the attributes of all types. Therefore, only an excerpt can be shown here:

$$A_{SERM} = \{Name, Attributes, Key, Cardinality, \dots\} \tag{5}$$

The attributes from A_{SERM}, with the exception of table-valued attributes, were then attached to object types and given a value range. Using FDMM's $card$ function, the number of attribute values per object was constrained. For example, each data object type has exactly one name:

$$\begin{aligned} domain(Name) &= \{DOT\} \\ range(Name) &= \{String\} \\ card(DOT, Name) &= \langle 1, 1 \rangle \end{aligned} \tag{6}$$

Finally, ADOxx relation classes were formalized in FDMM using "from" and "to" attributes, e.g.:

$$\begin{aligned} domain(relatesFrom) &= \{DOTRelation\} \\ range(relatesFrom) &= \{DOT\} \\ card(DOTRelation, relatesFrom) &= \langle 1, 1 \rangle \\ \\ domain(relatesTo) &= \{DOTRelation\} \\ range(relatesTo) &= \{DOT\} \\ card(DOTRelation, relatesTo) &= \langle 1, 1 \rangle \end{aligned} \tag{7}$$

3.4 Iterative Design and Implementation

The abstract and concrete syntax of SERM was implemented in ADOxx over the course of about ten weeks. Included in this time frame are teaching sessions as well as result discussions, which were alternating according to the syllabus. The iterative process covered the implementation of the abstract syntax by extending the ADOxx meta model with the already formalized SERM meta model and the implementation of the concrete syntax by specifying a graphical representation. Beyond syntax and notation, algorithms for the inheritance of primary key attributes among data object types and the generation of SQL code were implemented using ADOxx expression attributes in conjunction with a customization programmed in the ADOscript programming language.

Implementation of Abstract Syntax The elements of the abstract syntax of SERM were implemented as extensions of Class and Relation Class of the ADOxx meta model. Class was extended by the abstract Data Object Type (DOT) which was in turn extended with Entity Type (E-Type), Entity-Relationship Type (ER-Type), Relationship Type (R-Type), and Triangle for generalizations.

In order to allow relations between instances of Data Objects Types, a Relation Class DOT-Relation having a From-Class DOT and a To-Class DOT was specified, so that such a relation always connects two DOT instances. Cardinalities between DOT instances were set as attribute values in the Cardinality attribute of a DOT-Relation instance as an Enum value out of (0,1), (1,1), (0,*), and (1,*). To represent key inheritance between two instances of Data Object Types, DOT-Relation contains a Key attribute of type Enum with the predefined values PK, FK and No Key, indicating that the primary key of a DOT instance is inherited to another DOT instance as part of the primary key, foreign key or as a non-key attribute respectively.

In contrast to relations between DOT instances, a generalization connects one instance of DOT with one or more instances of specialized ER-Type. A

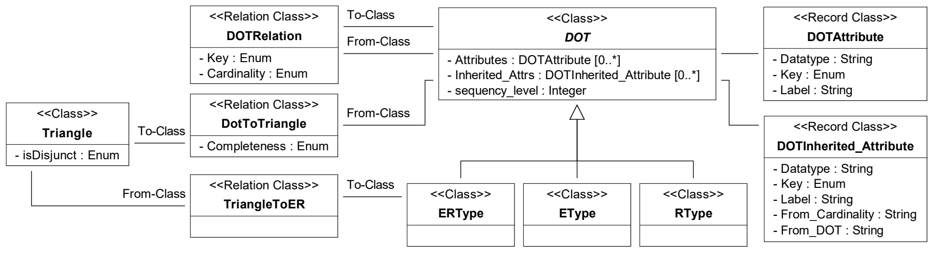

Fig. 1. Meta Model for ADOxx Model Type SERM

single Relation Class is therefore not sufficient, but rather two Relation Class DotToTriangle and TriangleToER as well a Class Triangle. Due to notation, completeness of generalized subtypes is stored in an Enum attribute in DotToTriangle, whereas disjunctive generalizations are indicated by an attribute of Triangle.

SERM allows the specification of attributes when modeling E-, ER-, and R-Types. For the representation of these instance-level attributes in a model, the E-, ER- and R-Type Class feature an attribute Attributes in conjunction with a Record Class DOTAttribute. The Record Class defines that such an instance-level attribute has a label, a datatype from an enumeration of commonly used SQL data types, as well as key to differentiate between primary key (PK), foreign key (FK), PK and FK, and non-key attributes. In order to allow such attributes to be edited by the modeler, they were added to a user accessible Notebook using the AttrRep attribute of E-, ER-, and R-Type. Inherited instance-level attributes were stored in a similar fashion. Additionally, the Sequency Level attribute was provided to store an ordering property of the quasi-hierarchical graph a SERM constitutes. It is used for ADOscript implementations of algorithms for key inheritance as well as to prepare the generation of SQL code through an XML model.

Implementation of Concrete Syntax The graphical representation of a Class or Relation Class was defined by their GraphRep attribute containing code according to the GraphRep grammar. SERM depicts E-Types and R-Types in the fashion of ERM. The notation of ER-Types reflects the combination of the E- and R-Type, given a 1:1-relation between them. In addition to the respective shapes, the GraphRep of these Classes contains scaling logic using variable-sized tables to allow resizing. SERM itself does not define a notation for attributes; a line-by-line attribute list below each data object type was chosen for size-efficiency. For displaying attributes, the GraphRep implementation contains logic to extend the actual shape, read instance-level attribute values at run-time and write them out line-by-line using token lists.

E-, ER-, and R-Types may be connected using edges for cardinality (0,1), (1,1), (0,*), or (1,*). According to SERM notation, the lower bound is indicated by a single (0) or double (1) line, whereas the upper bound is indicated by arrow (*) or no arrow (1). To make changes of cardinality possible after an edge has

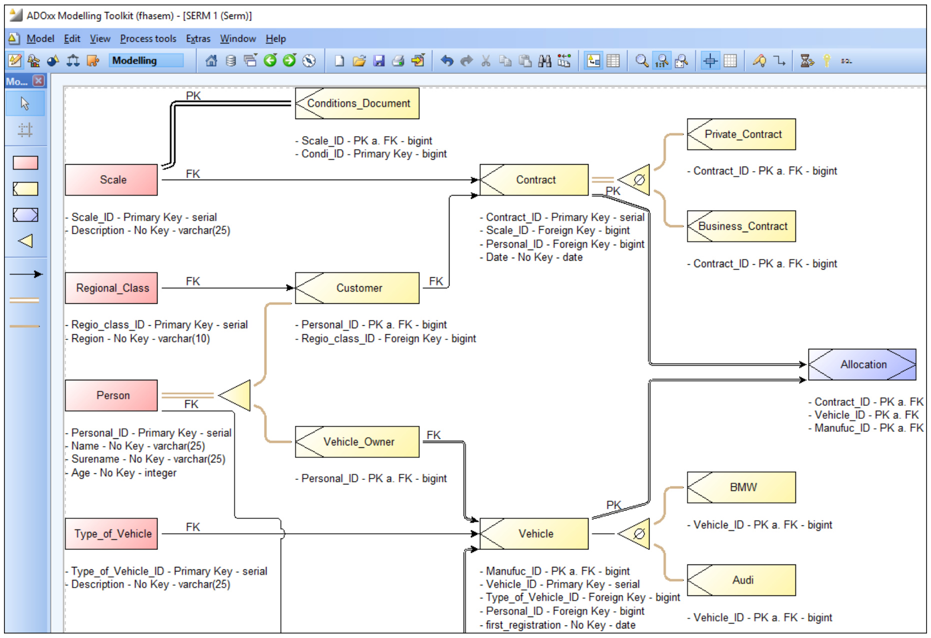
ADOxx Modelling Toolkit (fhasem) - [SERM 1 (Serm)]
Model Edit View Process tools Extras Window Help
Modelling
Mo...
Conditions_Document
- Scale_ID - PK a. FK - bigint
- Condi_ID - Primary Key - bigint
PK
Scale
FK
- Scale_ID - Primary Key - serial
- Description - No Key - varchar(25)
Contract
- Contract_ID - Primary Key - serial
- Scale_ID - Foreign Key - bigint
- Personal_ID - Foreign Key - bigint
- Date - No Key - date
PK
Private_Contract
- Contract_ID - PK a. FK - bigint
Business_Contract
- Contract_ID - PK a. FK - bigint
Regional_Class
FK
- Regio_class_ID - Primary Key - serial
- Region - No Key - varchar(10)
Customer
FK
- Personal_ID - PK a. FK - bigint
- Regio_class_ID - Foreign Key - bigint
Allocation
- Contract_ID - PK a. FK
- Vehicle_ID - PK a. FK
- Manufuc_ID - PK a. FK
Person
FK
- Personal_ID - Primary Key - serial
- Name - No Key - varchar(25)
- Surename - No Key - varchar(25)
- Age - No Key - integer
Vehicle_Owner
FK
- Personal_ID - PK a. FK - bigint
BMW
- Vehicle_ID - PK a. FK - bigint
PK
Type_of_Vehicle
FK
- Type_of_Vehicle_ID - Primary Key - serial
- Description - No Key - varchar(25)
Vehicle
- Manufuc_ID - PK a. FK - bigint
- Vehicle_ID - Primary Key - serial
- Type_of_Vehicle_ID - Foreign Key - bigint
- Personal_ID - Foreign Key - bigint
- first_registration - No Key - date
Audi
- Vehicle_ID - PK a. FK - bigint

Fig. 2. ADOxx SERM Modeling Tool with Sample Model

been placed, the cardinality is an attribute of DOT-Relation. A consequence of this decision was that the modeler is presented with only edge to choose from. The GraphRep logic contains the required line and arrow notation as well a display for key inheritance.

Realizing the graphical notation of generalizations required separate GraphRep logic for DotToTriangle, TriangleToER and Triangle. The completeness property in the DotToTriangle GraphRep displays single or double lines from the supertype to the triangle, indicating given or not given completeness respectively. Disjunct subtypes are indicated in the Triangle GraphRep using the empty set symbol. A limitation of this implementation is the visual depiction of three separate symbols for generalization model elements in the toolbar next to an ADOxx SERM instance. From a usability perspective this might still be practical since all elements fulfill different functions and are subsequently needed when modelling a generalization.

Implementation of the Transformation to SQL With the features for creating SERM model being ready, the next step was the development of the SQL code generation feature. The ADOxx platform offered two alternative ways for implementing a transformation from SERM to SQL code. The first alternative was to implement the transformation only using ADOxx's scripting facility based on the ADOscript language. The second alternative was to use the ADOxx XML export functionality to export a SERM model as an XML file, which can be further processed using standard XML processing languages and tools.

As a pure ADOscript based solution was expected to become hard to debug and maintain quickly, the team decided to use the XML based approach for the transformation. It should be noted, that the student team had no prior in-depth experience in XML processing. Therefore, some introductory learning material on XML – covering topics such as XML basics, XSLT and XQuery – was provided to the students. As the students were familiar with SQL, they chose XQuery for the processing of the exported XML files, as some of the language's elements have more resemblance to SQL when compared to XSLT.

Evaluation Using a Test Model The syntax of SERM required the implementation of three data object type elements, four edge elements and four generalization elements [15, p. 160-166]. To evaluate the implementation, a reference model making use of at least one instance per syntax element was designed as depicted in Figure 2. Table 1 shows the number of instances per element in the model. Due to $n > 0$ for each element, completeness of the SERM syntax is demonstrated.

Data Object Type	n
E-Type	5
ER-Type	12
R-Type	1

Edge	n
(0,1)-relation	1
(1,1)-relation	1
(0,*)-relation	5
(1,*)-relation	4

Generalization	n
Non-Disjunctive and Incomplete	1
Disjunctive and Incomplete	1
Non-Disjunctive and Complete	1
Disjunctive and Complete	1

Table 1. Number of Instances per SERM Syntax Element

4 Experiences gained from the Project

After the description of the project phases we will now summarize our experiences. This will be discussed both from the perspective of the involved students (Thilo Maximilian Glässner, Florian Heumann, and Luca Keßler) as well as from the perspective of the teachers (Felix Härer, Andreas Steffan, Hans-Georg Fill).

4.1 Experiences from Students

One of the major advantages from the side of the students was the familiarity with the SERM modeling method from previous courses. This significantly lowered the barrier for designing and implementing an according modeling tool. On the other hand, ADOxx was not known to the students. As ADOxx offers a large number of functions it was first necessary to become acquainted with the platform. This had to be primarily done by the students themselves as no courses on ADOxx are currently offered at the University of Bamberg. In addition, neither the processing of XML had been taught in courses before. Therefore, the students also had to become familiar with the corresponding technologies, in particular the XQuery language.

The learning curve in regard to the ADOscript language and the expression language used for programming with ADOxx was rather steep. It could not be accomplished by the students on their own without support from the teachers. Especially, it was unclear which functions in ADOxx have to be used for achieving certain results and what the different options and trade-offs are. This was despite the fact that the syntax of the used languages was rather easy to comprehend, although it is not based on object oriented principles which the students were already familiar with.

The scheduling of weekly meetings with the teachers was very helpful for receiving support during the development. One aspect that proved difficult was the distributed cooperation of the three students on the ADOxx library as the ADOxx version used in the course did not permit multi-user distributed development per se. It was therefore decided by the students to resolve this issue by using a virtual machine hosted on Amazon Web Services where all students had simultaneous access. This also permitted to use different development platforms on the side of the students, i.e. Mac, Linux, and Windows systems.

4.2 Experiences from Teachers

From the side of the teachers it was a unique opportunity to have a very small number of students in the course. This permitted a very close interaction with weekly meetings and immediate feedback on the conceptualization, the formalization, and the technical implementation of the modeling method. For larger groups - which had originally been expected - it would not be possible to have the same level of interaction.

Concerning the material required for the course it was possible to re-use material provided by the adoxx.org and the omilab.org websites. Thereby, it proved particularly beneficial that a large number of existing modeling tools together with their openly available libraries in ADOxx could be used for showcasing different scenarios and the usage of ADOxx functions. Especially, for elaborating best practices on the specification of the visual notation in the GraphRep grammar (e.g. the rather complex ways of correctly specifying scalable graphical objects) or the specification of expression attributes in the LEO grammar and ADOscript algorithms it was helpful to revert to existing resources and examples.

Due to the familiarity of the students with the SERM modeling method from previous courses, it was easy to explain to them the goals of the project. In addition, the motivation of the students to implement their own modeling tool including an automated SQL transformation from SERM models seemed very high throughout the project.

In case the modeling method is not known to the students or, if a modeling method does not yet exist and needs to be conceptualized from the beginning, it may be more difficult to convince and motivate the students for such a project. The challenge of the project described here lied mainly in the technical realization and the automated code generation. Designing a modeling method from scratch, including according abstraction could prove more challenging. This is planned to be explored in future student projects.

4.3 Recommendations for Future Projects

In summary, we can give the following recommendations for meta modeling projects that take a similar direction. During the definition of the *project's goals* it needs to be considered what can be expected of the intended participants of the project. For example, if the focus of the project is to *re-implement* an already existing modeling method - i.e. one that has either been implemented as a modeling tool before or that is specified in such great detail that the meta model can be immediately implemented without further changes - then the participants only need to acquire additional know-how about the implementation environment. If, however, the modeling method has not yet been implemented or not in the way intended for the project and the participants thus have to take decisions on the core parts of the meta model, also a high familiarity with *abstraction* and *design* concepts is necessary.

In any case, *regular interactions* between the project's participants and the teachers / experts in meta modeling should be planned. This ensures continuous and early feedback as practical meta modeling projects tend to be very complex. This primarily stems from the many inter-linkages between the domain representation, the processing and storage aspects, and the user interface / user interaction design. One way to cope with this complexity is to provide examples from other implemented modeling methods to *illustrate best practices*. For this purpose, repositories such as omilab.org or adoxx.org can be consulted in the case of ADOxx.

Finally, it needs to be ensured that the participants find a way to effectively *collaborate* on the project. For example, if possible, resources for sharing documents, libraries or even virtual machines should be foreseen.

5 Conclusion and Next Steps

In this paper we have reported on the experiences from a meta modeling project in a master seminar. Besides a detailed description of the phases and results of the project we summarized the experiences gained both from the students and the teachers.

The next steps will be to tackle a more complex project setting and thus enlarge the project focus. We currently plan to redo the meta modeling project for a modeling method that has not been implemented before and where the students thus need to take decisions on the adequate design of the meta model and the visual notation.

References

1. Sandkuhl, K., Fill, H.G., Hoppenbrouwers, S., Krogstie, J., Leue, A., Matthes, F., Opdahl, A.L., Schwabe, G., Uludag, Ö., Winter, R.: Enterprise modelling for the masses – from elitist discipline to common practice. In: The Practice of Enterprise Modeling PoEM 2016. Springer (2016) 225–240
2. Fill, H.G., Karagiannis, D.: On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. Enterprise Modelling and Information Systems Architectures **8**(1) (2013) 4–25
3. Buchmann, R.: Conceptual Modeling for Mobile Maintenance: The ComVantage Case. In: 47th International Conference on System Sciences, IEEE (2014)
4. Winter, R.: Working for e-Business – The Business Engineering Approach. International Journal of Business Studies **9**(1) (2001) 101–117
5. 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
6. Scheer, A.W.: Architektur integrierter Informationssysteme - Grundlagen der Unternehmensmodellierung. Springer, Berlin (1991)
7. Fill, H.G., Karagiannis, D., Lichka, C.: Integration of Conceptual Models and Data Services Using Metamodeling. In: IEEE SoEA4EE Workshop, IEEE (2015)
8. Chen, P.P.S.: The entity-relationship model-toward a unified view of data. ACM Transactions on Database Systems **1**(1) (1976) 9–36
9. Roussopoulos, N., Yeh, T.: An adaptable methodology for database design. IEEE Computer (May 1984) (1984)
10. Kramer, F., Thalheim, B.: Holistic Conceptual and Logical Database Structure Modeling with ADOxx. In Karagiannis, D., Mayr, H., Mylopoulos, J., eds.: Domain-Specific Conceptual Modeling. Springer (2016) 269–290
11. Sinz, E.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM). Bamberger Beiträge zur Wirtschaftsinformatik **10** (1992)
12. Ferstl, O.K., Sinz, E.: Grundlagen der Wirtschaftsinformatik. De Gruyter Oldenbourg (2013)
13. Goetzinger, D., Miron, E.T., Staffel, F.: OMiLAB: An Open Collaborative Environment for Modeling Method Engineering. In Karagiannis, D., Mayr, H., Mylopoulos, J., eds.: Domain-Specific Conceptual Modeling. Springer (2016) 55–76
14. Karagiannis, D., Kuehn, H.: Metamodeling platforms. In: EC-Web 2002 – Dexa 2002. Springer (2002) 182
15. Ferstl, O.K., Sinz, E.J.: Grundlagen der Wirtschaftsinformatik. 7. edn. Oldenbourg, München (2013)
16. Bork, D., Fill, H.G.: Formal aspects of enterprise modeling methods: A comparison framework. In: 47th Hawaii International Conference on System Sciences, IEEE (2014) 3400–3409
17. 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.: ICEIS 2012 - 14th International Conference on Enterprise Information Systems, Portugal, SciTePress (2012)