Hans-Georg Fill and Dimitris Karagiannis

On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform

In this paper we analyse the conceptualisation of modelling methods. Thereby we understand, how the components of a specific implementation platform support the design of modelling methods. For this purpose we use the ADOxx meta modelling platform and investigate, how four selected functionalities of enterprise information systems for supporting user interaction, process-based optimisation, interfaces to other systems, and complex analyses are realised. We discuss these four functionalities by reverting to excerpts of the visual representation of modelling methods from the areas of requirements engineering, business process management, e-learning, and enterprise architecture management. This permits us to highlight the interdependencies between the modelling language, the modelling procedure, mechanisms and algorithms and the functionalities of the underlying technical platform that have to be taken into account during the conceptualisation.

1 Introduction

In the domain of information systems a large variety of different types of enterprise modelling methods are today being used both in academia and industry. These range from modelling methods on the strategic level (e.g., Ronaghi 2005), the business process and organisational levels (List and Korherr 2006) to modelling methods for describing and implementing IT architectures and software systems (e.g., Aier et al. 2009). For the successful handling of modelling methods, the use of modelling tools is today regarded as state-of-the-art. These tools not only support the definition of modelling languages and thus ease the creation of machine-processable representations of the models' contents. They also provide facility services, e.g., for accessing, exchanging and persistently storing meta models and models, for applying algorithms or for querying model contents (Karagiannis and Kühn 2002).

In the course of the implementation of a modelling method in a modelling tool, several design decisions have to be taken. These decisions directly correspond to the intended use of the modelling method and involve tasks such as the definition of the modelling language, the specification of the modelling procedure, i.e., how the modelling language is used to achieve results, as well as the specification of mechanisms and algorithms to ensure an adequate user experience and perform the machine-processing of the models. Thereby, it has to be taken into account, how the components of the chosen implementation platform support the design of a modelling method. We will denote this aspect as the *conceptualisation of a modelling method*. In our view this conceptualisation process that so far remained more of an art than a science, needs to take into account both aspects, i.e., a combination of artistic and scientific concepts, to realise appropriate design and processing functionalities.

Therefore we will analyse the conceptualisation in respect to four selected functionalities that are typically provided by so-called *meta modelling platforms* that support the implementation of arbitrary types of modelling methods and yield as a result modelling tools, i.e., software applications for interacting with a modelling method. The functionalities we will regard for the conceptualisation are *visualisation* and *transformation*

functionalities, as well as analysis functionalities for *simulation* and *querying*. In order to focus on the process of the conceptualisation, we restrict our investigation to modelling methods that are based on the same meta meta model and that make particular use of these functionalities. For this purpose we revert to modelling methods that are based on the ADOxx[1] meta meta model and that are available via the Austrian section of the Open Models Initiative[2] (Karagiannis et al. 2008; Koch et al. 2006). The reason why we chose the ADOxx meta modelling approach and the corresponding software platform is, that it has been successfully developed, used, and tested for over more than fifteen years in a large number of research and industrial projects that included some of the largest German and Austrian companies as customers. It is therefore an industry-proven approach that goes far beyond a research prototype in terms of functionalities, scalability, and reliability. In this way we will be able to derive insights into the conceptualisation of modelling methods that are also relevant from an industry perspective.

The remainder of the paper is structured as follows: in Sect 2 we will briefly discuss the foundations for our analysis. In particular we will describe our view on the components of modelling methods and the meta modelling in ADOxx. Work related to the conceptualisation of modelling methods will be reviewed in Sect 3. This will then permit us to perform the analysis of the conceptualisation for the visualisation, transformation, simulation and querying functionalities in Sect 4. The paper will conclude with a discussion of the results of the investigation in Sect 5 and an outlook on the future work in Sect 6.

[1] ADOxx is a commercial product and trademark of BOC AG.

[2] See `http://www.openmodels.at` for an overview of current projects, community information, and foundations on the Austrian section of the Open Models Initiative.

2 Foundations

In this section we will outline the foundations for describing the components of modelling methods from a general perspective. Subsequently, we will present the core constituents of the ADOxx meta modelling approach including the underlying meta meta model.

2.1 Components of Modelling Methods

To clarify our understanding of the terms and elements of modelling methods and thus provide a solid basis for the further discussion, we revert to a framework that has originally been proposed by Karagiannis and Kühn in 2002 – see Fig. 1. In this framework a *modelling method* is composed of a *modelling technique* and *mechanisms and algorithms*. Thereby the modelling technique is further divided into a *modelling language* and a *modelling procedure*. The modelling procedure consists of *steps* for defining the application of the modelling language and delivers *results*. For this purpose it reverts to mechanisms and algorithms. The modelling language has a *syntax* that defines the grammar and *semantics* that defines the meaning of the elements of the syntax. This is achieved by a *semantic mapping* that connects the syntactical constructs with their meaning defined in a *semantic schema*. The semantic schema may either be formally defined or may come in the form of (informal) textual descriptions as it is used, e.g., in the definition of the UML or BPMN (Object Management Group OMG 2007, 2011a).

In contrast to other approaches such as described, e.g., by Harel and Rumpe (2000), the *notation* defines the visualisation of the modelling language through the elements of the syntax and by obeying the attached semantics. The notation may either be defined in a static way, e.g., by using pixel-based or vector images, or in a dynamic way by splitting it in a representation and a control part. By using the second direction, the control part can dynamically adapt the representation part depending on the current state

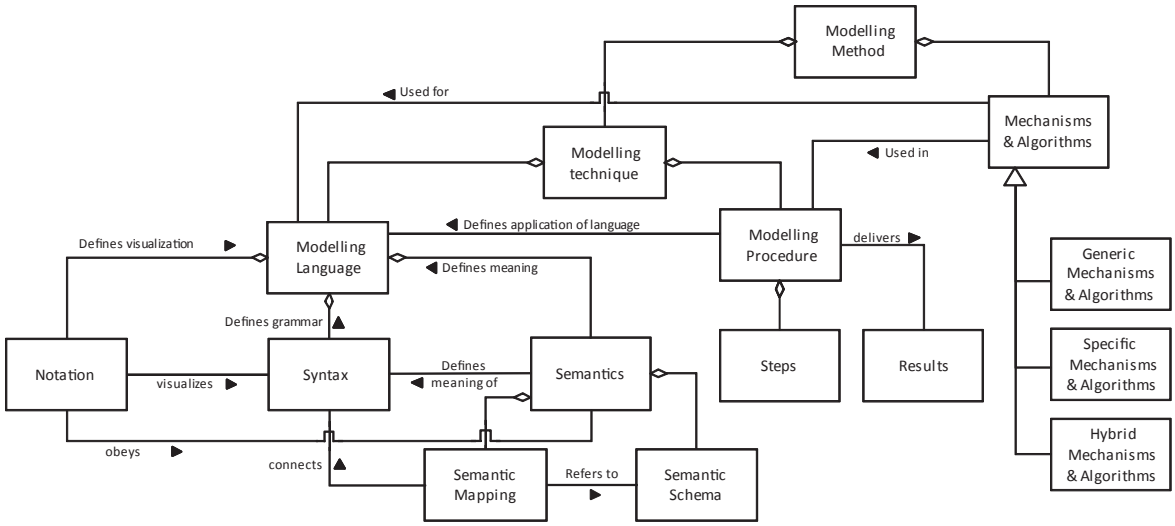

Figure 1: Components of modelling methods (Karagiannis and Kühn 2002)

of a model. Finally, mechanisms and algorithms are used for the modelling language and in the modelling procedure. *Generic mechanisms and algorithms* can be applied to arbitrary modelling languages, *specific mechanisms and algorithms* only to particular modelling languages and such of the *hybrid* type are specific ones that can be configured for several modelling languages.

2.2 Meta Modelling in ADOxx

As there exist several different views on the term *meta model*, we will also briefly describe the view we will employ in the following. Therefore we revert to the characterisations used by Harel and Rumpe (2000, 2004); Sprinkle et al. (2010) who denote a *meta model* as a representation of the abstract syntax of a modelling language and a *model* as the representation of its concrete syntax. Also the meta model itself can be described by using a modelling language, which is then denoted as the *meta modelling language*. From this follows that the abstract syntax of the meta modelling language can be represented by a *meta meta model* or *meta*^{2} *model* (M^{2}M) for short and the concrete syntax of the meta modelling language defines the meta model (Kern et al. 2011).

When using a meta modelling approach to implement modelling languages, the meta^2 model is typically held fixed and thus provides the basic elements for describing a meta model (Sprinkle et al. 2010). This direction was also chosen for the ADOxx approach that has evolved from the conception of the Adonis business process management toolkit (Junginger et al. 2000). ADOxx

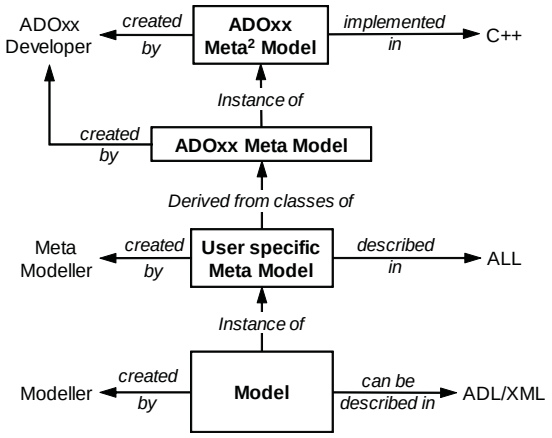

Figure 2: Roles and languages in the modelling hierarchy of ADOxx

is today available as a commercial product. An open use version for academic purposes is available for projects via the Austrian section of the

Open Models Initiative[3]. In Fig. 2 the roles and languages of the ADOxx approach are depicted. The ADOxx meta^2 model is implemented in the C++ programming language and created by the developers of ADOxx. From this meta^2 model the ADOxx Meta Model is instantiated that provides a set of pre-defined constructs. As will be shown later in this paper, these constructs are for example necessary to realise simulation functionalities in ADOxx. User specific meta models are derived from classes of the ADOxx Meta Model and described by a meta modeller using the proprietary ADOxx Library Language (ALL). This language provides concepts for describing meta models by using the constructs defined in the ADOxx meta^2 model. From these user specific meta models the actual models are created as instances by the modellers. These models can be described in the proprietary ADOxx export format ADL or in XML.

For the instantiation of meta models, the ADOxx meta^2 model provides the constructs shown in Fig. 3. Its core part comprises *classes* and *relationclasses* that are grouped by *model types*. Based on the shown cardinalities, model types must have at least one class assigned. In addition, model types may be further detailed by *views* that can restrict which classes and relationclasses are shown. Both classes and relationclasses have *attributes* that can be detailed by *facets* such as a helptext or regular expressions to further constrain the attribute values. A special type of attributes are *class attributes*. Two types of class attributes are defined: *notebook definitions* for attribute representation definitions in the ATTRREP grammar, that determine which attributes are visible in the dialogues of modelling objects to the modeller[4]; and *graphical representations* that define the dynamic visual representation of classes and relationclasses in the proprietary GRAPHREP grammar. The *instance attributes* can be of various types including single- or multi-value data types such as string, float and integer as well as the special types *interref* and *record class*. Attributes of the type interref are used to reference other objects in the same or a different model and other models as a whole. Attributes of the type record class are collections of attributes of the other types which are represented in a table-based structure. Classes can be arranged in a hierarchy by defining sub-class relationships between them. Thereby the attributes of a superclass are inherited by its sub-classes. Relationclasses always need to define exactly one class for its source (Is From-Class) and one class for its target (Is To-Class). Both classes and relationclasses may either be pre-defined, i.e., to constitute the ADOxx Meta Model or user-defined, i.e., to define a user-specific meta model, which are grouped in a *user-defined class hierarchy*. Recently, also a formalism has been developed to describe ADOxx meta models and models in a mathematically exact way – we refer the interested reader to the specification of the FDMM formalism, cf. Fill et al. (2012).

Based on the constructs defined by the meta^2 model, ADOxx provides a set of functionalities for supporting the realisation of modelling methods. These are implemented in the form of components on top of a modelling sub-system, which is a database-driven client-server repository – see the architecture of ADOxx shown in Fig. 4. The *acquisition* component is used to acquire external data, e.g., in the form of spreadsheets and make it available as models; the *modelling* component is the central component for handling models and provides visual model editors that are automatically generated based on the supplied meta models – it is also responsible for the visualisation of models; the *analysis* component provides mechanisms for formulating and executing static queries; the *simulation* component provides four different algorithms that are based on principles of discrete-event simulation; the *evaluation* component supports the comparison

[3] See `http://www.openmodels.at`.

[4] The name for these definitions originates from the fact that these dialogues are visualised in the model editors in a way that resembles notebooks.

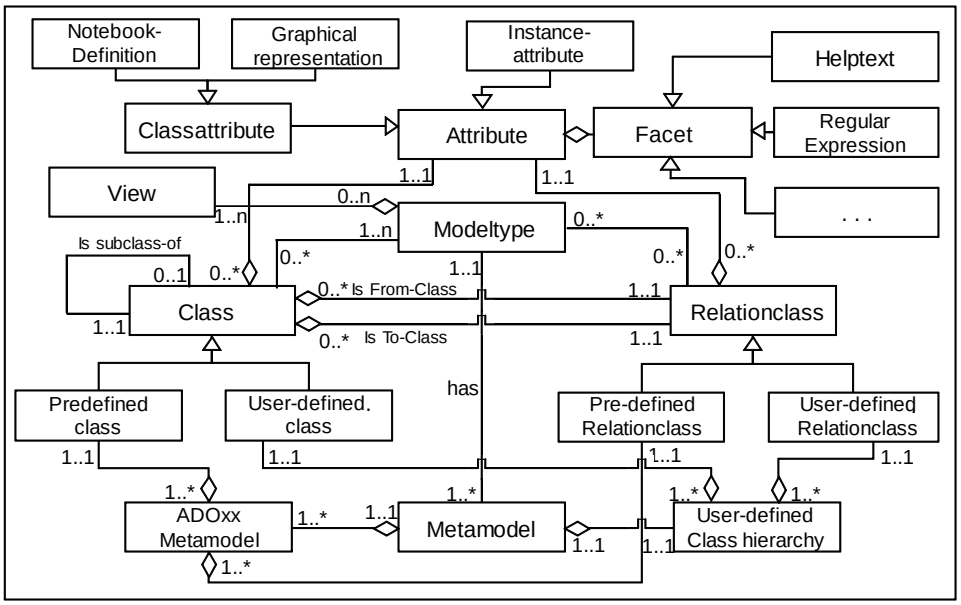

Figure 3: The ADOxx meta^2 model

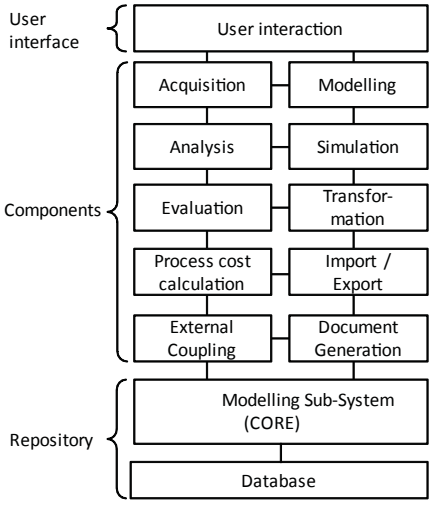

Figure 4: Architecture of ADOxx showing the components that can be used for conceptualising modelling methods

of results acquired both from the simulation component as well as from external real-time data such as audit trails from workflow management systems; the *transformation* component handles the transformation of models to formats used by CASE tools; the *process cost calculation* component provides algorithms for determining the cost of processes based on simulations; the *import/export* component provides generic algorithms that work on all ADOxx based meta models to import and export files in ADL and a generic XML format; the *external coupling component* is used to add mechanisms and algorithms that are not part of any of the other components; and finally the *document generation* component supports the automatic generation of document formats such as HTML, RTF or DOC from models and can be configured to individual needs.

3 Related Work

Before we investigate the conceptualisation of modelling methods on ADOxx we will briefly review related work. A more general view on methods and how they are being developed has been discussed in the context of *method engineering* (Brinkkemper et al. 1999; Harmsen and Saeki 1996). However, although there are several publications in method engineering that discuss the construction of methods and often refer to meta models as well, the design of modelling languages or the conceptualisation of modelling methods does not seem to be a focus of this discipline. As Frank states, the approaches in method

engineering "mainly focus on the construction and configuration of process models and take the modelling language as given" (Frank 2011, p. 94). Method engineering thus investigates how the generic structure of development processes is codified in methods (Kurpjuweit and Winter 2007; Odell 1996), which corresponds partly to the aspect of a modelling procedure we discussed in Sect 2.1.

In regard to the development of modelling languages, several authors have recently proposed design guidelines and patterns. Thereby, it has to be distinguished between the development of domain specific languages (DSL) on the general level and domain specific modelling languages (DSML) on a more specific level. Although domain specific languages sometimes also revert to graphical representations, they are often characterised by a comparison to general-purpose programming languages in the way that they offer a better expressiveness and ease of use for a particular domain of application (Mernik et al. 2005). Domain specific modelling languages, in the way we consider them for the scope of this article, however constrain the properties of DSLs. For example, the use of a visual notation and the optional use of formal semantics is - although also valid for some DSLs - a feature that is commonly assumed for DSMLs. Focusing on the design of a DSML, it can again be distinguished between guidelines that provide general suggestions on the overall design and guidelines that focus on specific aspects of language design.

As an example for the first direction, the nine principles presented by Paige et al. can be consulted. These state for example that a modelling language should not contain unnecessary complexity, redundant or overlapping features, be consistent, use the same abstractions throughout the development or ensure that concise models are produced (Paige et al. 2000). More specific guidelines on modelling language design are for example given by Frank who defines six criteria for determining whether a term is suited to be incorporated in a language as a concept (Frank 2011). These guidelines define the general frame for the conceptualisation of modelling methods and are applicable in any tool environment.

For the composition of meta models based on existing meta models and model fragments, Emerson and Sztipanovits propose the merging and interfacing of meta models and the refinement of classes for composing new meta models (Emerson and Sztipanovits 2006). For a concrete case, such a synthesis of two existing modelling methods is, e.g., shown by the examples of UML activity diagrams and iStar by Xu et al. (2010). Similarly, Moody presents nine principles for designing cognitively effective visual notations for modelling languages that can guide the developer of a modelling language (Moody 2009).

Concerning other meta modelling frameworks and platforms several approaches are available today, cf. Kern et al. (2011). Any of these could be used for analysing the conceptualisation of modelling methods - however, we can identify two aspects that depend on the properties of the underlying meta modelling approach and that need to be taken account. The first aspect concerns the functionality provided by the meta modelling platform and the second aspect concerns the practical realisation of modelling tools.

To illustrate the specificities of the ADOxx approach for these aspects we can compare it to the approach of the Eclipse Modelling Framework (EMF, cf. McNeill 2008) and MetaEdit+ (cf. Tolvanen and Rossi 2003) as two prominent examples in the field. In case of the EMF, meta models are primarily developed in a Java environment. A method engineer therefore has to have in-depth knowledge of the Java programming language to specify a modelling language and then to implement a visual model editor using the Eclipse Graphical Editing Framework (GEF). To abstract from this technical view, the Graphical Modelling Framework (GMF) has been created. GMF supports the implementation of visual model editors on top of EMF by partially automating the construction of a number of intermediate models (Aniszczyk 2006). However, as Kolovos et al.

(2009, p. 1) remark, "implementing a visual editor using the built-in GMF facilities is a particularly complex and error-prone task and requires a steep learning curve."

In comparison, the ADOxx approach does not require any knowledge of a programming language. Instead, meta models can be specified visually and the model editors are automatically created either for desktop or web usage without any compilation as required in the Eclipse case. At the same time, ADOxx automatically provides a multi-user environment and a repository based on a relational database for meta models and models. Although these functionalities may also be added for Eclipse, they would require extensive additional programming effort. Another distinguishing factor of ADOxx and EMF is that Eclipse and EMF/GMF are provided on an open source basis and ADOxx on an open use basis. Therefore, if a developer wants to extend the functionality of EMF or GMF to fit his or her own needs this is possible. For ADOxx only the given functionalities together with the provided extension mechanisms and external interfaces can be used.

MetaEdit+ shares several similarities with ADOxx. For example, it also features the easy, visual definition of meta models and the corresponding graphical representation of models as well as the provision of a repository. The difference to MetaEdit+ lies however in the focus on different domains. Whereas ADOxx has been developed for the domain of enterprise information systems, MetaEdit+ "is intended to provide a platform for developing CASE tools" (Kelly et al. 2005, p. 26). Thereby, CASE stands for Computer Aided Software Engineering that aims to support software system developers in improving the quality and efficiency of the software development process (Case 1985).

In more detail, MetaEdit+ is a "customisable CASE environment" (Rossi et al. 2004, p. 374) that supports building and the integration of multiple methods. This leads, in the case of MetaEdit+, to a focus on code generation, model analyses, and the creation of reports (Kelly et al. 2005), which are essential features for software developers to speed up the development of systems. ADOxx in comparison does not specifically focus on software development and thus does not provide specific functionalities for code generation. Rather, ADOxx takes a broader view and provides a number of business related functionalities such as process simulation, evaluation, or process cost calculation.

In these latter aspects ADOxx also differs from other modelling approaches such as GME (Ledeczi et al. 2001) that targets the domain of electrical engineering and MOFLON (Amelunxen et al. 2007) that builds upon the meta modelling paradigm of the Meta Object Facility (MOF). Although MOF can also be used to implement meta models, it is primarily a metadata management framework to enable the development and interoperability of model and metadata driven systems (Object Management Group OMG 2011b, p. 5).

When summarising what has been achieved so far, it can be found that in the past there has been a strong focus on the design of modelling languages and their realisation in modelling tools. What is missing is a holistic view on the conceptualisation of modelling methods. This concerns for example the complex interrelationships between the tasks originating from a modelling procedure, how they are supported by using appropriate mechanisms and algorithms and how this influences the design of a modelling language. At the same time, these interrelationships depend on the underlying meta modelling platform and its capabilities in terms of its meta^2 model and the provided platform components.

4 Conceptualisation of Modelling Methods on ADOxx: Selected Functionalities

With the foundations presented above we can now start to investigate how the conceptualisation of modelling methods is conducted. To start

with, we have identified four possible strategies for an analysis that take into account the *conceptualisation base* (CB) and the *meta^2 model* (M^{2}M) of the chosen meta modelling approach.

The conceptualisation base stands for the available specifications of a modelling method and typically comes in the form of informal, natural language descriptions in documents or books, semi-formal descriptions, e.g., by using any type of existing modelling language, or formal mathematical descriptions. For an analysis, either a single or multiple conceptualisation bases and/or meta^2 models can be regarded. When combining these options, the following four strategies emerge as shown in Tab. 1.

	Single M^{2}M	**Multiple M^{2}M**
Single CB	*Variant Analysis*	*M^2 M Influence Analysis*
Multiple CB	*CB Influence Analysis*	*Mutual Influence Analysis*

Table 1: Possible analysis types based on the combination of conceptualisation bases (CB) and meta meta models (M^{2}M)

In case of both a single conceptualisation base and a single meta^2 model, we speak of a *variant analysis*. This means that different variants of using one specific conceptualisation base and one specific meta^2 model are analysed. To illustrate this with a simple example, we can revert to the well known description of entity relationship models as a conceptualisation base (Chen 1976) and analyse different variants by building upon on the ADOxx meta^2 model. Thereby particular decisions during the conceptualisation process become apparent, such as that in one variant also according transformation algorithms for deriving relational database schemata have been added, whereas in another variant only a visual modelling editor is provided. This in turn may affect, how the corresponding meta models are specified. Whereas for a variant that uses algorithms, the cardinality assignments have to be expressed in a machine processable format, e.g., using predefined attribute values, for a variant that focuses only on a visual modelling editor this information may also be specified by free text. If there is one conceptualisation base and multiple meta^2 models, we denote this as a *meta^2 model influence analysis*. In this case it is analysed, how different meta^2 models influence the conceptualisation of one specific modelling method. Following the example from above, we can analyse a conceptualisation of entity relationship models using the ADOxx meta^2 model and one using EMF. This would permit for example to compare the effort of using different meta^2 models as well as the properties of the resulting tools. In the third case there is one meta^2 model and multiple conceptualisation bases. This is denoted as a *conceptualisation base influence analysis*. In this way the focus is set on how different conceptualisation bases are transformed into modelling methods by using one particular meta^2 model. This means that we compare for example the conceptualisation of entity relationship models and the conceptualisation of BPMN models by using the ADOxx meta^2 model in both cases. As a a result, we can compare how the functionalities provided by ADOxx influence the conceptualisation of different modelling methods. Finally, when there are multiple conceptualisation bases and multiple meta^2 models, we denote this as *mutual influence analysis*. Thereby, varieties of different conceptualisation bases and different meta^2 models are analysed. As an example, a conceptualisation of BPMN models using EMF and a conceptualisation of ADONIS business process models using ADOxx could be compared. In this way, insights on using the resulting tools for a specific purpose, e.g., business process management, may be gained.

For our investigation we will take the approach of the *conceptualisation base influence analysis*. Therefore we will set the meta^2 model to the

ADOxx meta^2 model and refer to different conceptualisation bases. With this direction, our goal is to analyse how four functionalities that are common to the conceptualisation of modelling methods are realised using the ADOxx approach. These are derived from common requirements concerning enterprise information systems in terms of user interaction, interfaces to other systems for integration purposes, optimisation and the analysis of systems (cf. Frank and Strecker 2009). The functionalities we thus selected are: the *visualisation of models* for user interaction aspects, the *transformation* of models to specific data formats for interface aspects, the *simulation* of process-based structures for optimisation aspects, and the *querying* of model content for analysis aspects. Next, we chose for each functionality one modelling method from the Open Models Initiative that makes particular use of the functionality, which led to the cases we will describe in the following.

4.1 Visualisation

The graphical representation of models is obviously an integral part when dealing with visual modelling languages. By using the term 'visualisation', we refer to the definition given in (Fill 2009)[p. 19] as "the use of graphical representations to amplify human cognition". Thereby, the pragmatic aspect, i.e., the relation between graphical representations and human interpreters, is stressed which contributes significantly to the effectiveness of the representations and goes beyond the purely notational aspects (Gurr 1999).

For investigating the role of visualisation in the conceptualisation of modelling methods, we can distinguish between two cases. In the first case, the conceptualisation base for the modelling method does not provide any information about graphical representations. It then becomes the task of the method engineer to develop an adequate visualisation for the elements of the syntax of the modelling language by taking into account the assigned semantics. For this purpose it can be reverted to a number of sources in the literature that discuss the design of visual languages (e.g., Costagliola et al. 2002; Marriott and Meyer 1998) and in particular the construction of visual notations (e.g., Gabriel and Gluchowski 1998; Moody 2009; Moody et al. 2009). From a technical perspective also existing repositories and services of notations or notation fragments such as provided by the OMI[5] can be used (Fill and Karagiannis 2006).

In the second case, a specific visualisation is already defined in or directly related to the conceptualisation base. Then the focus lies on the exact representation of the visualisation by using the given constructs of the chosen meta^2 model. The degree of freedom for adapting the visualisation is in this case rather limited. However, as we will show, there are some aspects where the visualisation has to be adapted during the conceptualisation. This is the case particularly for dynamic notations as described in Sect 2.1 which are used by many modelling methods but are typically not part of discussions on notations as in (Moody 2009).

To show in detail how visualisation functionalities are realised in ADOxx we will take up the example of the iStar (i*) modelling method in the following (Schwab et al. 2010). Very briefly described, iStar is a framework that defines two types of models (Yu et al. 2001): strategic dependency models and strategic rationale models. With these types of models relationships among actors and their goals, tasks and resources as well as the reasoning of each actor about the relationships with other actors can be represented and analysed. For the conceptualisation of the modelling method on ADOxx, the two types of models were represented as views on the common model type *intentional actors and elements model*: the view *strategic rationales* and the view *strategic dependencies* – see Fig. 5. The view functionality of the ADOxx modelling component thus permits to automatically switch between two different visualisations of one model type. This was possible

[5] See http://openmodels.at/web/omi/services

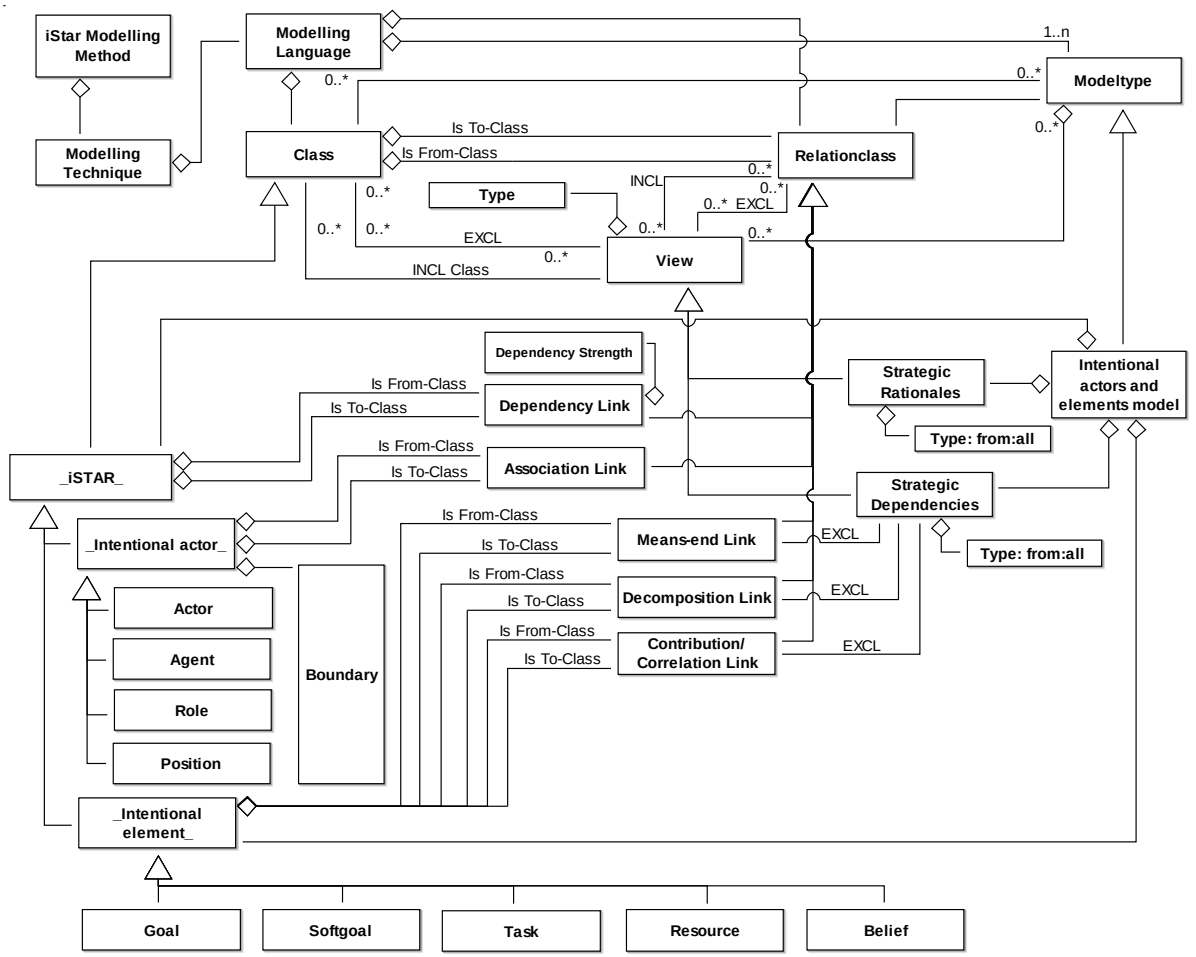

Figure 5: Excerpt of the iStar modelling method showing the intentional actors and elements model type, its classes, relationclasses, and attributes that are relevant for the specification of the visual notation

because the classes and relationclasses that were required for the two types of models overlap. In more detail, the strategic dependencies view excludes the three relationclasses *means-end link*, *decomposition link*, and *contribution/correlation link*, whereas the strategic rationales view contains all classes and relationclasses[6].

In regard to the visual notation of the classes and relationclasses, iStar has a very distinct set of symbols that are common to most literature sources. Although suggestions have been made to alter the representation for adding semiotic clarity (Moody et al. 2009), it is still the most used and understandable representation. Therefore, the GRAPHREP grammar of ADOxx was used to match the shapes and colours of the iStar standard symbols for classes as shown in Fig. 6. Concerning the aspects of dynamic notation, this standard representation was extended with control structures to take into account different attribute states. An example of this is shown in Fig. 7: on the right an excerpt of the GRAPHREP code for the dependency link relationclass is given. In this code segment an AVAL statement retrieves the current value of the attribute *dependency strength* and assigns it to the variable 's'. This variable is then used in an IF-

[6]Note: in Fig. 5 abstract classes that can only be instantiated via one of its subclasses are represented by leading and trailing underscores.

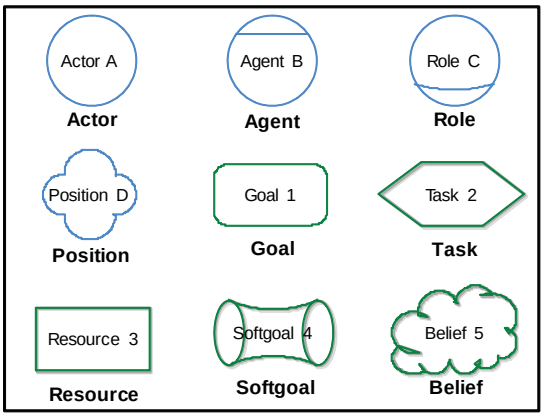

Figure 6: Visual notation for iStar classes

control statement to distingish between the attribute states 'committed' and 'open'. Accordingly, two different visualisations are shown on the left, where in the case of the state 'open' the letter 'O' is added at the start of the relation. The original iStar notation is thus enhanced to improve the user interaction. By using the dynamic notation attribute states are now directly visible and do not have to be specifically retrieved.

4.2 Transformation

The transformation of model contents to specific data formats is often needed to exchange models between different tools, feed the information contained in models to other systems, or create reports about model contents via document formats. As discussed previously, ADOxx provides a generic, ADOxx meta[2] model based XML import and export that permits to serialise and deserialise any model that is based on an ADOxx meta model to a generic XML format. This format comprises constructs such as *instances* that correspond to instances of the meta model classes, *connectors* that correspond to instances of the meta model relationclasses and *attributes* for the instances and connectors.

Although this generic XML format can be easily transformed to other formats, e.g., as discussed in the field of business process modelling (Mendling et al. 2004), or act as a basis for the generation of document formats such as HTML or DOCX via XSL formatting objects[7], it has to be ensured during the conceptualisation of a modelling method that the information required for generating a particular format is available in the meta model. Subsequently, either the import/export component can be used for the generic XML and ADL formats or, in case a specific format should be generated directly, according customisations of the transformation via the external coupling component have to be implemented to establish a mapping.

As an example for a modelling method that specifically focuses on such transformation functionalities, we selected the eduWeaver modelling method (Bajnai et al. 2005). eduWeaver is a modelling method that originated from an Austrian research project in the area of e-learning. It supports the representation of courses, modules, and lectures in a process-oriented style that is complemented with learning objects – see Fig. 10. Learning objects stand for any type of teaching material such as documents or multimedia objects and contain a reference to the corresponding files. By using this modelling method, teachers can design their courses on a conceptual level, link them to their teaching material and export all information in standardised e-learning content management formats. These formats can then be imported in an e-learning platform. This allows the re-use of the course content and structure for different platforms as well as the easy re-arrangement of course sections.

Regarding the transformation in the context of eduWeaver, one export format is the IMS content packaging format (IMS 2001). This format is based on the information model shown in Fig. 8 and consists of a package interchange file in zip-format that contains an XML manifest file and the physical files for the actual content. For realising this transformation for the eduWeaver modelling method, a specific algorithm had to

[7] See `http://www.w3.org/standards/xml/publishing` (accessed 05-11-2012) for further information on using XSL formatting objects.

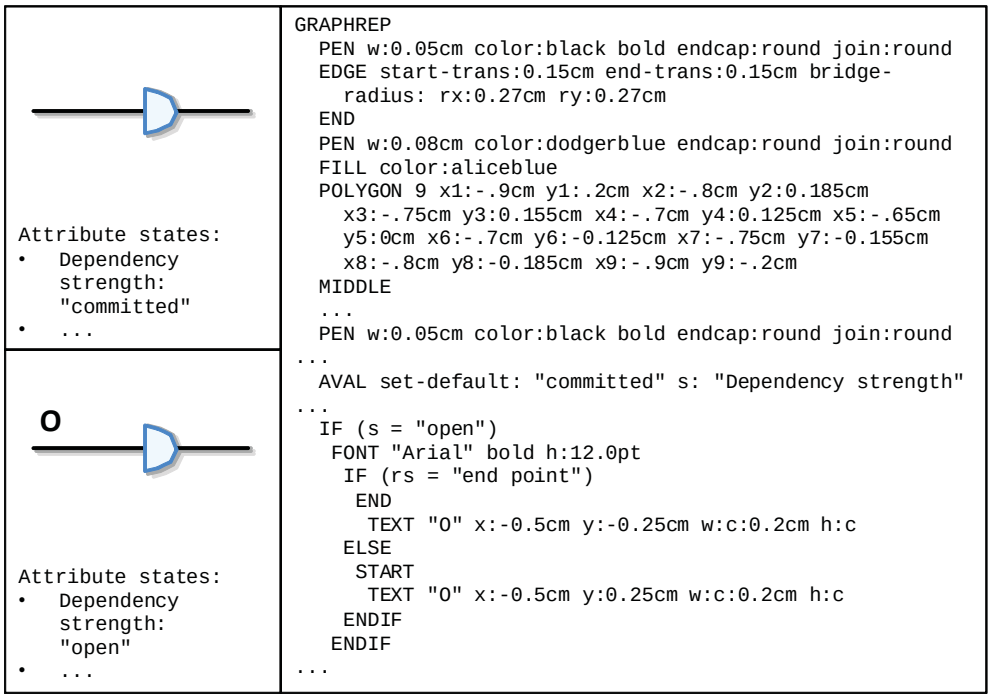

Figure 7: GRAPHREP definition for iStar dependency link relationclass

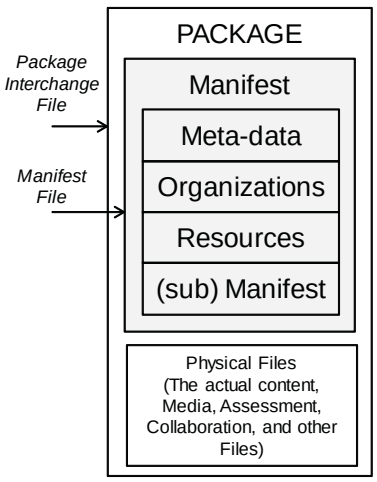

Figure 8: IMS content packaging scope, source: IMS 2001

be created that accesses the information in the eduWeaver model types and retrieves in particular the physical files from the learning object pool. For this purpose the external coupling component of ADOxx was used. This component supports the use of the ADOxx specific scripting language ADOscript that permits to access and process information stored in models. In addition, it also features a native XML interface for creating arbitrary XML formats and can be linked to external programs for influencing their behaviour. For eduWeaver, an ADOscript procedure for creating the IMS manifest was designed and used in the IMS export algorithm together with calls to an external zip packaging application for creating the package interchange file.

In this way the model information necessary for the IMS manifest file was retrieved. In particular this concerns the structure of the course, modules and lectures and the attached learning objects together with their meta data such as the assigned keywords. Additionally, the physical files referenced in the attributes *CI resources*, i.e., learning *c*ontent related *i*nformation, *PI resources*, i.e., *p*reparation related *i*nformation for exams, and *AI resources*, i.e., *a*ssessment related *i*nformation for self-assessments, of the learning objects were copied to the same directory as the manifest file and then compressed using the zip application – see Fig. 9 for an excerpt of the

```
...
# write header of xml file
CC "Documentation" XML_WRITEPLAIN
  "<?xml version=\"1.0\" encoding=\"iso-
  8859-1\"?>\n\n"

# write IMSManifest for selected course
CC "AdoScript" MSGWIN "IMS manifest is
    created and files are copied. Please
    wait..."

WRITE_IMSMANIFEST (kursid)
                  objid:(kursobjid)

# close IMSManifest.xml
CC "Documentation" XML_CLOSE write

# zip IMS-directory using fbzip.exe
CC "AdoScript" MSGWIN "IMS directory is
   compressed. Please wait…"
SET zipcmd:("zip\\fbzip.exe -a -p -r
    \""+path+"\\"+rootname+".zip\"
    \""+imsroot+"\"")
SET zipcmd:(replall (zipcmd, "\\\\",
     "\\"))
...
```

Figure 9: Excerpt of the eduWeaver IMS export algorithm in ADOscript

algorithm in ADOscript.

4.3 Simulation

In the course of the optimisation of enterprise information systems, the simulation of prospective structures for an a-priori evaluation of their performance is a well recognised technique (Mielke 1999). The provision of models is thereby the first step to successfully study not only the structural system aspects but also the dynamic behaviour of the systems and their interaction with the environment (Oberweis and Sander 1996). In the field of business process management, simulation has not only been used to analyse and enhance the efficiency of processes but also to assess quantitative requirements in terms of human and material resources. For the simulation of business processes by a process simulation engine, it is necessary to supply additional data besides the purely structural description of the processes. This concerns descriptions of the flow semantics of the underlying process modelling language as well as detailed information on simulation parameters such as process quantities and time and transition probability properties. Depending on the type of simulation even more data may have to be added, e.g., when conducting simulations for planning and scheduling personnel capacities or dynamic evaluations of the workload of an organisation (Herbst et al. 1997). The acquisition of such simulation data can thereby either be added to existing models or the models themselves may be acquired through workflow mining and machine learning techniques as it has been initially described by Herbst and Karagiannis (Herbst and Karagiannis 1998).

The simulation component of the ADOxx platform contains four simulation algorithms that range from simple path analyses of business process models to complex process and organisational evaluations based on queuing networks (Herbst et al. 1997; Kühn and Junginger 1999)[8]. They are based on object-oriented discrete-event simulation, whose flow semantics can be described using labelled Petri nets (Herbst 2001). The algorithms are specified in the form of hybrid algorithms that can be applied to arbitrary types of process modelling languages. The links between the algorithms and the modelling language are established by sub-typing abstract classes of the ADOxx Meta Model that has been discussed in Sect 2.2. In this way the information from the pre-defined simulation classes is made available in the classes of the user-defined meta model. The simulation algorithms can then retrieve this information and execute the models.

In the past, the ADOxx simulation algorithms have been applied to several process modelling languages including UML activity diagrams (Kühn and Junginger 1999) and most recently BPMN 2.0 in the Adonis CE edition[9], too. Due to the requirement of sub-typing specific classes of the ADOxx Meta Model to enable the application of

[8] A detailed discussion of these algorithms can be found in the user manual of ADONIS as the predecessor of ADOxx (BOC GmbH 1999, pages 309ff.)

[9] Adonis is a commercial product and trademark of BOC AG. The free community edition is available at `http://www.adonis-community.com/`.

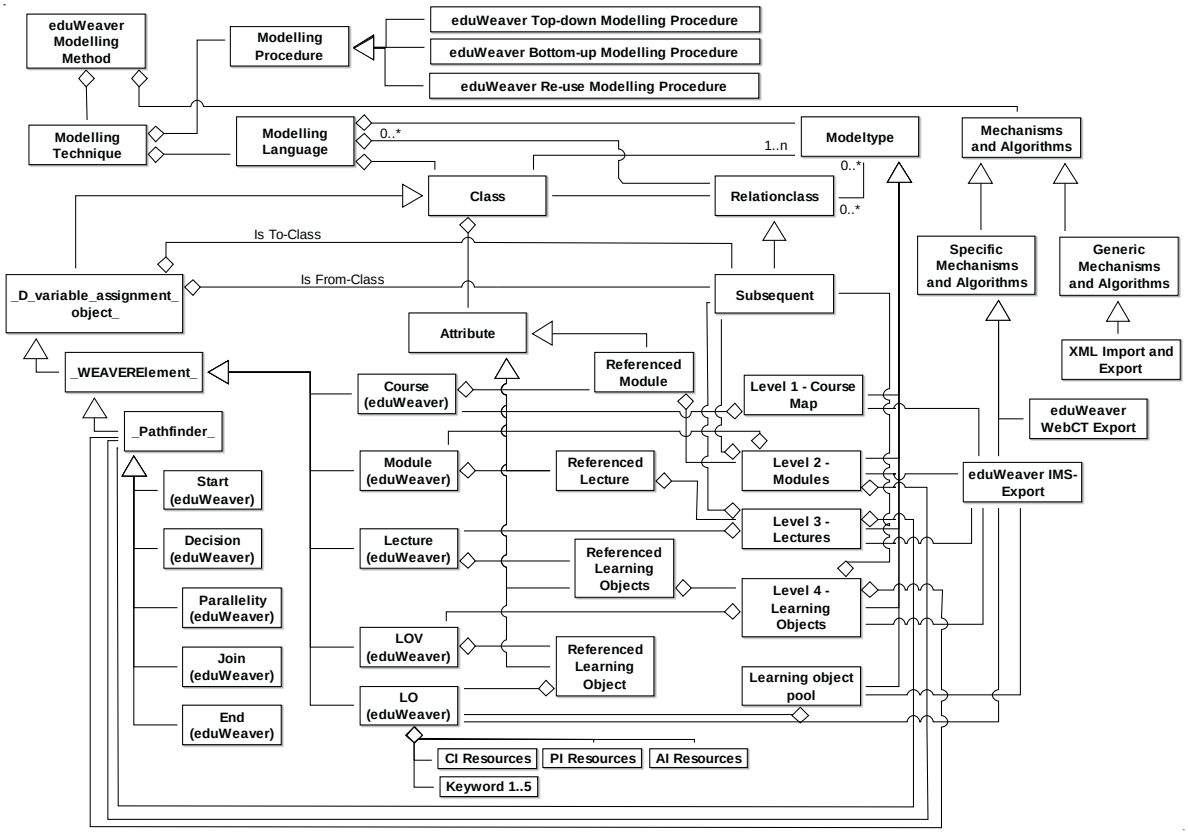

Figure 10: Excerpt of the eduWeaver modelling method showing the model types, classes, relationclasses, and attributes that are relevant for the IMS export algorithm

the simulation algorithms, it already has to be decided when designing a modelling method, if any of the algorithms should be used later.

To illustrate these aspects we have selected the *BPMS modelling method* and the application of the *path analysis* simulation algorithm. BPMS is the modelling method underlying the Adonis business process management toolkit and has a long history of successful applications in business and research projects (Junginger et al. 2000). The excerpt of the modelling method that highlights the corresponding definitions is shown in Fig. 11. On the right hand side the path analysis simulation algorithm is defined as a sub-type of hybrid mechanisms and algorithms. The result of this type of simulation algorithm is detailed information about the average time and cost of a business process including the frequency of occurrence of the single paths. For this purpose the algorithm requires as input a process structure based on the abstract classes shown on the left hand side of Fig. 11. Thus, the elements of the model type *business process model* 'Trigger', 'Process start', 'Subprocess', 'Activity', 'Decision', 'Parallelity', 'Merging', and 'End' are defined as sub-classes of the abstract classes '_Neutral element_(Metamodel)', '_Process start_(Metamodel)', '_Subprocess_(Metamodel)', '_Activity_(Metamodel)', '_Decision_(Metamodel)', '_Merging_(Metamodel)', and '_End_(Metamodel)'. Through this sub-typing they are assigned the flow semantics attached to the abstract classes. For example, the abstract class '_Decision_(Metamodel)' implies that it has to be chosen from any outgoing flows of this element exclusively, i.e., in the sense of an exclusive disjunction or XOR

operator. Similarly, the abstract class '_Subprocess_(Metamodel)' enables a hierarchical decomposition of process models and the re-use of process fragments. In the same way also attributes that are attached to the abstract classes are made available in their subclasses[10]. For example the abstract class '_Activity_(Metamodel)' provides a number of time related attributes such as 'Execution time', 'Waiting time', 'Resting time' and 'Transport time' that are also required by the path analysis algorithm.

To define the flow between process elements, two more aspects need to be taken into account. The first is the definition of transition conditions for exclusive paths and the second is the attachment of stochastic distributions to the variables used in the transition conditions. For the transition conditions, the ADOxx Meta Model provides the relationclass 'Subsequent' together with an according attribute. The value of this attribute has to conform to a particular expression grammar that permits to reference the state of variables. These variables are defined using subclasses of the abstract classes '_Variable_(Metamodel)' and '_Random generator_(Metamodel)'. Random generators define the stochastic distributions of the variables and are linked to the variable classes using the 'Sets Variable' relationclass and to the elements in the process flow using the 'Sets' relationclass, which are also both part of the ADOxx Meta Model. With all these definitions in place, the path analysis algorithm can then successfully retrieve the information on the process structure and its attributes that now corresponds to the requirements of the flow semantics[11].

4.4 Querying

One important function of enterprise information systems is their support for analyses of business operations. These can be conducted on any level, ranging from strategic decisions, the design and re-engineering of business processes and organisational structures, the allocation of resources, to the execution of workflows and the evaluation of performance (Karagiannis 1995). To successfully support decision makers, the data for such analyses may also need to be aggregated for specific scenarios, e.g., as it is typically addressed by systems such as data warehouses or online analytical processing. With the enormous amounts of models that are today maintained in the repositories of some large organisations (Rosemann 2006), similar concepts are also required for enterprise models. In particular, the combination of models that are generated automatically based on existing data, e.g., in the area of IT infrastructure management (Moser and Bayer 2005), and models that are manually created for the explication of knowledge, requires sophisticated analysis mechanisms. The basis for such analyses are query languages that permit the formulation of complex data retrieval operations in an intuitive way, which abstracts from the underlying technical implementation. For this purpose the analysis component of the ADOxx platform provides the AQL query language that allows to formulate queries on models in a style similar to SQL[12]. The AQL queries can either be entered manually by a user or can be pre-defined. In the first case ADOxx provides a dialogue-based assistant for composing the queries, whereas in the latter case the method engineer defines the queries together with a query-specific user interface for entering variables. The advantage of such pre-defined queries is that the user who wishes to execute a pre-defined query only has to fill in values in text fields or select from pre-defined values and does not need to know about the actual composition of AQL queries. During the conceptualisation of a modelling method that requires the use of queries, e.g., for certain steps in the modelling procedure, the method engineer thus has to provide this information.

[10]Note: The attributes of the abstract classes are not shown in Fig. 11 due to space limitations.

[11]For the details on the working of the algorithm we refer the interested reader again to (BOC GmbH 1999).

[12]For the details of the AQL language including its grammar we refer to (BOC GmbH 1999, pages 695ff.).

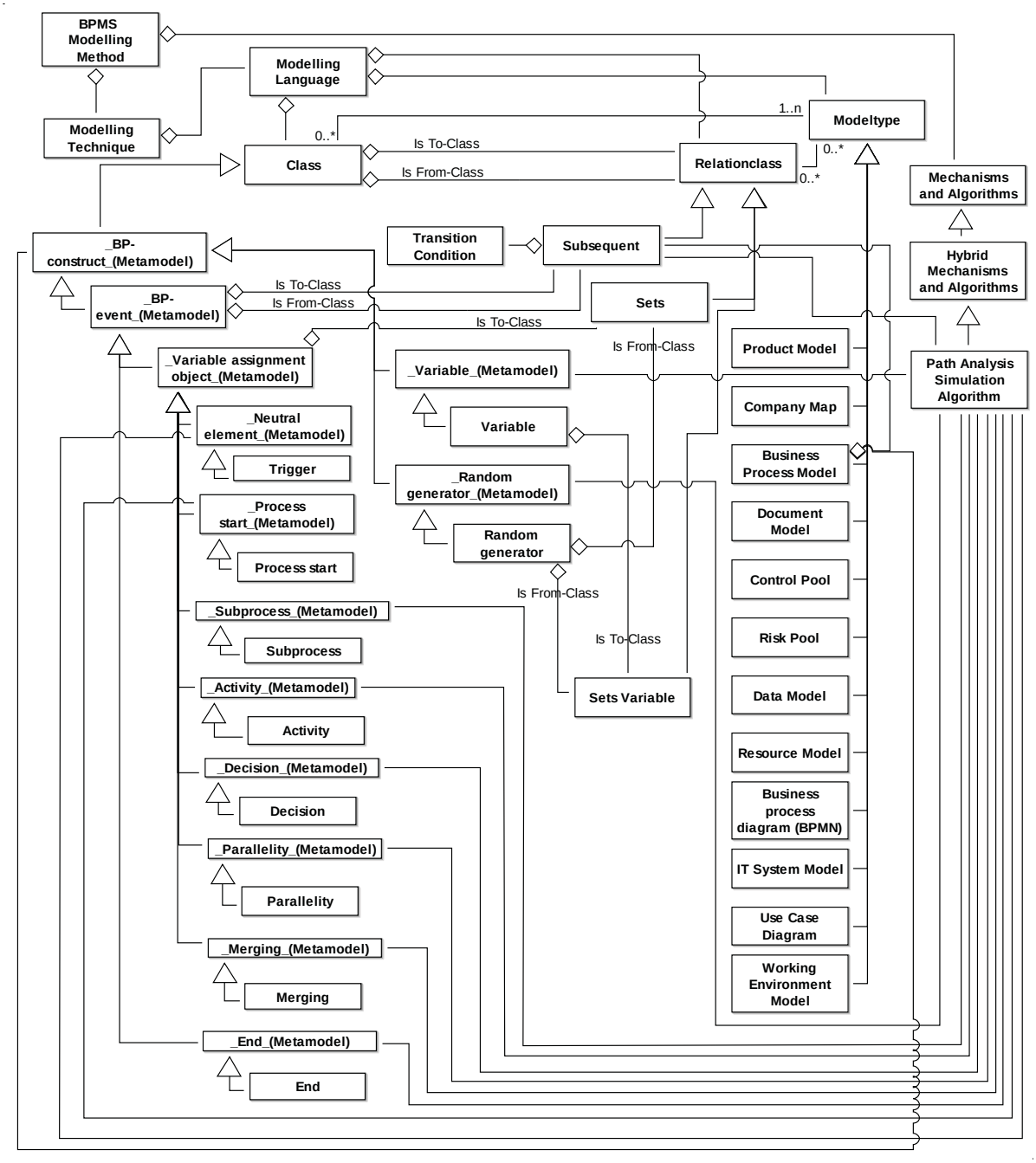

Figure 11: Excerpt of the BPMS modelling method showing the model types, the classes and relationclasses of the business process model type and how they are derived from the abstract classes of the ADOxx meta model that are targeted by the path analysis simulation algorithm

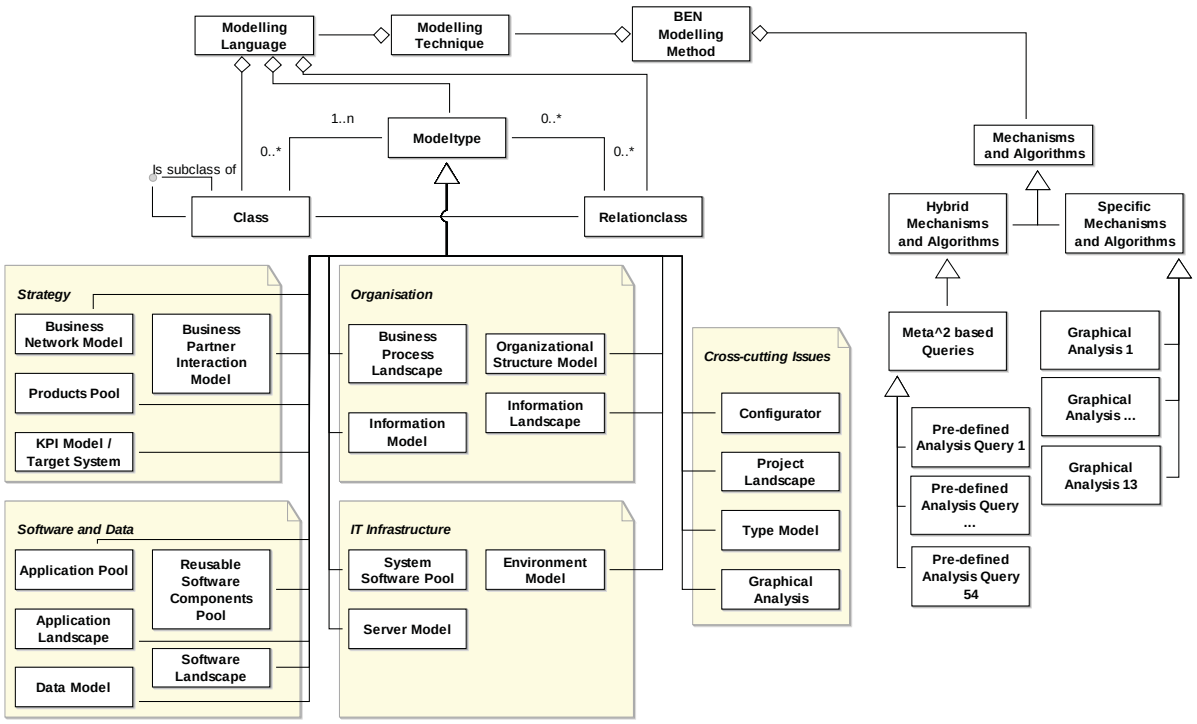

Figure 12: Excerpt of the BEN modelling method showing only the model types and the hierarchy of mechanisms and algorithms

For illustrating these aspects we will revert in the following to the Business Engineering Navigator (BEN) modelling method (Aier et al. 2009). BEN is based upon the concepts of business engineering that focuses on the model-based conceptualisation, design, and implementation of transformations in businesses (Winter 2001). Thereby, BEN puts special emphasis on the domain of enterprise architecture management (EAM) to achieve a holistic business-to-IT transformation approach. By using the BEN method, the relationships between a company's strategy, organisation, software and data, and its IT infrastructure can be represented and analysed. This is also reflected by the model types used in BEN as shown in Fig. 12. For the representation of cross-cutting issues additional model types are defined to enable functionalities such as specific graphical analyses. To support the analysis of complex business-IT relationships, BEN provides a set of 13 specific algorithms for conducting graphical analyses and 54 pre-defined analysis queries. To show in detail how these queries are specified, we will use the following query as an example: "Show all servers that have a specific type of system software at a specific patch level installed". As a result we would like to retrieve the server attributes such as its ID, name, and description. The model types affected by this query are the *system software pool* and the *server model* in the context of IT infrastructure, with the server model being the primary model type. When accessing the pre-defined query, a user should have the option to select the specific type of system software in the form of a selection box that contains all instances of a system software. In addition, a user should be able to select from the available patch levels. To define an according user interface, the definitions shown in the upper part of Fig. 13 have to be given. The attribute value field and the edit field are thereby assigned the references '@R@@2@' and '@R@@4@'. By referring to these references, the AQL query can be formulated subsequently, together with the

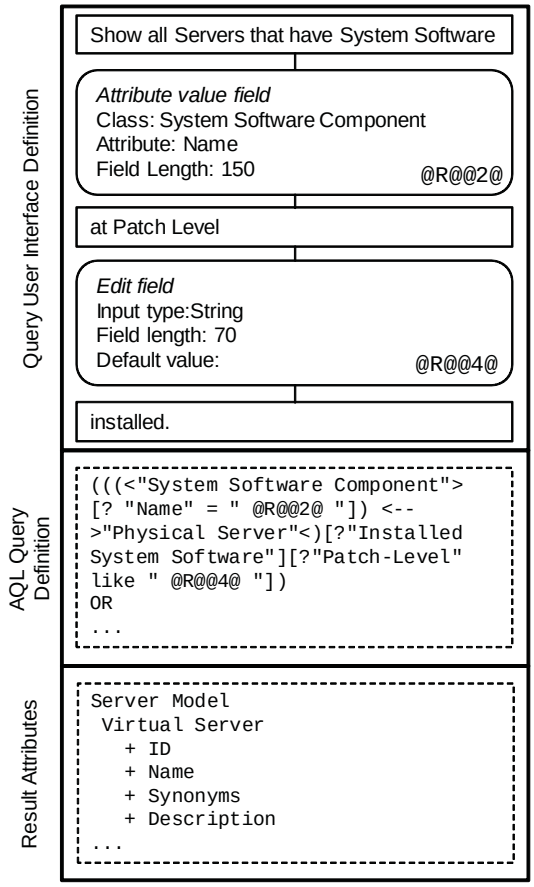

Figure 13: Excerpt of the conception of a pre-defined analysis query

required result attributes. When a user accesses this pre-defined query, at first a server model has to be selected. Next, a selection box for all types of system software component elements that are referenced by this server model are shown. After the selection of one or more system software components the available patch levels for these components are shown, again as a selection box. Finally, the result of the query is presented in a table.

5 Discussion

For the analysis of the conceptualisation of modelling methods we have presented four cases that illustrate the use of components of the ADOxx platform. It could be shown how user interaction aspects are considered for the definition of visualisations with a given graphical notation; how interface aspects to other systems have to be taken into account for designing transformation algorithms based on model data; how the requirement of using process-based simulation functionalities affects the way of designing meta model classes and relationclasses; and finally how complex querying functionalities that are required for modelling procedures can be realised in a user-friendly way by pre-defined queries.

With these selected examples it can already be derived, how the interdependencies between: a. the constructs of a modelling language, b. the requirements of the modelling procedure, c. the capabilities and requirements of mechanisms and algorithms and d. the components of the underlying technical platform have to be taken into account simultaneously during the design of a modelling method. Thereby, a particularly important relation are the interdependencies between a., c., and d., i.e., when mechanisms and algorithms are required for a modelling method. Although for many cases it may also be sufficient to defer the design of algorithms to a later stage during the conceptualisation, the application of complex processing functionalities, such as simulation algorithms, can not be easily separated from the design of the modelling language due to very specific data requirements. This also applies in part to the use of querying functionalities, whose optimal configuration depends on the constructs available in the modelling language. In regard to the provision of specific interchange formats for transferring the content of models to external applications and platforms, the easiest way to integrate this functionality in a modelling method is to build upon the generic import and export component. By mapping and transforming the thus retrieved generic data structures to the required formats, it can also be reacted quickly to potential changes in the target formats. However, when either from the side of the user interaction or due to the complexity of the target format this procedure is not sufficient – as it was, e.g.,

the case for the eduWeaver modelling method that required also physical files to be included in a .zip archive – a specific external coupling needs to be established for such purposes. Although the visualisation aspects concerning the graphical representation of the classes and relationclasses could be treated independently in the case shown for iStar, this may not be true for other cases. For example, the graphical analysis algorithms of the BEN modelling method even require the definition of a distinct model type to realise more complex visualisation requirements. Additionally, also the optimal support of a modelling procedure may lead to further requirements in terms of dynamic notations, e.g., to enable the visual checking of constraints.

6 Conclusion and Outlook

The chosen type of the conceptualisation base influence analysis proved to be a suitable way for gaining insights into the conceptualisation of modelling methods. Based on the components of the ADOxx meta modelling platform it was possible to investigate how visualisation, transformation, simulation, and querying functionalities are realised and how they impact the design of a modelling method. For the future it is planned to extend this investigation to more functionalities, e.g., specific mechanisms for supporting modelling procedures, constraint checking facilities, the integration of web-based and semantic technologies or the bi-directional communication with external systems. Furthermore, a future task will be to support the conceptualisation of modelling methods in a way that it can be handled also by domain experts with no or little technical knowledge. This concerns in particular the composition of fragments of existing modelling methods in the sense of hybrid modelling (Karagiannis 2012) and method integration (Kühn 2004). Thereby, for example, a part of a meta model and an algorithm from a method A shall be combined with another meta model and algorithms from a method B. Although these compositions can be fully realised from a technical point of view, it still requires in-depth technical knowledge to conduct these alignments. Therefore, further research is needed to abstract from the technical details while as the same time enabling the full functionality for composing modelling methods. Additionally, also the specification of visualisations for modelling methods can be further simplified. Possible approaches in this direction are the re-use of existing visualisation patterns, e.g., as proposed in the context of semantic visualisation (Fill 2009), or the use of notation repositories as had been discussed in Sect 4.1.

Acknowledgement

The authors would like to thank Harald Kühn for his valuable feedback during the preparation of this article.

References

Aier S., Kurpjuweit S., Saat J., Winter R. (2009) Business Engineering Navigator – A Business to IT Approach to Enterprise Architecture Management In: Coherency Management – Architecting the Enterprise for Alignment, Agility, and Assurance Bernard S., Doucet G., Gotze J., Saha P. (eds.) Bloomington, pp. 77–98

Amelunxen C., Koenigs A., Roetschke T., Schuerr A. (2007) Metamodeling with MOFLON. In: AGTIVE 2007. Springer

Aniszczyk C. (2006) Learn Eclipse GMF in 15 minutes - Get started with model-driven development the Eclipse way. Last Access: http://www.ibm.com/developerworks/opensource/library/os-ecl-gmf/ accessed 07-11-2012

BOC GmbH (1999) ADONIS Version 3.0 Band II - Benutzerhandbuch (English: ADONIS Version 3.0 Volume II - User Handbook). BOC GmbH

Bajnai J., Karagiannis D., Steinberger C. (2005) eduWeaver – the Courseware Modeling Tool. In: Akoka, J. et al. (ed.) Perspectives in Conceptual Modeling, ER 2005 Workshops. Springer

Brinkkemper S., Saeki M., Harmsen F. (1999) Meta-modelling based assembly techniques for situational method engineering. In: Information Systems 24(3), pp. 209–228

Case A. (1985) Computer-aided software engineering (CASE): Technology for improving software development productivity. In: Data Base (Fall 1985), pp. 35–43

Chen P. P.-S. (1976) The Entity-Relationship Model-Toward a Unified View of Data. In: ACM Transactions on Database Systems 1(1), pp. 9–36

Costagliola G., Delucia A., Orefice S., Polese G. (2002) A Classification Framework to Support the Design of Visual Languages. In: Journal of Visual Languages and Computing 13, pp. 573–600

Emerson M., Sztipanovits J. (2006) Techniques for Metamodel Composition. In: OOPSLA – 6th Workshop on Domain Specific Modeling, pp. 123–139

Fill H.-G. (2009) Visualisation for Semantic Information Systems. Gabler

Fill H.-G., Karagiannis D. (2006) Semantic Visualization for Business Process Models In: Twelth International Conference on Distributed Multimedia Systems – International Workshop on Visual Languages and Computing 2006 Knowledge Systems Institute, Grand Canyon, USA, pp. 168–173

Fill H.-G., Redmond T., Karagiannis D. (2012) FDMM: A Formalism for Describing ADOxx Meta Models and Models In: Proceedings of ICEIS 2012 - 14th International Conference on Enterprise Information Systems, Wroclaw, Poland Maciaszek L., Cuzzocrea A., Cordeiro J. (eds.)

Frank U. (2011) Some Guidelines for the Conception of Domain-Specific Modelling Languages. In: Nüttgens M., Thomas O., Weber B. (eds.) EMISA 2011 Vol. P-190. GI, pp. 93–106

Frank U., Strecker S. (2009) Beyond ERP Systems:An Outline of Self-Referential Enterprise Systems. In: ICB-Research Report – Universität Duisburg Essen 31

Gabriel R., Gluchowski P. (1998) Grafische Notationen für die semantische Modellierung multidimensionaler Datenstrukturen in Management Support Systemen (German). In: Wirtschaftsinformatik 40(6), pp. 493–502

Gurr C. (1999) Effective Diagrammatic Communication: Syntactic, Semantic and Pragmatic Issues. In: Journal of Visual Languages and Computing 10, pp. 317–342

Harel D., Rumpe B. (2000) Modeling Languages: Syntax, Semantics and All That Stuff – Part I: The Basic Stuff. MCS00-16. The Weizmann Institute of Science

Harel D., Rumpe B. (2004) Meaningful Modeling: What's the Semantics of Semantics? In: IEEE Computer October 2004, pp. 64–72

Harmsen F., Saeki M. (1996) Comparison of four Method Engineering languages In: Method Engineering: Principles of method construction and tool support Brinkkemper S., Lyytinen K., Welke R. (eds.) Chapman and Hall, pp. 209–231

Herbst J. (2001) Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen (German). PhD thesis

Herbst J., Karagiannis D. (1998) Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. In: Proceedings of the 9th International Workshop on Database and Expert Systems Applications. IEEE, pp. 745–752

Herbst J., Junginger S., Kühn H. (1997) Simulation in Financial Services with the Business Process Management System ADONIS In: 9th European Simulation Symposium (ESS97) Society for Computer Simulation

IMS G. (2001) IMS Content Packaging Information Model – Version 1.1.2 Final Specification. Last Access: http://www.imsglobal.org/content/packaging/cpv1p1p2/imscp_infov1p1p2.html, 27-05-2012

Junginger S., Kühn H., Strobl R., Karagiannis D. (2000) Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation – ADONIS: Konzeption und Anwendungen (German). In: Wirtschaftsinformatik 42(5), pp. 392–401

Karagiannis D. (1995) BPMS: Business Process Management Systems. In: SIGOIS Bulletin 16(1), pp. 10–13

Karagiannis D. (2012) Modelling Aspects for Next-Generation Enterprise Systems. FInES Workshop at Aalborg - May 9, 2012. Last Access: http://de.slideshare.net/FInESCluster/p3-2dimitris-karagiannisv2 accessed 12-11-2012

Karagiannis D., Kühn H. (2002) Metamodeling Platforms In: 3rd International Conference EC-Web 2002 – Dexa 2002 Bauknecht K., Min Tjoa A., Quirchmayr G. (eds.) LNCS2455 Springer, Aix-en-Provence, France, p. 182

Karagiannis D., Grossmann W., Höfferer P. (2008) Open Model Initiative – A Feasibility Study. Last Access: http://cms.dke.univie.ac.at/uploads/media/Open_Models_Feasibility_Study_SEPT_2008.pdf

Kelly S., Rossi M., Tolvanen J.-P. (2005) What is Needed in a MetaCASE Environment? In: Enterprise Modelling and Information Systems Architecture 1(1), pp. 25–35

Kern H., Hummel A., Kuehne S. (2011) Towards a Comparative Analysis of Meta-Metamodels. In: The 11th Workshop on Domain-Specific Modeling. http://www.dsmforum.org/events/DSM11/Papers/kern.pdf 05-01-2012

Koch S., Strecker S., Frank U. (2006) Conceptual Modelling as a New Entry in the Bazaar: The Open Model Approach In: Open Source Systems Vol. 203/2006 IFIP International Federation for Information Processing, pp. 9–20

Kolovos D. S., Rose L. M., Paige R., Polack F. (2009) Raising the Level of Abstraction in the Development of GMF-based Graphical Model Editors. In: MiSE'09. IEEE, pp. 13–19

Kühn H., Junginger S. (1999) An Approach to use UML for Business Process Modeling and Simulation in ADONIS In: 13th European Simulation Multiconference – Modeling and Simulation: A Tool for the Next Millenium Szczerbicka H. (ed.) Warsaw, Poland, pp. 634–639

Kurpjuweit S., Winter R. (2007) Viewpoint-based Meta Model Engineering. In: Reichert M., Strecker S., Turowski K. (eds.) 2nd International Workshop on Enterprise Modelling and Information Systems Architectures. Gesellschaft für Informatik

Kühn H. (2004) Methodenintegration im Business Engineering (English: Method Integration in Business Engineering). PhD thesis

Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason C., Nordstrom G., Sprinkle J., Volgyesi P. (2001) The Generic Modeling Environment. In: WISP'2001. IEEE

List B., Korherr B. (2006) An Evaluation of Conceptual Business Process Modelling Languages. In: SAC'06. ACM

Marriott K., Meyer B. (1998) Visual language theory. Springer, New York

McNeill K. (2008) Metamodeling with EMF: Generating concrete, reusable Java snippets. Last Access: http://www.ibm.com/developerworks/library/os-eclipse-emfmetamodel/index.html?S_TACT=105AGX44&S_CMP=EDU

Mendling J., Neumann G., Nüttgens M. (2004) A Comparison of XML Interchange Formats for Business Process Modelling. In: EMISA 2004. German Informatics Society, pp. 129–140

Mernik M., Heering J., Sloane A. (2005) When and how to develop domain-specific languages. In: ACM Computing Surveys 37(4), pp. 316–344

Mielke R. (1999) Applications for Enterprise Simulation. In: Farrington P., Nembhard H., Sturrock D., Evans G. (eds.) Winter Simulation Conference. IEEE

Moody D. (2009) The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. In: IEEE Transactions on Software Engineering 35(6), pp. 765ff

Moody D., Heymans P., Matulevicius R. (2009) Improving the Effectiveness of Visual Representations in Requirements Engineering: An Evaluation of i* Visual Syntax. In: 17th IEEE International Requirements Engineering Conference. IEEE, pp. 171–180

Moser C., Bayer F. (2005) IT Architecture Man-

agement: A Framework for IT Services In: Proceedings of the Workshop on Enterprise Modelling and Information Systems Architectures Desel J., Frank U. (eds.) Lecture Notes in Informatics – Gesellschaft für Informatik (GI), Klagenfurt, Austria

Oberweis A., Sander P. (1996) Information system behavior specification by high level Petri nets. In: ACM Transactions on Information Systems 14(4), pp. 380–420

Object Management Group OMG (2007) OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. Last Access: http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/ 01-03-2011

Object Management Group OMG (2011a) Business Process Model and Notation (BPMN) Version 2.0. Last Access: http://www.omg.org/spec/BPMN/2.0/PDF/ 01-03-2011

Object Management Group OMG (2011b) OMG Meta Object Facility (MOF) Core Specification Version 2.4.1. Last Access: http://www.omg.org/spec/MOF/2.4.1/PDF/

Odell J. (1996) A Primer to Method Engineering In: Method Engineering – Principles of method construction and support Brinkkemper S., Lyytinen K., Welke R. (eds.) Chapman and Hall, pp. 1–28

Paige R., Ostroff J., Brooke P. (2000) Principles for Modeling Language Design. In: Information and Software Technology 42(10), pp. 665–675

Ronaghi F. (2005) A Modeling Method for Integrated Performance Management. In: 16th International Workshop on Database and Expert Systems Applications (DEXA'05). IEEE, pp. 972–976

Rosemann M. (2006) Potential pitfalls of process modeling: part B. In: Business Process Management Journal 12(3), pp. 377–384

Rossi M., Ramesh B., Lyytinen K., Tolvanen J.-P. (2004) Managing Evolutionary Method Engineering by Method Rationale. In: Journal of the AIS 5(9), pp. 356–391

Schwab M., Karagiannis D., Bergmayr A. (2010) i* on ADOxx(R): A Case Study. In: Proceedings of the 4th International i* Workshop – iStar10 – CAiSE Workshop Proceedings. Springer, pp. 92–97

Sprinkle J., Rumpe B., Vangheluwe H., Karsai G. (2010) Metamodelling – State of the Art and Research Challenges In: MBEERTS Giese, H. et al. (ed.) Vol. LNCS 6100 Springer, pp. 57–76

Tolvanen J.-P., Rossi M. (2003) MetaEdit+: defining and using domain-specific modeling languages and code generators. In: OOPSLA'03 Companion of the 18th annual ACM SIGPLAN conference. ACM, pp. 92–93

Winter R. (2001) Working for e-Business – The Business Engineering Approach. In: International Journal of Business Studies 9(1), pp. 101–117

Xu T., Ma W., Liu L., Karagiannis D. (2010) Hybrid Modeling: Synthesizing Strategic Model and Business Processes in Active i*. In: International Enterprise Distributed Object Computing Conference Workshops. IEEE, pp. 345–354

Yu E., Liu L., Li Y. (2001) Modelling Strategic Actor Relationships to Support Intellectual Property Management. In: Conceptual Modeling – ER 2001. Springer

Hans-Georg Fill, Dimitris Karagiannis

University of Vienna,
Research Group Knowledge Engineering,
Währinger Strasse 29
1090 Vienna
Austria
{hgf | dk}@dke.univie.ac.at