An Engineering Approach for the Design of Hybrid Modelling Methods

Dimitris Karagiannis(✉) and Margit Schwab

Universitaet Wien, Research Group Knowledge Engineering,
Waehringer Strasse 29, 1090 Vienna, Austria
{dk, ms}@dke.univie.ac.at

Abstract. A fast moving business environment requires flexible and open conceptual modelling approaches for the discussion of diverse needs from the business point of view. In the conception and design of these business needs and resulting requirements, manifold modelling languages and methods on different levels of the realisation process are available. Despite their number, flexibility in this paper is referred to the exchange of 'building blocks' of modelling frameworks and the composition of *hybrid modelling methods*. That this claim for flexibility does neither affect efficiency nor goes at the account of a sound conception of the hybrid modelling method, a deliberate procedure comprising different steps is required. We call this procedure *conceptualisation*. At the end of this procedure an implementation on a meta-modelling platform is performed and the result of the conceptualisation process is a 'deployable tool'. Furthermore we imply the hybrid modelling method is of a graphical, semi-formal kind and based on a meta-modelling approach. The conceptualisation is a platform dependent activity. In the paper at hand the meta-modelling platform ADOxx® is used.

Keywords: Meta-model · Modelling language · Hybrid modelling · Adoxx® · Meta-modelling platform · Method engineering

1 Introduction

In the disciplines of informatics and business informatics models form in many cases the methodological basis, but represent in particular the connection to reality. Depending on the scenario underlying the model construction some of the situations that are described in form of models are known. In a diversity of 'developing situations' the design of models is required because reality itself does not yet exist [9]. In both cases a deliberate consideration of the described concepts as well as expected behaviour is insofar of essential relevance. Aligned to these considerations and resulting requirements the modelling language intended to be used for the model construction must offer an appropriate set of expressions and incorporate topic related concinnity. The more expressive the modelling language is the more detailed information can be modelled. The better the abstraction of generic modelling concepts succeeds the more expressive is the language.

J. Cordeiro et al. (Eds.): ICEIS 2012, LNBIP 141, pp. 3–17, 2013.
DOI: 10.1007/978-3-642-40654-6_1,

Based on the initial requirements the different modelling methods are developed for a specific domain and for a particular purpose. In a 'reuse scenario' of the modelling language where the application scenario respectively the purpose of the models vary, not all of the modelling concepts the language offers, may be meaningful for or meet the needs of 'the new scenario'. This is even true for basically generically held modelling languages. A surplus of modelling concepts is certainly a negligible 'annoyance', if there are too little or semantically different modelling concepts as required, then this is graver. The development of a new modelling language could solve this issue. Another scenario is that the missing concepts can be regained in another modelling language respectively modelling method. In such a case a new development might not seem feasible, e.g. too laborious, but the composition of a 'user-need-specific' modelling method is a tempting approach. This is due to several reasons for example the availability of established modelling concepts with an already clear recognition and clear in their semantic interpretation, an explicit usage scenario with an already determined high level aggregated versus micro-flow knowledge representation, an already solved solution approach for a certain problem, and once established – the alignment to the existing knowledge base, eventually existing implementation examples in form of tools, etc.

In the paper we speak of *hybrid modelling* for describing the process of creation of such a 'user-need-specific' modelling method. We assume an actual modelling need and therefore discuss this process from the angle of 'implementation entailing preparation steps', i.e. the *conceptualisation*. We will highlight the fundamental integration problems of this undertaking and discuss them from a meta-model point of view, from syntactical, structural and semantic heterogeneity. We use the meta-modelling framework of Karagiannis/Kühn for the definition and distinction of the different parts of a modelling method [5].

The paper is organized as follows. Section 2 is devoted to explications on hybrid modelling and resulting integration challenges. Section 3 describes the actual procedure for hybrid modelling in form of a conceptualisation life cycle. Section 4 concludes the paper and gives an outlook on further research questions to be addressed.

2 Hybrid Modelling

The topic of hybrid modelling is predominantly related to two major disciplines – conceptual and enterprise modelling. In the informatics domain conceptual modelling has first emerged for the description and design of databases in form of semantic data models, of programming languages in form of object-orientation and for example UML and of artificial intelligence, AI in form of knowledge representation, e.g. description logics [2, p. 4]. The development of conceptual modelling shows that it was at its beginning a sole computer science topic but got increasingly adopted in enterprise modelling at this stage. In enterprise modelling scenarios the angle on information that is represented in form of models is more related to business and management needs and adds in particular the 'process perspective' [11]. In both strands distinct modelling languages offering appropriate modelling concepts have been developed for expressing the specific requirements of the different domain.

Traditionally conceptual modelling languages are rather data-structure-oriented, a property that represents the major difference in contrast to enterprise modelling. In enterprise modelling the need is to represent business processes and the way how to process content from a business logic view in form of models. This need requires other modelling concepts [4, pp. 12–20].

An increasingly powerful IT-support enables gradually more comprehensive and complex modelling approaches. In order to facilitate the business needs with flexible and deliberate IT-solutions modelling frameworks tend to offer a vertical integration of different 'conceptional' levels, i.e. from a high level strategic goal definition to the actual micro flow representation within a specific IT system. Hand in hand with these comprehensive approaches goes that the actual 'end user' of these modelling frameworks must show comprehensive knowledge about the 'intended use' of these frameworks also. Still, in most cases the offered modelling frameworks provide a certain direction with the supported application scenarios, e.g. process versus actor-goal oriented modelling approaches. We claim that due to diverse business requirements in a fast moving environment amplified flexibility and open modelling approaches in form of a 'modular construction system' are of need – the hybrid modelling approach.

Despite, the meaning of hybrid defined in the Oxford English Dictionary as "[…] anything derived from heterogeneous sources, or composed of different or incongruous elements; in *Philol.* a compound formed of elements belonging to different languages […]" the resulting hybrid modelling method must fulfil certain formalisms [10].

The first requirement is that the result of hybrid modelling is a modelling method that falls in the category of a graphical semi-formal modelling method based on a meta-modelling approach. This output-related requirement already claims certain formalisms regarding the composition and structure of the modelling method.

A second requirement is that the hybrid modelling method is implemented and offered in form of a modelling tool. The implementation itself uses the programming language/s as formalism/s. The better the resulting hybrid modelling method is conceptually composed, the smoother the transformation of the concepts to the codes can be done. The requirement of an implementation is based on the conviction that an efficient modelling support can only be guaranteed by a modelling tool. Hand in hand with this persuasion goes the assertion that the user requires a structured procedure for creating models that are 'machine-processable', e.g. for corresponding evaluation algorithms. In the latter case we do not focus on the creation of executable code from the conceptual model.

Adopting modelling concepts from different modelling frameworks entails several challenges on different levels regarding their integration. For all further explanations the focus is on the metamodel of language levels as depicted in Fig. 1. The classification is according to Karagiannis/Kühn [5, p. 453].

Although, the aimed structure shows a metamodel formalism, the different 'building blocks' can be of different formalisms, e.g. logic-based formal grammar. Furthermore they can exhibit diverse abstraction levels. On a metamodel level this brings along the following states in a way that metamodels vary

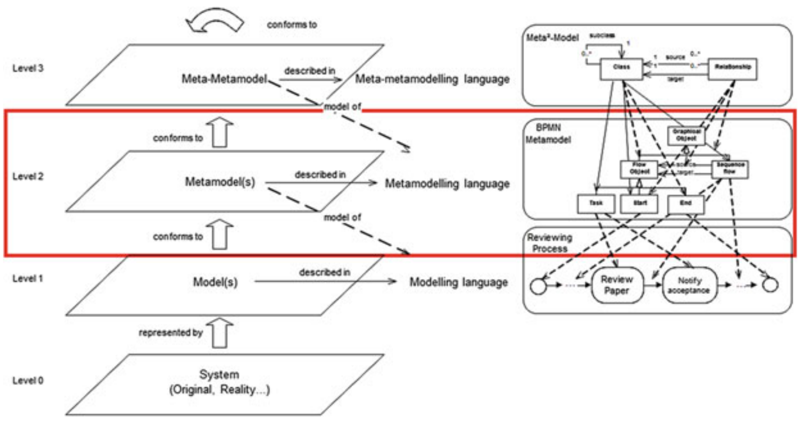


Fig. 1. Focus on the Meta-Model of Language Levels [5].

- Vertically, showing different levels of detail;
- Horizontally, the modelling concepts are on the same abstraction level but describe different aspects;
- In both ways.

One of the most essential parts in hybrid modelling is the integration of the modelling concepts and the translation from dissimilar formalism on the metamodel level. As on the metamodel level not at least as modelling language inherent constraints are determined in this integration, e.g. which association connects which modelling elements. The intricacies of a modelling method are rather found on the abstract syntax and semantic parts that also come along with the expressiveness and dynamics of the notation. This implies that the integration on a metamodel level is insufficient.

Though, the structure and constraints of a modelling language are often in focus, a modelling method provides further concepts apart from its modelling language. At first extent the 'functionality' a modelling method offers is not obvious but it becomes effective once implemented 'as a piece of software'. In this context we would like to stress mechanisms and algorithms. These are characterised by the fact that they express a form of dynamic behaviour of the modelling method and become applicable in the composition of the actual 'end user models' as instances created by means of the modelling method. Assuming that an algorithm is part of the hybrid modelling method, it requires an appropriate 'translation' of its general behaviour and semantic behaviour to the new modelling method, e.g. attributes containing input values.

For all the adopted parts and pieces apply that they were initially designed from a particular angle, the one of the initial method developer, in order to convey particular semantics and demand a particular handling in their effective use. Assuming the hybrid modelling approach does not mean a reuse of existing concepts for a further development, the initial meaning needs to be taken into account in essential accuracy.

3 The Conceptualisation Life Cycle for Hybrid Modelling

In the process of hybrid modelling, we identify three basic phases. These are the creation phase, the design phase and the compilation phase and form together the conceptualisation life cycle that is shown in Fig. 2. The first phase is related to the application scenario and the need of the user and refers basically to the selection process the user performs for identifying the existing modelling concepts within the hybrid method. The second phase is related to preparatory steps for the implementation. The most essential task is the determination of the meta-modelling platform.

The design phase cannot be done without knowing the target platform. We will concentrate on describing the tasks of that phase in the sequel. The compilation phase relates to the creation of the modelling tool. Depending on the deployment strategy different solutions are conceivable, e.g. standalone with web-access, mobile app, etc.

3.1 The Creation Phase

At the starting point of this phase the actual application scenario for which models should be created, must be clear. Based on this need the different building blocks for the hybrid modelling method can be determined. The detailed study and analysis of the selected building blocks goes hand in hand with this determination. The actual integration of the selected parts is understood as 'a merge' of existing concepts. The result of the creation phase is a detailed picture of the available hybrid modelling concepts and their dependencies, required mechanisms and algorithms including eventual requirements with regards to 'model processing' on an instance level, like user triggered data actualisations, report generation, dynamic visualisations. The detailed picture needs to be described in a way that the actual intended usage of the modelling concepts can be understandable. Depending on the maturity of the underlying building blocks the description can be of formal but also informal kind or can for example include the definition of a consistent meta-model of the hybrid modelling method on a sole conceptual level. The more concrete the description in the creation phase is, the easier is the engineering work within the subsequent design phase and the more aligned to the initial purpose of the hybrid modelling undertaking the resulting

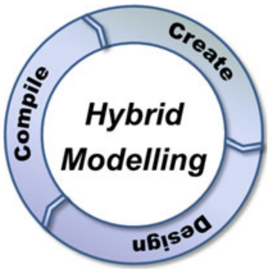


Fig. 2. Conceptualisation Life Cycle for Hybrid Modelling.

method will be. Independent of the level of formalism in the creation of a hybrid modelling method the three major parts must be addressed:

- Modelling language including syntax, semantics and notation,
- Mechanisms and algorithms and
- Modelling procedure comprising the actual usage of the modelling language as intended by the 'hybrid modelling method developer', i.e. method engineer.

Several challenges regarding transformation and integration emerge from bringing the different building blocks on a consistent level. From the angle of the method engineer this integration task is easier if the selected method and the selected platform provide the same concepts, e.g. a meta-modelling-approach.

Meta-models are not able to express all syntactical rules that have to be taken into account when creating a valid instance model that is conform to the defined modelling language [7, p. 68]. For this reason a detailed description of the three parts of a modelling language - the syntax, the semantic and notation - is of need. On this detailed level design decisions are required. These can be necessary due to the following points:

- *Syntactical heterogeneity,* which represent the difference in formats intended for the serialization of metamodels.
- *Structural heterogeneity:*
  - *Representational heterogeneity:* metamodels are represented using different metamodelling languages, each of them showing difference in its expressive power of available modelling primitives (classes, attributes, …);
  - *Schematic heterogeneity:* equal concepts are modelled either with different modelling primitives or with different number of primitives.
- *Semantic heterogeneity,* which represents differences in the meaning of the considered metamodel concepts.

Last but not least the 'look and feel' in the notations requires integration. Though it may seem the lesser part, for the untrained addressee with regards to the handling and reading of diagrammatic models, colour is for example an important distinction element to keep the available shapes and as a consequence their meaning apart. A consistent notation is a design decision from the usability point of view.

Depending on the design decisions made in the modelling language part of the hybrid modelling method, the parts 'mechanisms and algorithms' and the 'modelling procedure' part must be aligned.

3.2 The Design Phase

In the design phase the preconditions for the later implementation respectively customisation phase are elaborated. Depending on the 'degree of maturity', i.e. how many and well elaborated modelling concepts it offers, underlying formal definition, etc. of a given modelling method the design phase is more or less extensive.

The prerequisites for starting this phase are on the one hand the input from the creation phase and on the other hand the platform the hybrid modelling method is realised on. The hybrid modelling method at this stage is in a status in the range from ‘the raw script’ to ‘ready-to-use’ in its theoretical description and explains the method from a “business logic” point of view depending on the underlying domain. The meta-modelling platform, in this case ADOxx® adds the ‘technical’ point of view. The entire conceptualisation process cannot be done if the platform is not determined. The design phase of the conceptualisation life cycle shows two elementary results: a conceptualisation meta-model and an implementation meta-model.

3.2.1 The Meta-Models

The metamodel term in this paper is understood as the description model of a language, i.e. the modelling language. The metamodel comprises in itself a particular structure in form of rules and constraints how the modelling concepts are intended to be combined by the initial method developer. The metamodel is thus the grammar of the modelling language [12].

In contrast to the pure language metamodel there are the – in general generically held – concepts of the meta-modelling platform, so the meta^{2}model, i.e. level 3 of the graphic in Fig. 1. The particular challenge now is to map the language metamodel to the ‘generic’ platform concepts. We state that this is a critical task and the better this alignment succeeds the better the applicability and the more flexible the handling of the object language will be at the later stage.

Assuming a top-down approach a comprehensive analysis of the to-be-implemented modelling concepts is the first step and includes the identification of hybrid modelling method inherent dependencies and constraints. If a pure language metamodel is lacking, it is likely that the actual ‘immediate’ mapping of the modelling concepts including their dependencies respectively constraints gives a different result within the conceptualisation life cycle. The reason for this is, that ‘hierarchy of modelling concepts’ with regards to the platform metamodel can be different than from a sole ‘business logic’ point of view.

In any case the representation of the language metamodel in the platform metamodel requires design decisions. These are not necessarily due to an inaccurate description or specification of the hybrid language metamodel but related to the fact that some questions arise the first time in the preparation of its implementation. These questions are triggered by a combination of the structure of the platform and the later usability of the method. A typical example of such ‘gaps’ are the description of associations and the constraints they impose. As a popular description language for metamodels works the UML class diagram and although the cardinalities for the classes representing modelling concepts are frequently indicated, time and again the dependencies introduced due to for example inheritance assumptions are paid little attention. For the implementation the level where association-related constraints become effective are important. If the language metamodel is not expressive enough and if the method developer does not give any further information, only the analysis of eventually existing example models show the actual use of the respective modelling concepts including ‘theoretically’ integrated modelling constraints. Therefore inheritance and hierarchical concepts of both the language and the platform metamodel represent a source of additional design decisions due to the aimed implementation. There are further sources that are addressed in the sequel by means of the metamodel in a specific platform, i.e. the ADOxx® platform meta^{2}model [3, 7].[1]

Although, the analysis of the language and the platform metamodels gives information about the hybrid modelling method inherent dependencies and constraints, the actual parts of the hybrid modelling language require further attention. The conceptualisation of the notation and semantic part and from them an alignment of the syntax part, grants to the modelling language increased expressiveness and flexible handling with regards to conveying intricacies of information. Therefore the conceptualisation of the language meta-model is only part of the rent.

3.2.2 Notation, Syntax and Semantic

The structure of a graphical, semi-formal hybrid modelling method consists of three parts, the notation, the syntax and the semantic.

The Notation: The notation describes the graphical representation of the elements of the modelling language. Although the notation is very often seen as the least important part, it is a very obvious aspect as it is visualised. So, if for example the notation is not specified detailed enough, different interpretations depending on the method engineer's perception and design of the modelling concept are possible. This has for example happened in case of the *i** modelling method. Fig. 3 shows a selection of two modelling concepts originating from the *i** modelling method. Interestingly the shape of the 'Actor' modelling concept is rather consistent whereas the shape of the 'Softgoal' modelling concept is difficult to reach [13, 15].

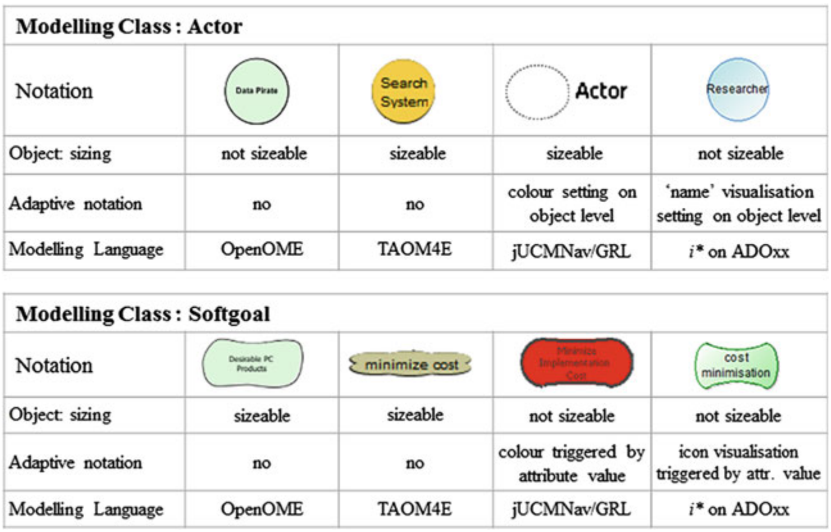
For the definition of the notation for a hybrid modelling method a similar situation is conceivable. The following combinations are likely, either there are

- More than one modelling classes for the same modelling concept providing different notations;
- Modelling concepts where the notation part has not yet been defined.

In both cases a design decision is required how to proceed for the hybrid modelling method. This design decision is again ideally done by the 'hybrid modelling language developer' in the creation phase already.

Summarising the said, the behaviour of the different classes with regards to their graphical representation is determined when specifying the notation of the hybrid modelling language. The notation is moreover related with the semantics and the syntax in a way that the graphical representation is determined by a specific attribute

[1] ADOxx® is the metamodelling platform that is used by the Research Group Knowledge Engineering of the University of Vienna for their research on metamodels, metamodelling and model languages. It is an extensible, repository-based platform, offers a three-step modelling hierarchy and is based on a rich meta-model. ADOxx® is a 'development' platform for modelling languages which are founded on a metamodel approach. Karagiannis, D.; Visic, N. (2011): "Next Generation of Modelling Platforms". In: Grabis, J.; Kirikova, M. (eds.): Perspectives in Business Informatics Research, Vol. 90, pages 19–28, Springer Berlin Heidelberg. Furthermore the projects of the Open Models Initiative are realised on this platform (http://www.openmodels.at).



Modelling Class : Actor				
Notation	Data Pirate	Search System	Actor	Researcher
Object sizing	not sizeable	sizeable	sizeable	not sizeable
Adaptive notation	no	no	colour setting on object level	'name' visualisation setting on object level
Modelling Language	OpenOME	TAOM4E	jUCMNav/GRL	*i** on ADOxx

Modelling Class : Softgoal				
Notation	Desirable PC Products	minimize cost	Minimize Implementation Cost	cost minimisation
Object sizing	sizeable	sizeable	not sizeable	not sizeable
Adaptive notation	no	no	colour triggered by attribute value	icon visualisation triggered by attr. value
Modelling Language	OpenOME	TAOM4E	jUCMNav/GRL	*i** on ADOxx

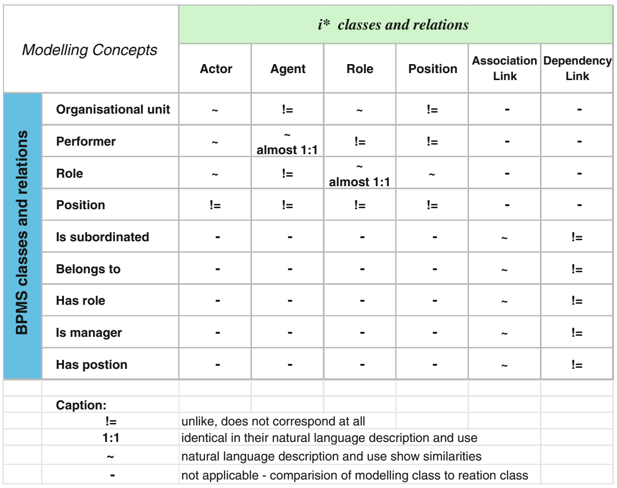
Fig. 3. Examples of Different Notations for the Same Modelling Concept.

value. The attribute value is part of the semantic whereas the attribute itself is part of the syntax.

The Semantic: The semantics describe the meaning of the modelling concepts of a modelling language, i.e. contrasting the objects in reality and which are to be mapped in the model and how the language elements have to be interpreted. The semantics are also expressed in the values the attributes defined in the syntax part can adopt. The graphical representation of the respective modelling concept eventually underlines the semantics. In the conceptualisation of a hybrid modelling method a clear delineation of the semantics of modelling concepts is of need. For a better explanation the discussion should be done by means of an example.

The hybrid modelling method should contain modelling concepts of the *i** method and of a business process modelling method, BPMS. Although the integration of parts of these two modelling languages is rather conceivable on a vertical level due to the general purpose of the modelling methods, some modelling concepts require special attention. In the *i** modelling method the 'view' of a strategic dependency model contains the modelling concepts of Actor, Agent, Role, Position, Dependency Link, Association Link and the 'intentional elements'. In the BPMS method there is the model type working environment model containing the modelling concepts of Organisational Unit, Performer, Role, Position, Is subordinated, Belongs to, Has role, Is manager and Has position. Selected elements of both modelling methods are given in Fig. 4.

Even if one would integrate the strategic dependency model and from the BPMS method the model type business process model, so sets of modelling concepts with different predetermined purposes, in the business process model consideration the Role and or Performer is a central element for the specification on which level respectively skill level the activities within the business process should be processed.



Modelling Concepts		*i* classes and relations*					
		Actor	Agent	Role	Position	Association Link	Dependency Link
BPMS classes and relations	Organisational unit	~	!=	~	!=	-	-
	Performer	~	~ almost 1:1	!=	!=	-	-
	Role	~	!=	~ almost 1:1	~	-	-
	Position	!=	!=	!=	!=	-	-
	Is subordinated	-	-	-	-	~	!=
	Belongs to	-	-	-	-	~	!=
	Has role	-	-	-	-	~	!=
	Is manager	-	-	-	-	~	!=
	Has postion	-	-	-	-	~	!=

Caption:	
!=	unlike, does not correspond at all
1:1	identical in their natural language description and use
~	natural language description and use show similarities
-	not applicable - comparision of modelling class to reation class

Fig. 4. Contrasting Modelling Concepts from the *i** and the BPMS Method.

In order to determine if for example the Role in the *i** modelling method can be reused for the business process models the assigned semantic from the description and usage interpretation of the modelling concepts is of need.

What can be learnt from Fig. 4 is that those elements that show similarities require a clear interpretation for avoiding a 'muddling through' of the modelling concepts in their actual use on an object level.

The Syntax: The syntax describes the dependencies and constraints in between the modelling concepts and is furthermore represented in the description of the properties of these in form of attributes. Almost every modelling concept offers a 'name' attribute. Other attributes are used for a comprehensive description of the domain and application scenario the modelling method is used for. An integration of attributes of semantically identical modelling concepts of two different modelling methods will rather be the creation of the 'common multiple'. In the modelling concept of a Role in the *i** method only the attribute 'name' was defined; in the Role of the BPMS method besides the 'name' attribute seventeen further attributes are defined for a comprehensive description of the modelling concept. Moreover three of them trigger by means of predefined values a change of the graphical notation [1].

The detailed steps of the design phase of the conceptualisation life cycle for hybrid modelling are discussed by Xu et al. using modelling concepts of *i** and UML to merge them to *Active i**. The *Active i** modelling method is a representative example that has emerged from hybrid modelling [14].

3.2.3 Mechanisms and Algorithms

The conceptualisation of the hybrid modelling language was only described for the language part so far. The underlying modelling framework of Karagiannis/Kühn requires the modelling procedure and algorithms and mechanisms as integrated modelling method parts. The modelling procedure becomes manifest in the composition of the hybrid modelling language, e.g. the definition of model types and an accompanying instruction manual for example in form of language-specific modelling guidelines. The mechanisms and algorithms are of different kind as they usually represent extensions of the modelling language respectively of the usage scenario of the modelling language. In this understanding the mechanisms and algorithms necessitate consideration regarding parameters which form their input and a consideration in which way they are realised in the modelling language, e.g. which attribute of what attribute type contains values required for a composition algorithm. Furthermore we learn from the modelling framework of Karagiannis/Kühn that three different mechanisms and algorithms are distinguished by the criterion if they are specific for the modelling technique or if they are generic, e.g. breadth-first-search algorithm. If the algorithms work with the modelling concepts of the hybrid modelling language, it has to be defined if the algorithms require an alignment of the syntax and related semantics parts, e.g. in form of additional attribute values to gather input values or to capture calculation results from algorithms. Further examples where algorithms affect directly the modelling concepts are if constraints are triggered by means of algorithms, e.g. in form of the verification of modelling scenarios, cardinality checks. The design of such functionality-related algorithms is essential for the ‘smooth and easy going’ use of the implemented hybrid modelling method and is of high significance within the conceptualisation life cycle.

The fact that the platform itself offers general predefined functionality, e.g. report generation or ‘only’ interfaces where additional functionality can be integrated in the general language composition forms another aspect in the conceptualisation of mechanisms and algorithms. Conceptualisation tasks for predefined functionality are different as it is rather a configuration than actual code design that is required for language specific algorithms.

3.3 The Compilation Phase

The compilation phase is the last of the conceptualisation life cycle. In this phase it has to be decided in which way the realised hybrid modelling method should be offered to the end user. Depending on the scope and how comprehensive the method is in its composition some deployment alternatives are preferable to others, e.g. the deployment in form of a modelling app. The deployment variant also determines how the hybrid modelling method is intended to be used. The composition by the use of a terminal server and by means of a web-interface allows from the end user point of view quick access without taking care of any further installation routines. Furthermore the hybrid modelling method is more easily available for a bigger community. The deployment of the hybrid modelling method as an independent distributable unit, e.g. standalone version, is another option. Which deployment variant is meaningful depends on the actual ‘business model’ for the hybrid modelling method and represents the last design decision. In the compilation phase the actual realisation of the elaborated hybrid modelling is performed. We call this task customising as it is performed on an existing meta-modelling platform.

3.3.1 Platform Specifics

Depending on platform internal procedures, the actual sequence of implementation steps is predetermined. In order to allow defining the respective conceptualised parts, the provision of appropriate formalisms, e.g. programming or scripting languages, is required. These formalisms provide the actual support to codify the specifications and are either platform-specific, e.g. AdoScript for ADOxx® or rather generic like for example Java. ADOxx® offers integrated dialogs for the realisation of the conceptualised metamodel, the syntax creation in form of attributes and the design of the notation. Each dialog encapsulates specific functionality for achieving the expected result. For the mechanisms and algorithms 'message ports' are offered for a seamless integration. Depending on how specific the mechanisms and algorithms for the modelling technique are, not all implementation steps require actual coding. Some steps are solely configurations of generic platform functionality according to the hybrid modelling language needs, for example mechanisms for the notebook structure, predefining queries, print layouts or embedment of new menu buttons for launching algorithms and are therefore classified as 'customising' steps. The different nested formalisms guide the actual realisation on the platform and are prerequisite for a well-rounded, easy to handle modelling method.

3.3.2 Customising

The actual implementation belongs to the compilation phase of the conceptualisation life cycle.

In the ADOxx® meta^{2}model the central element is the 'library' that works as a container to which all formalisms and constructs of one of its instances, i.e. the modelling language metamodel are assigned to. The first step is the set-up of the platform-conceptualised metamodel of the modelling language. In this task, the ADOxx® meta^{2}model distinguishes between classes and relation classes. For relation classes at least two endpoints need to be defined and these are basically the specification *from* which modelling class *to* which modelling class the association is allowed to be drawn.

Once the basic 'skeleton' of the modelling language is given, the immediate next step would be the definition of the syntax. The modelling classes as well as relation classes have different attributes that are from predefined attribute types. For the platform the commonly known attribute types like integer, double, string, enumeration, etc. are defined. For the platform conceptualisation a distinction between class attributes and instance attributes is made. The difference between these two lies in the values the attribute can adopt. Class attributes are context neutral and not to be filled by the end user or modeller using the method once implemented. Instance attributes are context dependent and will be used by the modeller to capture data and convey certain information [8, p. 100].

In ADOxx® the concept 'notebooks', i.e. the dialog structure is defined by a number of attributes. 'Notebooks' need to be specified for those modelling and relation classes where attributes have been defined whilst the syntax realisation. In the ATTREP dialog of the platform, the attributes are summarised in chapters and sub-groups. Theses structural elements influence the display of the attributes for the 'end user' of the modelling method.

A further concept of the meta^{2}model platform shows, though on another level but most essential for the use of the modelling method, the structural element 'model types'. A model type is a classification element for the available modelling concepts of a modelling method. For the end user model types are 'a predefined set' of modelling and relation classes that specify the purpose of a model. The concept of model types is similar to the 'diagram types' classification within the UML modelling language. The definition, which elements are within a model type and which are not, is sometimes difficult and can best be answered by 'observing' the actual use of modelling concepts, i.e. by means of the created models. The source of difficulties is actually if a model is created by means of a particular model type and the further development of that same model, i.e. same instance, should be done by modelling classes that are assigned to a different model type. Such a 'model development' procedure is related to the modelling procedure part but influences the implementation at this stage. A solution for avoiding such handling constraints is either to foresee the same modelling concepts in more than one model type or to work with 'view mode' concepts. The platform specifies in this context the 'Mode' concept.

The last part of the modelling language, the notation part is realised in the GRAPHREP dialog of the ADOxx® platform for the modelling and relation classes. As discussed, the notation can show static and dynamic parts whereas in this context the conceptualisation goes as far as to the description of the 'behaviour' of the notation, e.g. change of graphical representation if a certain condition is fulfilled. In the actual realisation it has to be determined if such a dynamic part is realised by the notation-specific formalism or if it is more efficient when realising it by means of a language-specific algorithm and hence a different formalism.

The actual coding of modelling technique-related algorithms forms together with the configuration of mechanisms the last customising part of a hybrid modelling method realisation.

Subsequently the compilation of the hybrid modelling method to a modelling tool winds-up conceptualisation life cycle.

4 Conclusions and Future Work

In the paper at hand it has been discussed that the design and conceptualisation of hybrid modelling methods require a number of steps where deliberate design decisions are of need. The conceptualisation is a prerequisite for an implementation in form of a 'self-contained modelling tool'. In order to structure the different steps that are required for achieving this goal, a conceptualisation life cycle is suggested. Each of the single phases of the cycle shows a particular focus and contains parts where due to additional parameters provided by the platform a further design is of need. As the platform the hybrid modelling method is supposed to be realised on, provides functionality and structures which have not been relevant during the development phase of the hybrid modelling method 'on paper' more design decisions given by the platform logics are necessary.

Further work is required for integrating formalisms to describe language-specific algorithms apart from pseudo code which we consider as insufficient, providing mechanisms for reusability of concepts and transformations on a meta-model level.

References

1. BOC_Group: BPMS Method Manual for ADONIS 5.0, p. 1206. BOC Asset Management GmbH, Vienna (2012)
2. Brodie, M.L.: John Mylopoulos: Sewing Seeds of Conceptual Modelling. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 1–9. Springer, Heidelberg (2009)
3. Fill, H.-G., Redmond, T., Karagiannis, D.: FDMM: A Formalism for Describing ADOxx Meta Models and Models. In: Maciaszek, L., et al. (eds.) Proceedings of ICEIS 2012 - 14th International Conference on Enterprise Information Systems, Wrocław, Poland, Vol. **3**, pp. 133–144. SciTePress (2012)
4. Frank, U.: Multiperspective enterprise modelling: theoretical background and design of an object-oriented development environment (German original: Multiperspektivische Unternehmensmodellierung: Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung). Habilitation Thesis, Munich, R. Oldenbourg Verlag (1994)
5. Karagiannis, D., Kühn, H.: Metamodelling Platforms. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) EC-Web 2002. LNCS, vol. 2455, p. 182. Springer, Heidelberg (2002)
6. Karagiannis, D., Visic, N.: Next Generation of Modelling Platforms. In: Grabis, J., Kirikova, M. (eds.) BIR 2011. LNBIP, vol. 90, pp. 19–28. Springer, Heidelberg (2011)
7. Kühn, H.: The ADOxx® Metamodelling Platform. In: 1st Open Models Workshop Methods as Plug-Ins for Meta-Modelling. http://www.openmodels.at/web/omi/blogs/-/blogs/1st-international-workshop-on-omi?_33_redirect=%2Fweb%2Fomi%2Fblogs (2010). Accessed 07 November 2012
8. Kühn, H.: Method integration in business engineering (German Original: Methodenintegration im Business Engineering. Unpublished Thesis p. 284. Faculty of Business Administration and Informatics, University of Vienna, Vienna (2004)
9. Mahr, B.: Information science and the logic of models. Softw. Syst. Model. **8**(3), 365–383 (2009)
10. OED: hybrid, adj. Oxford_English_Dictionary (ed.). Online version of September 2012. http://www.oed.com/view/Entry/89809?redirectedFrom=hybrid#eid (2012). Accessed 29 October 2012
11. Staud, J. L.: Enterprise Modelling (German original: Unternehmensmodellierung), p. 379. Springer, Heidelberg (2010)
12. Strahringer, S.: Metamodel (German original: Metamodell). Enzyklopaedie der Wirtschaftsinformatik Kurbel, K., et al. (eds.), http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Problemanalyse-/konzeptuelle-modellierung-von-is/metamodell/index.html (2012). Accessed 07 November 2012
13. i* Wiki.: iStar Tools. http://istar.rwth-aachen.de/tiki-index.php?page=i%2A%20Tools (2011). Accessed 20 April 2011

14. Xu, T., Ma, W., Liu, L., Karagiannis, D.: Synthesizing Enterprise Strategic Model and Business Processes in Active-i*. In: 2010 14th IEEE Enterprise Distributed Object Computing Conference Workshops (EDOCW), pp. 345–354 (2010)
15. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering, p. 760. MIT Press, Cambridge (2011)