SIG MoBIS
Special Interest Group
on Modelling Business Information Systems

SIG EMISA
Special Interest Group
on Design Methods for Information Systems

Enterprise Modelling and Information Systems Architectures

An International Electronic Journal

Volume 8 | No. 1 **March 2013**

Special Issue on Design, Implementation and Evaluation of Modelling Tools

German Informatics Society

Table of Contents

Editorial Preface

Modelling tools support modellers in constructing, documenting, and maintaining conceptual models and provide functionality for model management. Meta modelling tools supply method and language designers with an infrastructure for method and language implementation including means for implementing modelling tools (e.g., graphical editors). Both modelling and meta modelling tools play a prominent role in model-based software development, and gain in importance in the light of domain-specific modelling, model-based analyses and decision-making. Referring to the economics of modelling, modelling tools represent a prerequisite to the effective and efficient use of modelling methods and languages. In addition to that, modelling tools open a new perspective on sophisticated management support systems that enable advanced users not only to navigate to the conceptual models their analyses are based on but also to modify these models. In this respect, modelling tools are widely regarded as a complementary yet pertinent subject in conceptual modeling research. Yet, surprisingly few scientific publications have so far dealt with the design, implementation and evaluation of modelling tools. We therefore dedicated this special issue to the current state-of-the-art in modelling tool research. The Dagstuhl seminar on 'Open Models as a Foundation of Future Enterprise Systems' and the workshop series on 'Methodical Development of Modelling Tools'[1] served as starting points for getting the special issue underway. Four papers were finally accepted for publication as a result of a double-blind review process.

The articles in this special issue approach modelling tool research from differing yet complementing angles, and, thus, demonstrate the spectrum of work in this research area. Hans-Georg Fill and Dimitris Karagiannis take an in-depth look into the implementation of a modelling method using the ADOxx meta modelling tool. They describe the meta modelling infrastructure provided by ADOxx including the meta modelling language and essential concepts related to tool support. The paper highlights four essential aspects of tool support, i.e., visualisation of models, model transformations, model-based simulations, and queries on models. Florian Matthias, Christian Neubert, and Alexander W. Schneider present a modelling tool, termed 'Hybrid Wiki', to address essential barriers to communication and collaboration in Enterprise Architecture endeavours. The tool is based on the concept of a Wiki page with extensions for collaborative modelling. Results of a survey and an industry case study suggest that the tool lowers barriers to communicate and to collaborate. Tony Clark and Balbir Barn introduce an approach to enterprise architecture modelling called LEAP, and show how its modelling language and corresponding modelling tool address open issues in requirements modelling. Building on goal-oriented requirements engineering approaches such as KAOS, the LEAP architecture modelling language adds operational semantics to express behavioural and non-functional goals. The paper reports on the language specification, tool design and implementation, and describes how the operational semantics are implemented. Marco Kuhrmann, Georg Kalus, and Alexander Knapp discuss PDE, a meta modelling toolset for developing domain-specific languages (DSL), and present a stepwise procedure for DSL development based on PDE. The toolset facilitates language design by reconstructing DSL elements from prototypical models drawn by domain experts in a process described as 'instance modelling'. The principal ideas behind PDE are discussed and illustrated by examples. The research articles in this issue are complemented by a report on the workshop 'Security in Business Processes' provided by the workshops organizers, Rafael Accorsi and Raimundas Matulevičius.

[1] http://www.wi-inf.uni-due.de/MeDMoT2013

We would like to thank Jens Gulden for his comments on the call for papers and for providing us with insights into the modelling tool research community. Our thanks also go to the reviewers involved in this special issue and to the authors of all submissions. The readers of this special issue may gain inspiring insights into modelling tool research.

Ulrich Frank
Andreas Oberweis
Stefan Strecker

Guest Editors' contact information:

Prof. Dr. Ulrich Frank
Chair for Information Systems
and Enterprise Modelling
University of Duisburg-Essen
Universitätsstr. 9
45141 Essen
Germany
ulrich.frank@uni-duisburg-essen.de

Prof. Dr. Andreas Oberweis
Chair for Business Information Systems
Karlsruhe Institute of Technology
Kaiserstraße 89
76133 Karlsruhe
Germany
oberweis@kit.edu

Prof. Dr. Stefan Strecker
Chair of Information Systems Development
University of Hagen
Profilstr. 8
58084 Hagen
Germany
stefan.strecker@fernuni-hagen.de

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

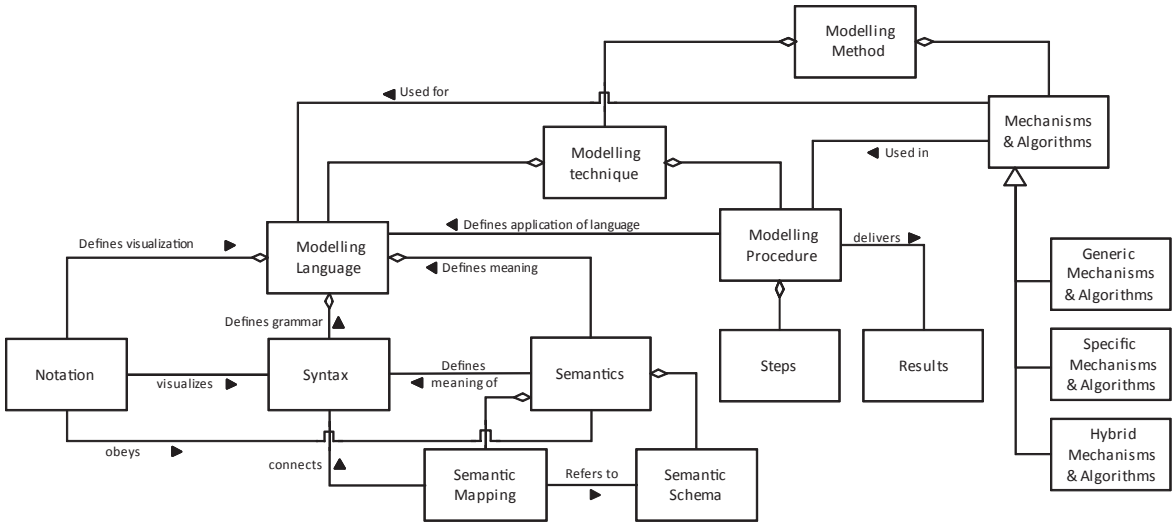

Figure 1: Components of modelling methods (Karagiannis and Kühn 2002)

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

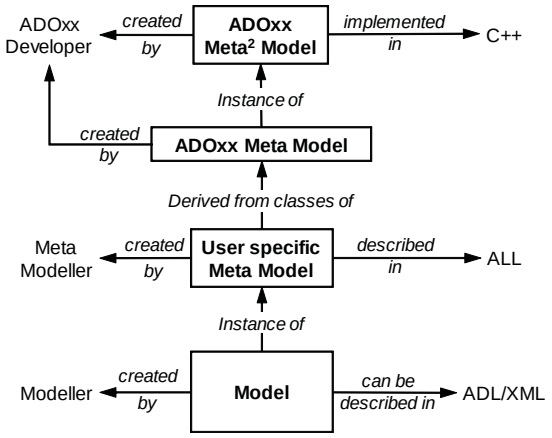

Figure 2: Roles and languages in the modelling hierarchy of ADOxx

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.

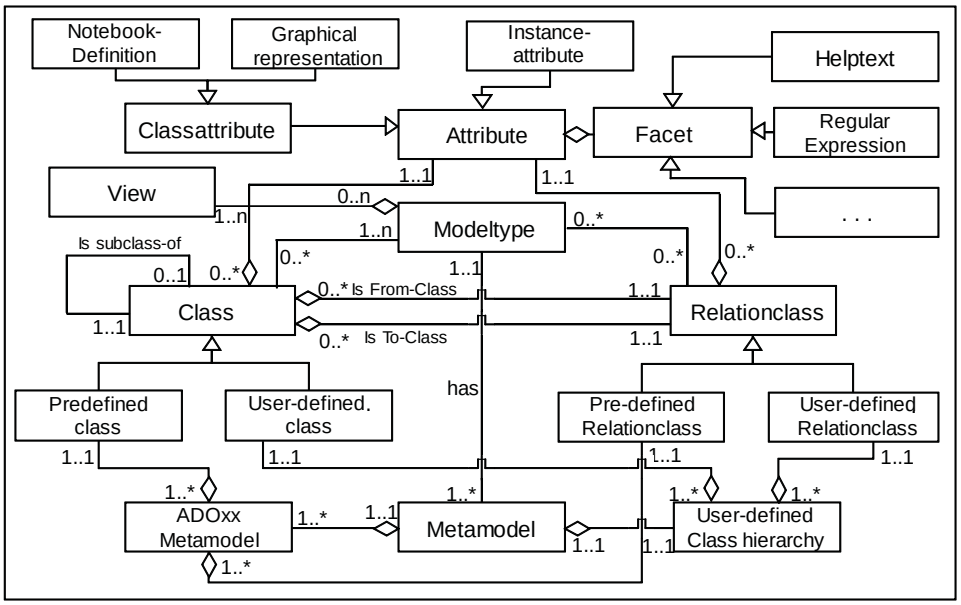

Figure 3: The ADOxx meta^2 model

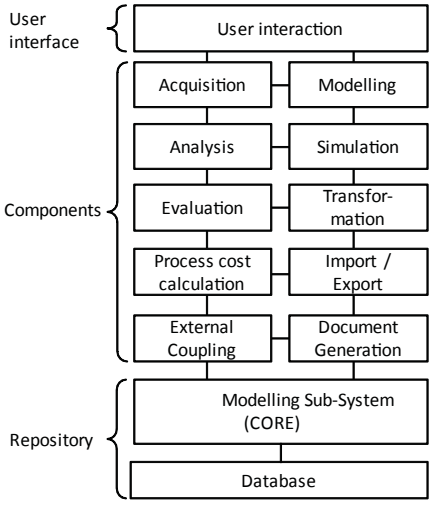

Figure 4: Architecture of ADOxx showing the components that can be used for conceptualising modelling methods

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

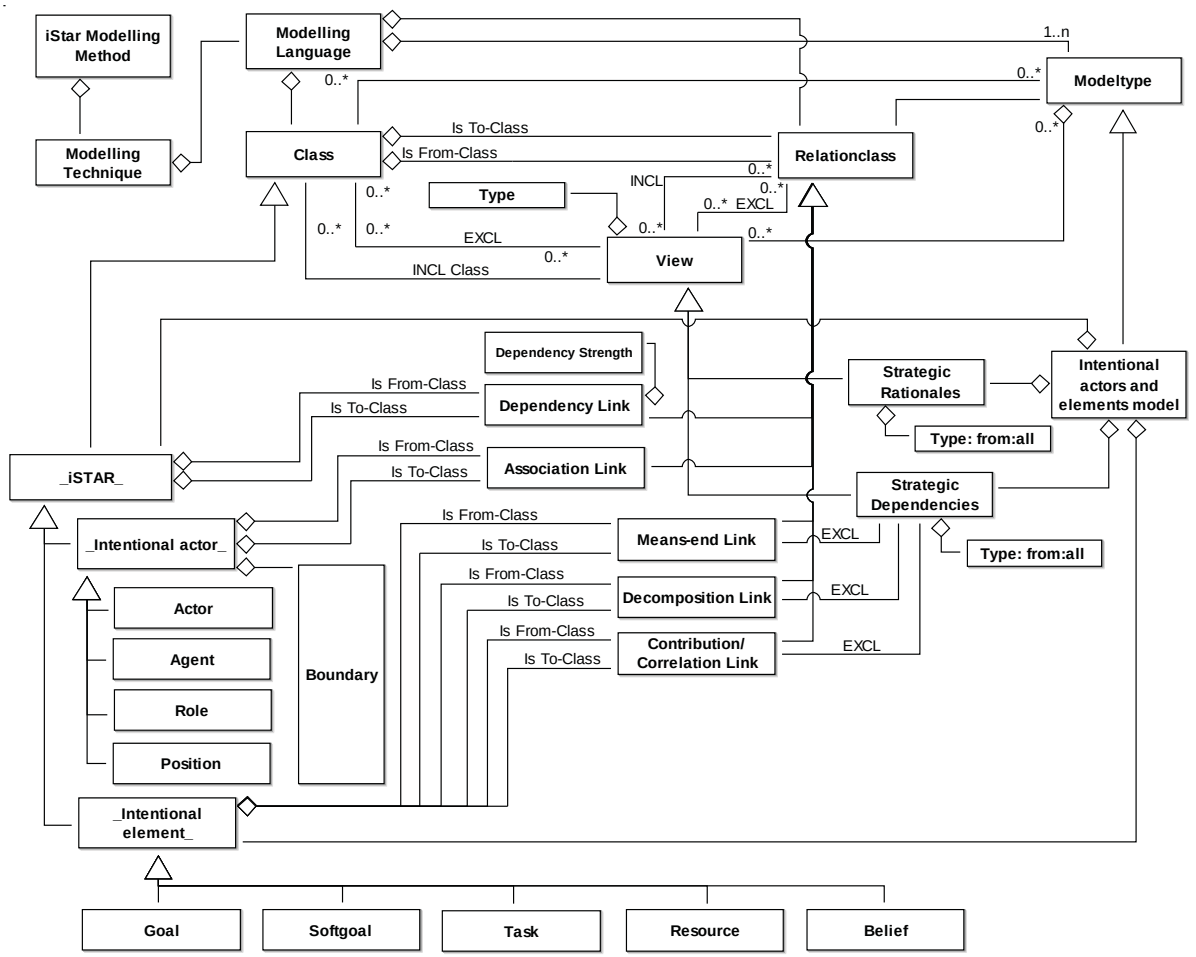

Figure 5: Excerpt of the iStar modelling method showing the intentional actors and elements model type, its classes, relationclasses, and attributes that are relevant for the specification of the visual notation

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.

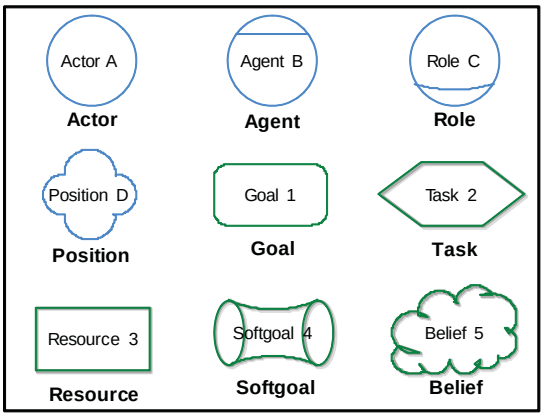

Figure 6: Visual notation for iStar classes

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.

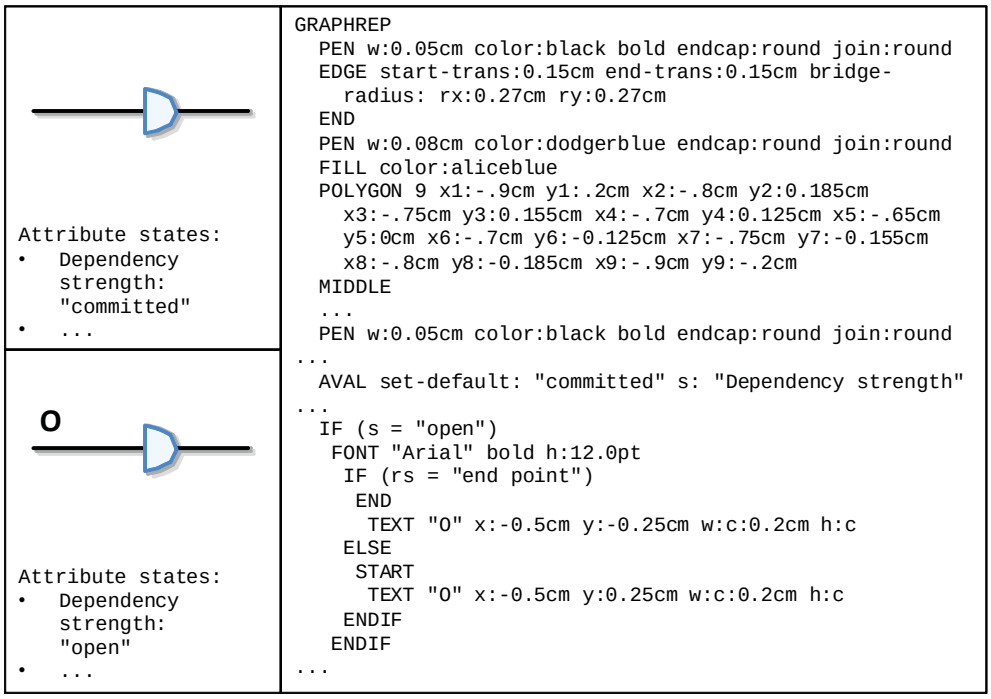

Figure 7: GRAPHREP definition for iStar dependency link relationclass

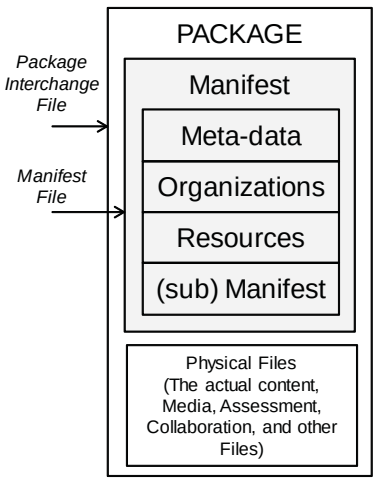

Figure 8: IMS content packaging scope, source: IMS 2001

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/`.

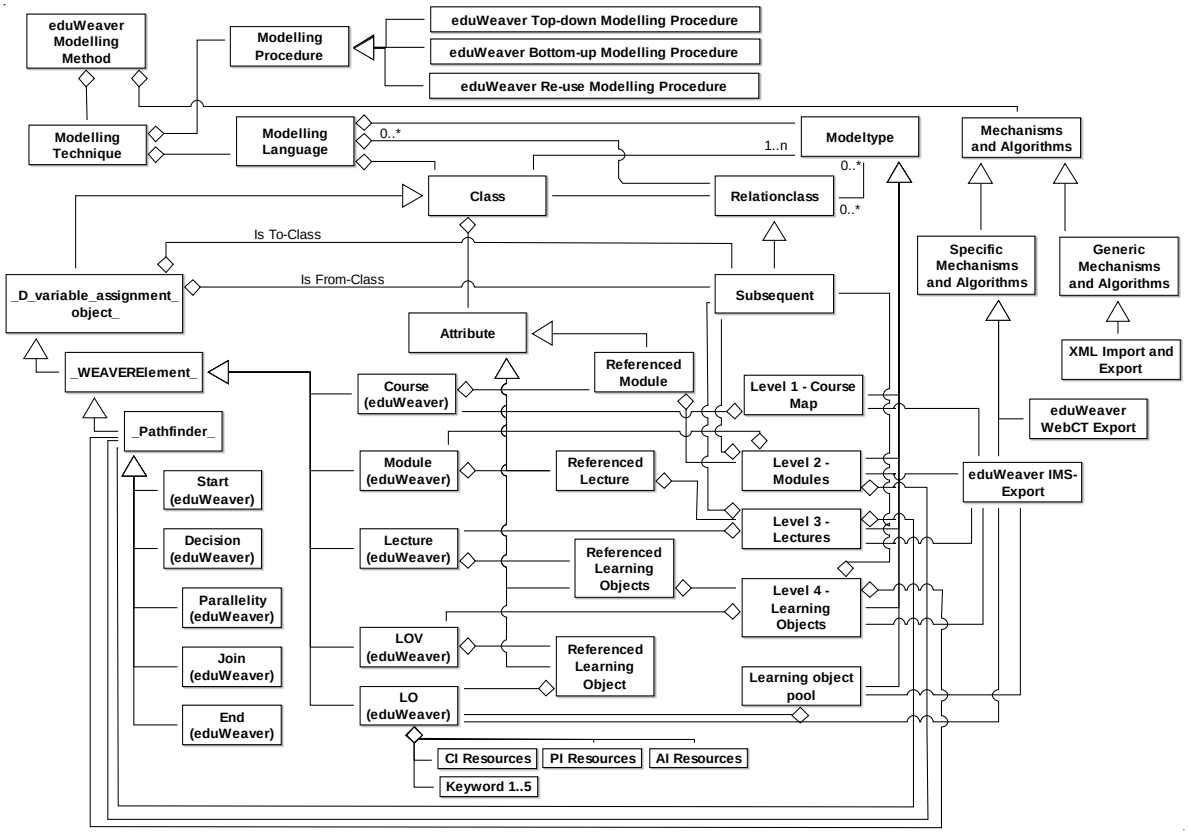

Figure 10: Excerpt of the eduWeaver modelling method showing the model types, classes, relationclasses, and attributes that are relevant for the IMS export algorithm

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.).

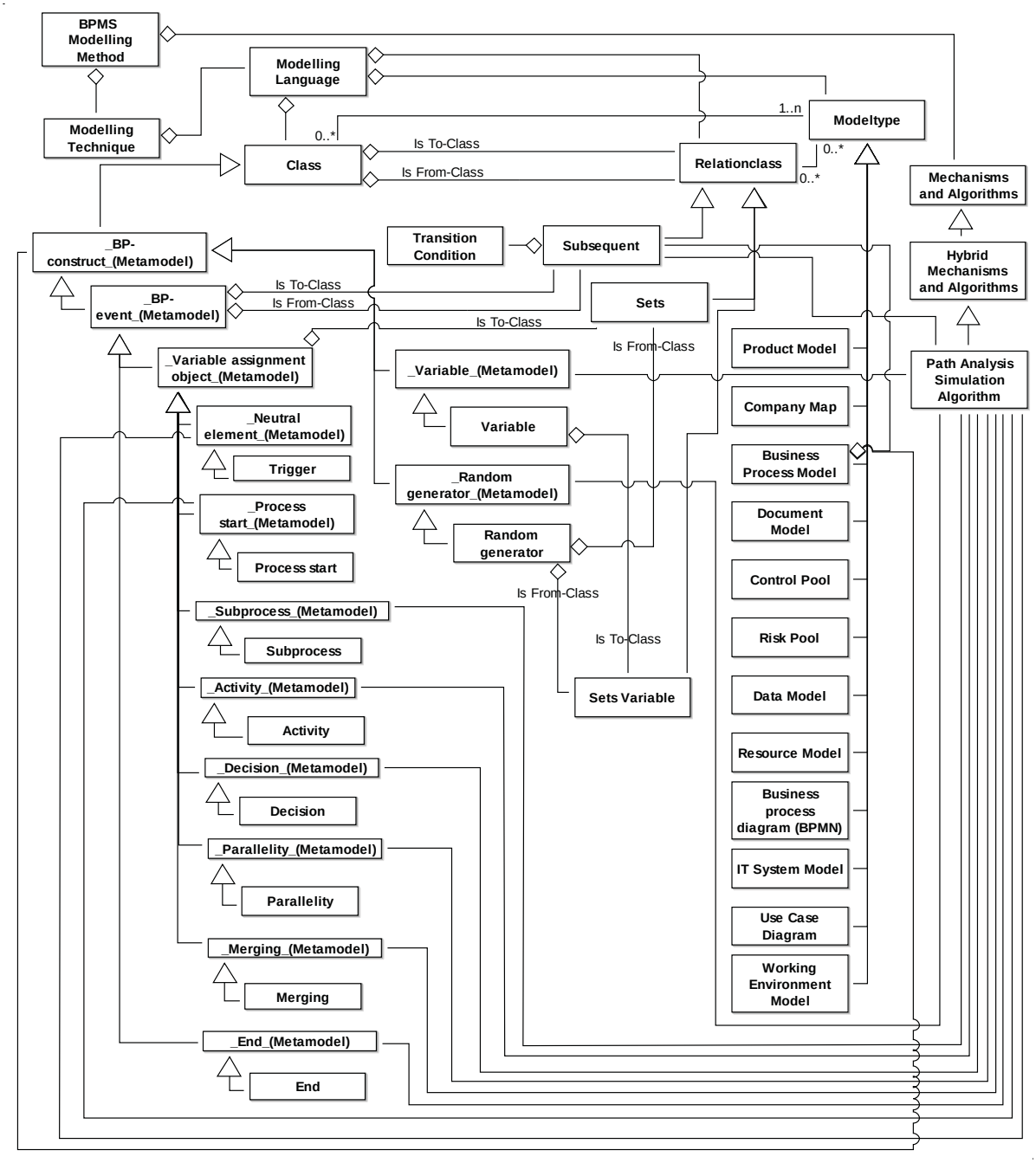

Figure 11: Excerpt of the BPMS modelling method showing the model types, the classes and relationclasses of the business process model type and how they are derived from the abstract classes of the ADOxx meta model that are targeted by the path analysis simulation algorithm

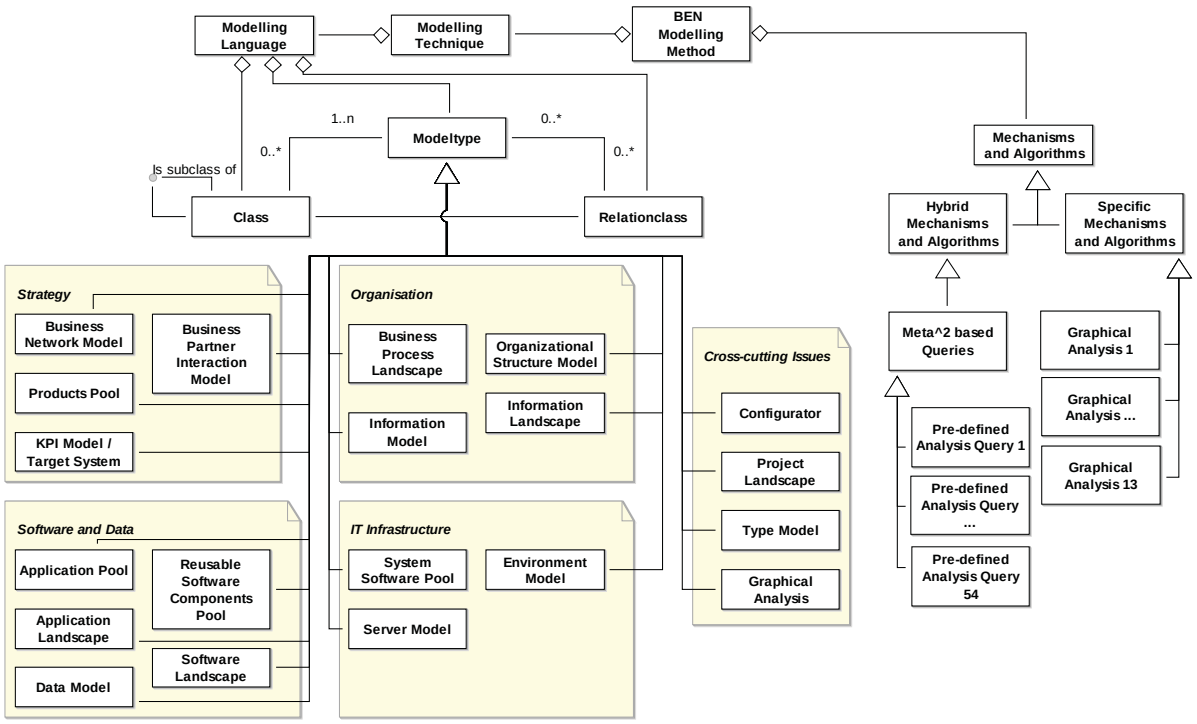

Figure 12: Excerpt of the BEN modelling method showing only the model types and the hierarchy of mechanisms and algorithms

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

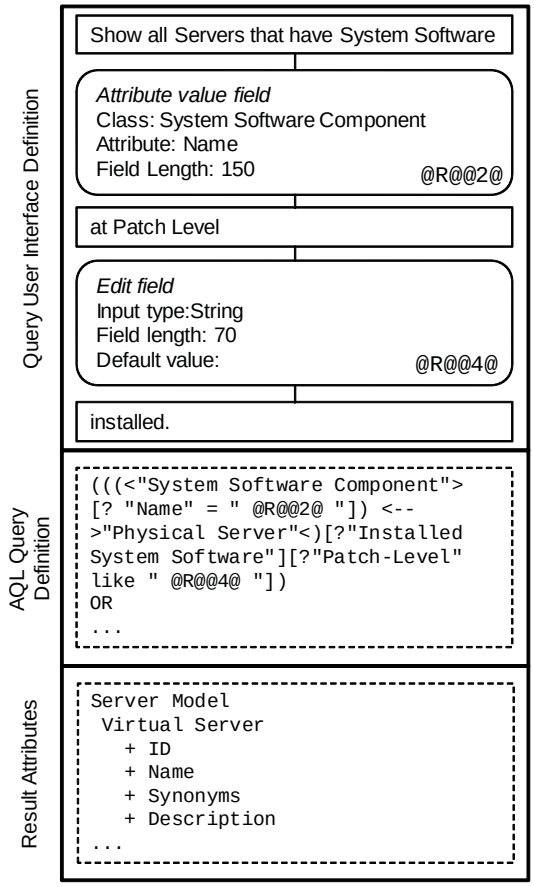

Figure 13: Excerpt of the conception of a pre-defined analysis query

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

Florian Matthes, Christian Neubert, Alexander W. Schneider

Fostering Collaborative and Integrated Enterprise Architecture Modelling

Enterprise Architecture Management (EAM) is a challenging task in modern enterprises. Tools supporting EA management activities are based on extensive models either created by enterprise architects or built-in. However, these models cannot be adapted according to specific data acquisition needs by persons having the architectural knowledge on the instance level. In this article, we describe how Hybrid Wikis empower these information carriers and enterprise architects to collaboratively and incrementally develop and manage a model in a bottom-up fashion by using wiki pages enriched with types and attributes. We visualise these emergent models by using UML-like class diagrams. Our approach is evaluated in a case study in a global operating bank participating in our Wiki4EAM community, a community of German enterprise architects applying Hybrid Wikis in different EA management contexts.

1 Motivation & Problem Statement

The field of Enterprise Architecture (EA) management constantly evolves since the often cited publication of Zachman (1987). Until today, the application of different subsequently published approaches did not result in the achievement of all promised benefits in practice. For example, Lucke et al. (2010) report that communication, shared understanding, and insufficient tool support are still critical issues in EA management. Tool support is often provided by means of heavyweight and expensive EA management tools to gather, structure, visualise, and analyse architectural information, such as business processes, applications, and organisational units. In order to obtain a holistic and consistent view of the EA to be managed it is required to define the concepts existing within an enterprise formally (cf. Bunge 1977). Although foundational ontologies exist for EAM frameworks currently applied in practice, for example ArchiMate (Azevedo et al. 2011) and TOGAF (Gerber et al. 2010), their meta-models need to be tailored (Källgren et al. 2009) to reflect the specific organisational context. In this situation often the so-called ivory tower syndrome (Raadt et al. 2008) emerges. That is, enterprise architects 'invent' an information model intended to be filled with data by the employees of the technical departments (Buckl et al. 2009b). However, these models are often rather unsuitable for the data acquisition as required by the persons having the concrete architectural knowledge as they are too abstract or too complex. Due to the high amount of data necessary to provide a holistic EA description many organisations today use specialised EAM tools (Matthes et al. 2008). The actual data input is thereby often performed by many different employees, because the architectural knowledge about single instances is spread over different employees within the company.

Our approach evaluates ways of bringing information carriers having deeper insights in an enterprise's structure within their individual scope and enterprise architects closer together by providing an integrated environment to collaboratively develop and manage both models and instances. It facilitates instance and model coevolution by using lightweight techniques, such as suggestions. Information carriers are enabled to create and adapt the description of instances according to their specific needs. From these in-

stances a model can be derived in a second step. Additionally, enterprise architects can explicitly define a model. For example, an architect can introduce new elements not yet represented by instances (e.g., introduce a new attribute) or introduce elements frequently used in instances but not yet defined in the enterprise model. Both models (i.e., derived and defined) are presented in a combined view helping enterprise architects to reflect their design decisions regarding the model. This way our approach mitigates the ivory tower syndrome.

The remainder of this article is structured as follows. In Sect. 2 we present a bottom-up approach for the creation of EA management models which uses Hybrid Wikis as concept and tool (Matthes et al. 2011). In order to enable enterprise architects to manage the resulting emergent and collaboratively created model, we introduce a suitable extension to UML class diagrams in Sect. 3. In Sect. 4 we present the findings of a case study with a globally operating bank, in order to demonstrate the feasibility of our approach for bottom-up and collaborative EA model creation. We demarcate our approach from other approaches for EA model creation in Sect. 5. Finally, Sect. 6 summarises this article and Sect. 7 provides a critical reflection and outlook on future research.

2 Approach

To address the challenges introduced in Sect. 1 Matthes et al. (2011) developed the concept of Hybrid Wikis and implemented it. Hybrid Wikis combine traditional wikis with a small set of structuring concepts in order to allow querying like in an object-oriented database. The provided structuring concepts are: attributes, types, attribute definitions, and constraints. These concepts are maintained (created, deleted, changed) by wiki users.

2.1 Structured Wiki Pages

Attributes and type assignments enrich individual wiki pages with structured information. For instance, a page can be typed with *business application* and provide an attribute *status* with value *planned* (cf. Fig. 1). A typed page represents an instance of a type. The data type of an attribute value can either be a literal (Text, Date, Number) or a hyperlink to another (typed) page. An attribute can be multivalued (i.e., an ordered list of values). Attributes can be freely added and removed to and from pages independent of the currently assigned types. The same applies to types, that is, a type, can be removed from a page without changing the currently assigned attributes. This means that types and attributes are basically independent from each other.

Wiki pages structured with attributes and type assignments represent the instances.

2.2 Underlying Model and its Validation

Types, attribute definitions, and constraints are elements representing the model of all instances. By means of attribute definitions attribute keys (e.g., *status* in Fig. 1) can be bound to a type. Constraints belong to an attribute definition and allow the specification of value ranges for attributes. A constraint, for example, can require that the values of an attribute (e.g., *responsible unit*) are links to pages having a specific type (e.g., *organisational unit*). A page's attributes (and their values) are validated by means of a constraint if the constraint's attribute definition is related to the respective attribute. That is, if their keys are matching (e.g., both have key *responsible unit*) and the page uses the same type the attribute definition belongs to. Constraint violations are indicated to the user on wiki page level by showing a validation message (cf. *requester* Fig. 1). However, users are never forced to enter valid values when changing an invalid page. Therefore constraints are referred to as 'soft' in Hybrid Wikis.

Additionally, types and attribute definitions can be declared as strict. Thus, changes to a page can only be stored if all attributes of this page are valid. However, in Hybrid Wikis everybody allowed to modify wiki pages (with attributes and type assignments) can also change types,

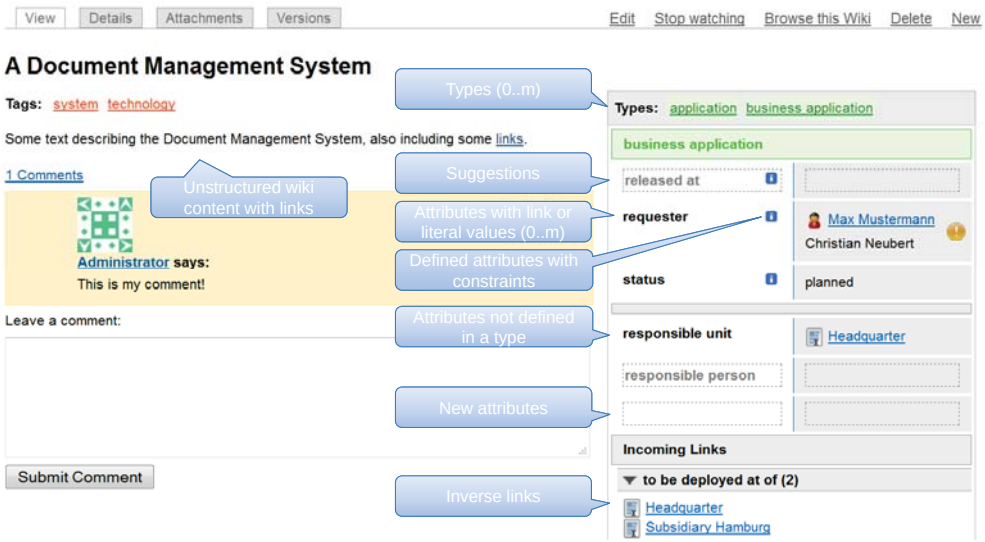

Figure 1: A wiki page provided by Hybrid Wikis

attribute definitions, and constraints. Therefore, when using strict types (or attribute definitions) users are rather urged to enter valid values only, but never forced.

In Hybrid Wikis instances and their model can diverge, even when using strict constraints. This is due to the fact that constraints can be defined even if violating pages result from this definition. But due to the loose coupling of both they can be changed independently without being restricted by each other. This way instance and model evolution is facilitated.

In Matthes et al. (2011) and Neubert (2012) all modelling elements of Hybrid Wikis, as shown in Fig. 2, are explained in detail. Additionally, it is explained how these concepts are implemented based on an existing wiki system in terms of, for example, algorithms, performance, system architecture, and technology. An important fact is, that attributes are related directly to wiki pages without the indirection of a type. Nevertheless, a type can be used to group attributes via the usage of attribute definitions. By this way of attribute modelling, users are enabled to assign them without thinking about types while modelers can still use familiar modelling concepts.

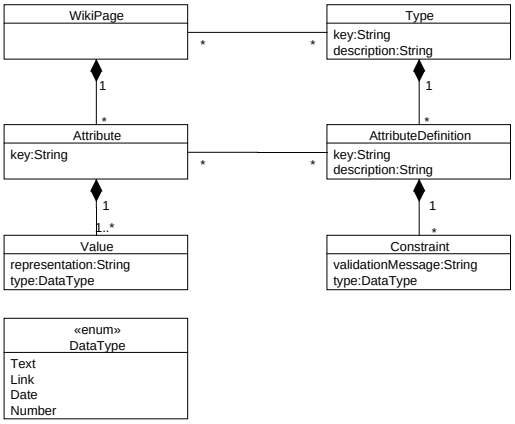

Figure 2: UML class diagram showing Hybrid Wikis core modelling concepts

2.3 Instance and Model Coherence and Evolution

With the possibility to assign attributes to content without using a type in parallel to the creation of models instances can differ from the model. In order to counteract a potential divergence of a model and its instances Hybrid Wikis provide the following mechanisms:

- Suggestions (e.g., of attributes)
- Search for violations of constraints
- Consolidation (e.g., of types)

2.3.1 Suggestions

On the instance level Hybrid Wikis provide attribute and type suggestions. Suggestions are calculated based on page's content and its currently used attributes and types. For instance, if a page is typed *business application* an attribute with key *status* is suggested if another page exists also typed *business application* additionally having an attribute *status*, or a key is suggested if the type of a typed page has an attribute definition.

On the model level, for example, a constraint is suggested if a certain number of pages (i.e., according to a specific threshold) uses similar attribute values. For example, if two pages typed with *business application* provide an attribute *responsible unit* and both have link values to pages typed with *organisational unit* a corresponding constraint is suggested for the attribute definition *responsible unit*, that is, restricting attribute values to be hyperlinks to pages of type *organisational unit*.

Furthermore, all input fields provide autocompletion support in order to guide users towards a consistent usage of terms and model elements. For instance, already used keys are suggested when the user enters an attribute key. Likewise, literals and links are suggested when entering an attribute value.

Suggestions are lightweight means to encourage users to provide structured content without forcing them. Moreover, they help to develop a coherent model when a bottom-up approach is used.

2.3.2 Inconsistency Detection

In Hybrid Wikis structured pages potentially become invalid without modifying the page, for example, by specifying a constraint. Therefore, users are supported in finding inconsistencies. Users can search for wiki pages

- containing any invalid attributes and
- having specific invalid attributes (e.g., searching for pages having an attribute *requester* with invalid values).

Based on these queries users can design their own dashboard in order to be aware of constraint violations.

2.3.3 Consolidation techniques

Once inconsistencies are detected, users can harmonise instances by means of the model. For example, renaming the key (e.g., *status*) of an attribute definition can be propagated to related pages (i.e., attributes of those pages are renamed accordingly). Or constraints can be applied to the related pages. For example, if *responsible unit* is restricted to links to pages with type *organisational unit* related attributes can be transformed into links matching this consistency rule. That is, literals are transformed into links (by creating new pages or referencing existing ones, both extended with type *organisational unit*, if required). In case of links the link target is typed with *organisational unit*, if required.

2.4 Integrated Model Management in Hybrid Wikis

In Hybrid Wikis models emerge from structured wiki pages and hyperlinks between them (i.e., from the actual instances). Model evolution is facilitated by suggesting attribute definitions and constraints based on an analysis of the instances. Evolution on the instance level is facilitated by providing attribute suggestions based on the underlying model. Additionally, the model can be used to harmonise (i.e., consolidate) derivations of the instances, because instances can be constrained by the model. This way, users are urged to follow the schema, but they are never forced.

Figure 3 depicts Hybrid Wikis' instance level and model level and sketches their interplay. Additionally, two participation roles are distinguished (Neubert 2012). Authors collaborate on the instance level by managing wiki pages with attributes and types, while tailors collaborate on the model level by managing attribute definitions and constraints.

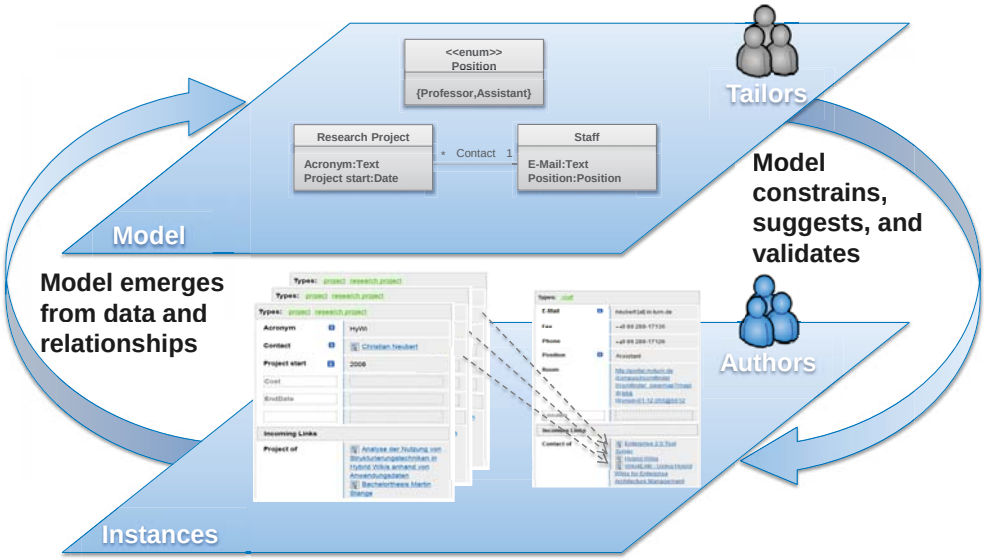

Figure 3: Data-driven information management and schema evolution via collaboration according to Neubert (2012).

2.5 Using Hybrid Wikis for EA Management

Hybrid Wikis provide means to collaboratively manage and model information for different purposes in different application domains such as notes and publications for (personal) information management within an university's chair. In the context of EA management, the idea is to start with existing unstructured information sources captured as wiki pages (e.g., derived from Office documents) and then to incrementally and collaboratively structure these pages with attributes and types as needed for enterprise architecture modelling. In Hybrid Wikis persons providing instance data (i.e., the authors, such as persons from the technical departments having the architectural knowledge) are never prevented from data entry caused by hard schema constraints and structures (i.e., types and attributes) can always flexibly be adapted as needed. This way, the 'true' model emerges bottom-up derived from user-managed, structured wiki pages (i.e., from the instances). Once a core model has emerged, enterprise architects (i.e., the tailors) can make this core explicit (e.g., by means of attribute definitions) and optionally define additional integrity constraints. However, the degree of rigidity can be softened by all users, if needed.

In this article we focus on how to derive EA models from collaboratively created and structured wiki pages and how these models can be visualised. In Buckl et al. (2009a) we discuss further advantages of using wikis for EA management (e.g., versioning, awareness).

3 Visualising Emergent Meta-Models

When authors model collaboratively with Hybrid Wikis, modelling usually occurs implicitly on the instance level. Thus, the model itself is not in the primary focus of participating authors and a visualisation of it is not available upfront. In the following the need for such visualisation as well as the used techniques will be described. In addition, an explicit model can be defined by tailors which also needs to be visualised. Finally, a combination of both is presented.

3.1 The Need for Visualisation

When authors create new instances or adapt the model implicitly during their daily work, they might be interested in its structure. This is particularly the case, if a type has many attributes

linking to different other types. In such case, the current way of displaying attributes and their values as a table might not give an adequate overview about a type and its relations to other types. In Hybrid Wikis the authors modelling on the instance level can only see directly associated instances. Hence, they are not able to see the structure or relationships of the associated types. With a UML class diagram visualising the underlying model this drawback of data-driven modelling can be mitigated as it provides a navigational aid. In addition, the visualisation of the model can be used to discover the terminology used on the instance level. This includes, among others, attribute names for new types or similar attributes in different types.

The second target group for the visualisation are enterprise architects who transfer control over the enterprise's model to authors providing data. If doing so, they need a familiar overview about the evolving model to be able to intervene if necessary. For example, if different semantically equivalent terms are used within the model enterprise architects can use the visualisation to discover them. Beside the consolidation of types and attributes enterprise architects have the ability to adapt the model by creating constraints or using the 'strict' property (cf. Sect. 2.2). Attributes declared as strict urge authors to provide a specific model element. Thus, enterprise architects can rather control the model evolution towards a specific target state.

By the use of a UML-like visualisation (cf. Fig. 7), which presents the model defined by the enterprise architects beside the actual instances filled by the authors, the enterprise architect is also able to reflect her decisions forming the enterprise model. With such visualisation at hand, it furthermore becomes easy to discover unused types and attributes, model parts which are regularly contravened as well as 'strict' properties which are not obeyed.

Extending Hybrid Wikis with the capability to visualise the data-driven model in parallel to the explicit model eases its usage for both, authors and tailors.

3.2 Mapping Hybrid Wikis' Instances to UML Object Diagrams

Each instance in Hybrid Wikis can be modeled as a single object in a UML object diagram. Thereby, the page's *type* is used as the UML object's type since it classifies a set of instances having a similar structure. Because there is no obligation to assign a type to each instance, untyped instances will be neglected within the resulting UML diagram.

The *attributes* within Hybrid Wikis can directly be mapped to UML's attributes. Therefore, the attribute's name becomes the UML attribute's name.

Within Hybrid Wikis, *hyperlinks* are the primary means to establish a relationship between two instances. They can either be used in the content section of a wiki page or as an attribute value. As the content part is completely neglected during model derivation, only links used as attribute values are analysed. If the linked instance has a type assigned, a UML association will be derived and the attribute's name will be used as role name. Currently, there is no possibility to define bidirectional links in Hybrid Wikis. Therefore, only independent directed links can be established on the instance level. As consequence, actually bidirectional links are represented as two independent directed associations in the derived UML diagram.

Figure 4 shows an exemplary object diagram. Therein, wiki pages of type *project* are shown with their respective attributes and associations.

As also visualised in Fig. 4 attributes are not associated to the type of an instance with the result that various instances of the same type can have different attributes. Consequently, a respective class diagram has to be abstract enough to account for that kind of diversity.

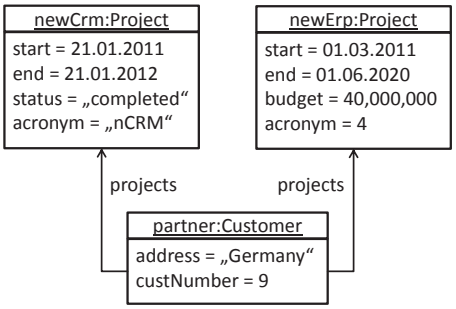

Figure 4: Exemplary UML object diagram of the instances

3.3 Mapping Hybrid Wikis' modelling Elements to UML Class Diagrams

In addition to an object diagram UML class diagrams are better suited to provide a holistic overview about the structure of types used for Hybrid Wikis' instances. The basis for such class diagram forms the already described object diagram. Classes and their attributes as well as their associations can be directly derived from it. By contrast, the attribute's data type has to be derived based on the various instances. Because an attribute's values can have different data types, the model derivation needs to account for inconsistencies. As a preliminary solution, the data type with the highest relative frequency will be used if attribute value types differ between instances.

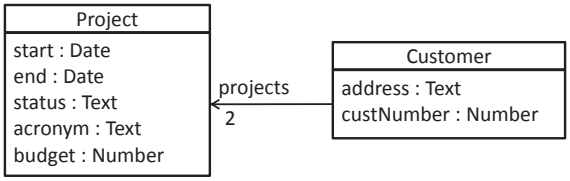

Figure 5: Exemplary UML class diagram of the implicit model

As a result, a class diagram visualising the structure of types used within Hybrid Wikis neglects the fact, that some attributes might not be present for some instances. Based on the object diagram shown in Fig. 4, Fig. 5 shows the corresponding class diagram.

3.4 Explicit Model Visualisation

In parallel to editing instances Hybrid Wikis support modelling also explicitly by the specification of *attribute definitions* and *constraints*. Always bound to a specific type, attribute definitions can be used to explicitly describe and refine type-specific attributes. By the use of constraints the values of an attribute can be defined in more detail. For example, the number of values can be set to 0..1, 1, 1..*, or *. An attribute's data type can be restricted to: Text, Number, Date, or Link whereas for data type Link one can also specify the concrete type the linked instance has to be assigned to. When defining a constraint for a specific attribute the constraint can optionally be marked to be 'strict'. In that case, the constraint will be enforced for all new instances while existing instances will not be changed. Because UML does not account for inconsistencies, the 'strict' property cannot be visualised in standard UML class diagrams. Using the same types already shown in Fig. 5, Fig. 6 shows an exemplary class diagram visualising explicitly modeled types.

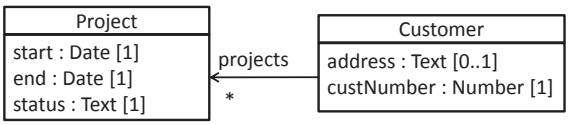

Figure 6: Exemplary UML class diagram of the explicit model

Since the explicit way of modelling is not bound to any instances it can of course lead to types without any instances. In that case, the respective UML classes are still depicted.

3.5 Instance and Explicit Model Co-Visualisation

When deriving UML class diagrams based on both, instance data and explicit model, some characteristics of emergent models cannot be expressed in UML. In the following, suggestions for UML extensions are presented to express emergent model characteristics as well as instance level and model level in parallel. Thereby, each

extension is presented in an exemplary UML-like class diagram derived from Hybrid Wiki data used within the domain of project management (cf. Fig. 7). In fact, the previously described class diagram visualising the instance level and the class diagram visualising the explicit model level are thereby combined.

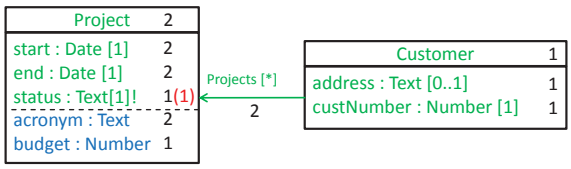

Figure 7: UML-like class diagram visualising an emergent model

3.5.1 Instance versus Model Level modelling

As previously explained, in Hybrid Wikis modelling takes place on two different levels: instance level and model level (cf. Sect. 2). Because UML class diagrams are not intended to distinguish between these two levels, an appropriate mechanism has to be introduced to express the difference of modelling levels.

First, the number of instances is an important characteristic of emergent models. Thus, it has to be reflected within the derived UML-like class diagrams for all elements including classes, attributes, and relationships. This can be achieved by inserting the concrete number after the respective element's name aligned to the right. Thereby, it can be easily distinguished from the UML instance number constraints which use brackets. For example, in Fig. 7 there are two instances of type *project* but only for one instance a *budget* is provided.

Second, within emergent models derived from instance data some information about the underlying model might be missing, because users are not forced to provide it. As a UML class diagram might require information of that kind, it has to be derived based on the available implicit instance data. For example, if all instances of a specific type have an attribute whose value links to exactly one instance of another type, a multiplicity of 1 will be derived for the association. Because the tailors have not already decided about an upper bound for the number of instances, this fact has to be expressed in the resulting diagram. A possible way therefore is the use of colours to express the difference between information provided explicitly by tailors (on the model level) and information derived from the instances. In Fig. 7 below dashed line the colour blue marks elements which are derived from the instances (i.e., the class diagram for instances, Fig 5). For example, the attribute *acronym* is mostly of data type Text but there might also be instances with an attribute *acronym* having another data type. Text is shown in the UML-like class diagram since it has the highest relative frequency.

Third, the distinction between the instance level and model level in general has to be expressed. This includes especially attribute definitions and constraints. Therefore, elements explicitly defined on the model level are highlighted in green and positioned above a dashed line (derived from the explicit class diagram, Fig.6). Using different colours according to the level of modelling directly shows the diagram reader how the respective elements emerged. For example, in Fig. 7 the attributes *start*, *end*, and *status* of type *project* are explicitly defined in the model by the use of attribute definitions and consequently marked green. Especially for associations between two types instance and model data is displayed in parallel if available. For example, the association between the types *project* and *customer* depicted in Fig. 7 shows, that there is a multiplicity constraint of [0..*] explicitly defined in the model but on the instance level each instance of type *customer* has exactly two *projects* assigned.

Within the current visualisation some facts of the instance model are overlaid by the explicit model. If a multiplicity of * is defined in the explicit model for an association the fact that each instance has exactly one object it refers to will be hidden. In addition, for attributes having at least

one instance using a hyperlink as value, a UML association is derived even if other instances use other data types like Text. In general, the data type derivation algorithm works as follows: if an attribute definition is present it will be shown accordingly. Otherwise, the data type used most often among the respective instances will be used. If none of this rules returns a single data type the following sequence of types will be used: Link, Date, Number, Text. This sequence ranks hyperlinks highest because we consider knowledge about relationships to be more important than literal attributes. The other data types are ranked according to their specificity.

3.5.2 Model Inconsistencies

The decoupling of the instances from the explicit model in Hybrid Wikis allows for inconsistent instances which do not comply to the constraints defined by the model. Because UML class diagrams disregard possible inconsistencies the expression of that characteristic also requires a UML extension. Again, the concrete number of non-compliant instances is displayed aligned to the right. To distinguish this number from the actual amount of instances, it is put into parentheses. Furthermore, using two different colours, the following two cases of non-compliance can be distinguished:

- The *Instance* violates a *Constraint*
- The *Instance* violates a 'strict' *Constraint*

As strict constraints force users to comply for all newly added instances, this kind of violation might be worse than violating a standard constraint. By using the colours red and yellow, this distinction is visualised. For example, the attribute *status* of type *project* displayed in Fig. 7 has a 'strict' constraint (visualised by an exclamation mark) that its value has to be a Date. As indicated by the red number, one instance is in violation to that constraint, having a *status* attribute of another data type. If the 'strict' property is not set for a specific constraint, a yellow number indicates the number of violations.

4 Application and Evaluation

To validate our approach, in December 2010 at Technische Universität München we established a community, namely Wiki4EAM (Matthes and Neubert 2011), of experienced enterprise architects from 25 large German organisations in order to pursue a lightweight, wiki-based approach to EA management (Buckl et al. 2009a). In the following we introduce the experiences gained in applying Hybrid Wikis in enterprises participating in our Wiki4EAM community. These experiences are described in detail in Neubert (2012).

In the period from December 2010 to March 2012 we conducted seven workshops together with the community members. In the first workshop the participants were introduced to the main concepts underlying Hybrid Wikis by presenting some slides. Additionally, we used a projector to demonstrate the core system functions (e.g., structuring of wiki pages, creating visualisations) by using a small EA management example scenario from the banking industry. After the workshops our software was made available to the members of the community. Some used the system hosted at Technische Universität München, some downloaded it and installed it locally. In the subsequent workshops, members of the community presented their developed wiki-based EA management solutions. Based on the experience gained with the use of Hybrid Wikis, new requirements were collected and discussed. This way, we constantly adapted and improved our solution according to the feedback of our industry partners.

4.1 Survey

In the 6th workshop in December 2011, we asked (paper-based) the community members to provide the main reasons why they are using Hybrid Wikis in their enterprises. The participating members stated (extract and most frequent answers) that

- the model is flexibly adaptable (i.e., can be created incrementally and does not need to be fixed in advance) (6 of 7) and
- Hybrid Wikis are easy to use (i.e., provide a high level of usability and a clean user interfaces) (4 of 7).

Since only seven members attended the community meeting in December 2011, these survey results do not represent a strong, well-founded evaluation. However, the answers allow to assume that the adaptability of structures is the main reason for applying Hybrid Wikis in EA management scenarios. Furthermore, the results indicate that business users understand the concepts underlying Hybrid Wikis since they adapt structures without additional IT-support facilitated by a web interface that participants consider to be easy to use.

4.2 Case Study

In the following we briefly introduce one selected case study from a global operating bank participating in the Wiki4EAM community. This bank is referred to as Bank A subsequently.

4.2.1 Starting Point

In December 2010, Bank A started participating in the community to deepen the knowledge about EA management. In particular, one enterprise architect of a unit concerned with infrastructure architecture management evaluated whether a previously created landscape of the unit's infrastructure can be represented by the concepts provided by Hybrid Wikis. That is, the architect evaluated if it is possible to build a model of the infrastructure landscape consisting of types, attributes, attribute definitions, and constraints. In an additional[1] two hours personal lesson with the architect of the infrastructure unit, the core concepts (e.g., wikis, wiki pages, types, attributes, constraints) were explained in detail and technical questions about some system functions were clarified (e.g., how to define queries, how to embed a query in a wiki page in order to create a visualisation). After this additional lesson, Bank A accessed Hybrid Wikis through a system hosted at TU München. The system included some EA management demo data (e.g., business applications, organisational units) and some exemplary visualisations (e.g., a cluster map showing the relation between organisational unit and business applications).

[1] In addition to the demonstration and visualisation during the initial workshop.

4.2.2 Model

In the period from December 2010 to December 2011 Bank A prototypically modeled the architectural elements of the infrastructure unit in a wiki separated from the wiki with the demo data. The resulting model representing the infrastructure architecture is depicted in Fig. 8.

The concepts of this model represent infrastructure elements on a certain level of detail. The model consists of four levels. Each level represents a different level of granularity in the IT infrastructure stack. For instance, an Infrastructure Component (IC) (level 4) can either be software or hardware. Likewise, an area (level 3) represents the logical unit for a set of ICs (e.g., a set of software components). Bank A incrementally developed this model by using wiki pages structured with attributes and types. The wiki's home page serves as a dashboard. The dashboard shows the relations between some types (e.g., between areas and infrastructure components) either as custom embedded table views or as graphical visualisations (e.g., as a cluster map). Both kinds of customised views answer specific EA management questions. For example, a cluster map shown on this dashboard indicates which ICs are used in which areas, or obsolete ICs are indicated in a (custom) tabular view by filtering for pages (typed with IC) having a colour attribute with value red.

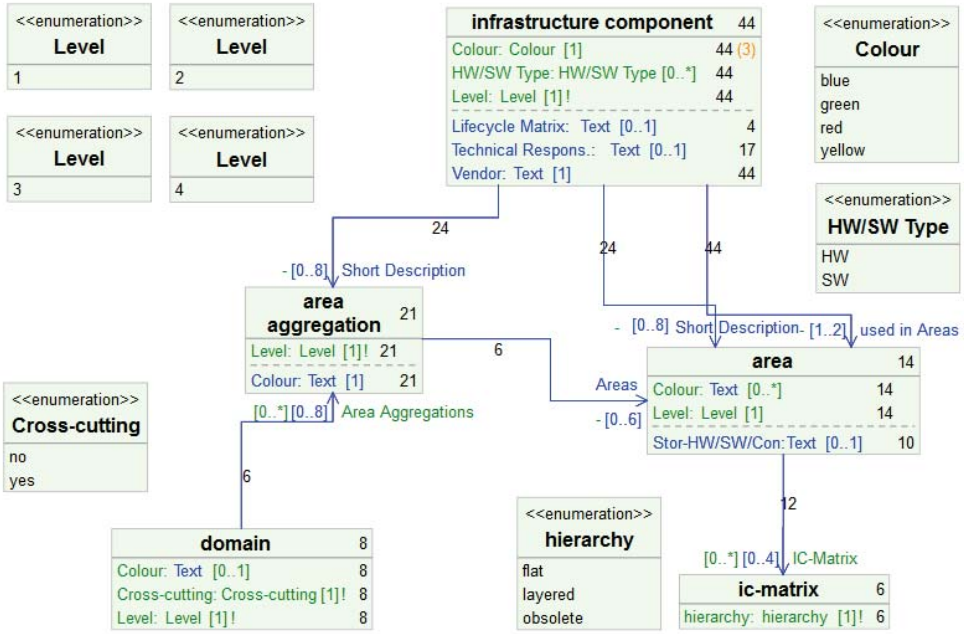

Figure 8: Emergent model used by Bank A for the management of infrastructure elements according to Neubert (2012)

4.2.3 Lessons Learned

In April 2011 in a Wiki4EAM community workshop, Bank A summarised the experiences gained with Hybrid Wikis (extract):

- Absolute beginners are not able to start without any introduction making a short lesson (about 2 hours) or user manual mandatory.
- Creating and adapting a model is possible without any further help from IT experts.
- Starting with instances is simpler than creating a holistic model first.
- Having a previously developed model in mind is helpful since it provides at least a basic frame for structuring.
- Gradually adding new attributes to wiki pages works well and creates beneficial structures.

These experiences show that Hybrid Wikis are applicable in the domain of enterprise architecture management. They empowered enterprise architects to develop a model in a data-driven incremental way. Furthermore, the adaptability of the model helped to shape the model according to changing information needs in order to answer typical EA management questions. In the introduced case, it was helpful to have a vague idea of the target model and some demo data at hand.

5 Related Work

In current literature different approaches for enterprise modelling exist. In order to distinguish the approach presented in this article from existing ones prominent representatives of different streams will be described in this section.

5.1 MoKi Wiki

MoKi (Modelling wiKi) is a wiki for the purpose of enterprise modelling (Casagni et al. 2011; Ghidini et al. 2009) built on the basis of Semantic MediaWiki. Its objective is to foster collaborative modelling of the constituents of an enterprise, in particular domain objects, business processes, and competencies. Therefore, templates and forms are used to prevent users from learning a special syntax. By contrast, Hybrid Wikis do not focus on the modelling part; models

emerge implicitly while users are documenting knowledge. It is also not tied to a specific domain such as enterprise modelling (Matthes et al. 2011).

5.2 DBpedia

The goal of DBpedia is to extract structured information from Wikipedia's content and to provide it publicly in the Web (Kobilarov et al. 2009; Lehmann et al. 2009). Thereby, a syntactic analysis examines the markup of Wikipedia pages using pattern matching techniques in order to extract RDF statements. More details about the extraction algorithm are described in (Fensel et al. 2011). In Hybrid Wikis, relationships between different pieces of content (wiki pages) can only be created explicitly in the structured part of each page. The unstructured part is not analysed to derive the underlying model up to now.

5.3 EA Management Tools

As outlined in the EA Management Tool Survey 2008 (Matthes et al. 2008) most EA management tools provide a rigid model. Often, such tools also include their own methodology of EA management and are not built to be adapted easily by the customer. Nevertheless, changes of class and attribute names as well as the introduction of new concepts is possible in some tools. By contrast to Hybrid Wikis, such changes cannot be implemented directly by the user but require the interaction of an administrator with special rights. Therefore, changes to a model can be implemented faster in Hybrid Wikis although losing a central governance instance.

5.4 Model and Meta-Model Co-Creation

The parallel evolution of model and meta-model are subject to research, for example, focusing on the model-to-model transformations (Cicchetti et al. 2008; Ruscio et al. 2011). Such approaches analyse possibilities of propagating changes made to a meta-model to the model and vice-versa. By contrast, Hybrid Wikis do not apply changes to one model to the other. Instead Hybrid Wikis just highlight possible mismatches of model and meta-model. Furthermore, both layers of models can now be visualised together in order to show implicit and explicit information structures in parallel.

5.5 Visualisation of Models

The visualisation of data models has a long tradition. In order to visualise a model it has to be present in some kind of standardised format. For instance, a widely used meta-model to describe models named Ecore has been developed by the Eclipse community (Steinberg et al. 2004). Because Ecore can be used to describe models as well as meta-models (since they are also models) both can be visualised by many different tools. But the goal of Hybrid Wikis is to visualise both models in a single diagram. To the authors' knowledge there is no standardised format nor tool available to create a visualisation depicting both instances and their underlying model.

6 Conclusion

In this article we discussed how the concept of Hybrid Wikis fosters the collaborative and integrated (meta-) model development in the context of EA management. In Sect. 2, we briefly introduced the modelling concepts of Hybrid Wikis and explained how their provided mechanisms facilitate (meta-) model evolution. Subsequently, in Sect. 3, we presented a new visualisation of emergent models inspired by UML class diagrams and enriched by additional information from the instance level. To validate our approach, in Sect. 4, we presented the results of a survey conducted among members of our Wiki4EAM community who applyed Hybrid Wikis within their organisation. Additionally, we presented a case study in a global operating bank using Hybrid Wikis for infrastructure modelling. We compared Hybrid Wikis in Sect. 5 with other modelling approaches described in scientific literature.

7 Discussion and Future Research

Although Hybrid Wikis seem to be useful for enterprise modelling their usage is subject to some preconditions. If used for EA management, the implementing organisation has to shift its culture to an open corporate culture. Enterprise architects transfer part of their control over the enterprise's model to employees responsible for data input. Since the presented collaborative enterprise modelling approach depends heavily on the active contribution of the employees having the architectural knowledge data input has sometimes to be motivated, for example, by architects leading with good example. Another lesson learned during the evaluation was that starting modelling from scratch is difficult. Therefore, Hybrid Wikis need to provide the ability to import existing modelling solutions and additionally to adapt them. Another threat of using Hybrid Wikis for enterprise modelling are subsequent alternations of versions also known as edit wars, cf. Viégas et al. (2004). The presented novel approach to visualise instances in parallel to their explicitly defined model has still some limitations. Possibly, mutually directed associations are used instead of bidirectional associations. Moreover, the visualisation readers might be mislead by an attribute data type derived from heterogeneous instance data and the number of its instantiations. If different data types are used for a single attribute, the actual visualisation suggests that all instances have the same data type.

In addition, future research should a) examine if Hybrid Wikis can serve as model repository to exchange and merge models from different EAM scenarios, b) investigate possibilities to support behaviour and views based on these emergent, adaptive models, and c) analyse and compare models of many other Wiki4EAM community members, for example, to identify modelling patterns.

References

Azevedo C. L. B., Almeida J. P. A., van Sinderen M., Quartel D., Guizzardi G. (2011) An Ontology-Based Semantics for the Motivation Extension to ArchiMate. In: Enterprise Distributed Object Computing Conference (EDOC). Helsinki, pp. 25–34

Buckl S., Matthes F., Neubert C., Schweda C. M. (2009a) A Wiki-based Approach to Enterprise Architecture Documentation and Analysis. In: The 17th European Conference on Information Systems. Verona, Italy, pp. 2192–2204

Buckl S., Matthes F., Schweda C. M. (2009b) Future Research Topics in Enterprise Architecture Management - A Knowledge Management Perspective. In: Workshop Trends in Enterprise Architecture Research (TEAR 2009). Stockholm, pp. 1–11

Bunge M. (1977) Treatise on Basic Philosophy: Ontology I: The furniture of the world. Reidel Publishing, New York

Casagni C., Francescomarino C. D., Dragoni M., Fiorentini L., Franci L., Gerosa M., Ghidini C., Rizzoli F., Rospocher M., Rovella A., Serafini L., Sparaco S., Tabarroni A. (2011) Wiki-Based Conceptual Modeling: An Experience with the Public Administration. In: Aroyo L., Welty C., Alani H., Taylor J., Bernstein A., Kagal L., Noy N. F., Blomqvist E. (eds.) International Semantic Web Conference (2). Lecture Notes in Computer Science Vol. 7032. Springer, pp. 17–32

Cicchetti A., Ruscio D., Eramo R., Pierantonio A. (2008) Automating co-evolution in model-driven engineering. In: Ceballos S. (ed.) 12th International Enterprise Distributed Object Computing Conference (EDOC), IEEE Computer Society

Fensel D., Facca F. M., Simperl E., Toma I., Fensel D., Facca F. M., Simperl E., Toma I. (2011) Semantic Web. In: Semantic Web Services. Springer, pp. 87–104

Gerber A., Kotzé P., van der Merwe A. (2010) Towards the Formalisation of the TOGAF Content Metamodel Using Ontologies. In: Proceedings of the 12th International Conference

on Enterprise Information Systems (ICEIS), pp. 54–64

Ghidini C., Kump B., Lindstaedt S., Mahbub N., Pammer V., Rospocher M., Serafini L. (2009) MoKi: The Enterprise Modelling Wiki. In: Aroyo L., Traverso P., Ciravegna F., Cimiano P., Heath T., Hyvönen E., Mizoguchi R., Oren E., Sabou M., Simperl E. (eds.) The Semantic Web: Research and Applications Vol. 5554. Springer, Berlin, pp. 831–835

Kobilarov G., Bizer C., Auer S., Lehmann J. (2009) DBpedia - A Linked Data Hub and Data Source for Web Applications and Enterprises. In: Proceedings of Developers Track of 18th International World Wide Web Conference (WWW 2009). Madrid, Spain

Källgren A., Ullberg J., Johnson P. (2009) A Method for Constructing a Company Specific Enterprise Architecture Model Framework. In: 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing. IEEE, pp. 346–351

Lehmann J., Bizer C., Kobilarov G., Auer S., Becker C., Cyganiak R., Hellmann S. (2009) DBpedia - A Crystallization Point for the Web of Data. In: Journal of Web Semantics 7(3), pp. 154–165

Lucke C., Krell S., Lechner U. (2010) Critical Issues in Enterprise Architecting - A Literature Review. In: Proceedings of the Sixteenth Americas Conference on Information Systems. Lima, Peru

Matthes F., Neubert C. (2011) Wiki4EAM: Using Hybrid Wikis for Enterprise Architecture Management. In: 7th International Symposium on Wikis and Open Collaboration (WikiSym). Mountain View, California, USA

Matthes F., Buck S., Leitl J., Schweda C. M. (2008) Enterprise Architecture Management Tool Survey 2008. TU München, Chair for Informatics 19, Prof. Matthes (sebis)

Matthes F., Neubert C., Steinhoff A. (2011) Hybrid Wikis: Empowering Users to Collaboratively Structure Information. In: Proceedings of the 6th International Conference on Software and Data Technologies. Seville, Spain, pp. 250–259

Neubert C. (2012) Facilitating Emergent and Adaptive Information Structures in Enterprise 2.0 Platforms (to appear). PhD thesis, Fakultät für Informatik, Technische Universität München

van der Raadt B., Schouten S., van Vliet H. (2008) Stakeholder Perception of Enterprise Architecture In: ECSA 2008 Morrison R, Balasubramaniam D, Falkner K (eds.) Springer, pp. 19–34

Ruscio D., Ralf L, Pierantonio A. (2011) Automated co-evolution of GMF editor models. In: Software Language Engineering. Springer, pp. 143–162

Steinberg D., Budinsky F., Merks E., Paternostro M. (2004) EMF: Eclipse Modeling Framework, Second Edition

Viégas F. B., Wattenberg M., Dave K. (2004) Studying cooperation and conflict between authors with history flow visualizations. In: Proceedings of the 2004 Conference on Human Factors in Computing Systems - CHI '04 6(1), pp. 575–582

Zachman J. A. (1987) A framework for information systems architecture. In: IBM Systems Journal 26 (3), pp. 276–292

Florian Matthes, Christian Neubert, Alexander W. Schneider

Chair for Software Engineering for Business Information Systems (sebis),
Institute for Informatics,
Technische Universität München
{matthes | christian.neubert | alexander.schneider}@tum.de

Tony Clark and Balbir Barn

Goal Driven Architecture Development using LEAP

Methods for goal driven system engineering exist and propose a number of categories of goals including behavioural, formal, informal and non-functional. This article goes further than existing goal driven approaches by linking goals directly to the semantics of an architectural modelling language called LEAP with an operational semantics. The behavioural goals are expressed using a Linear Temporal Logic and the non-functional goals are expressed as functions over meta-properties of the model. The meta-properties are supported using an encoding represented using Java reflection. The article describes the LEAP approach using a simple case study written in the LEAP language supported by the LEAP toolset.

1 Introduction

The architectures of modern IT systems are distributed and heterogeneous and therefore lend themselves to design using component-based approaches. A component based approach, as opposed to large scale ERP implementations leads to multiple possible configurations of system components raising questions such as what is the best component configuration and how to develop component-based designs. Any development that changes an architecture should start with a requirements analysis phase, yet most modelling approaches focus on *what* the system should do. In Architecture Design Languages (ADLs) or System Design Languages such as UML, functional behaviour is expressed using invariants and pre and post-conditions. In Enterprise Architecture (EA) these are represented by as-is and to-be architectures most clearly characterised by approaches such as TOGAF (Spencer et al. 2004). Other approaches such as Archimate (Lankhorst et al. 2010) also utilise informational, behavioural and structural models organised as different architectural layers such as business, application and technical infrastructure to express these architectures. Despite the exhaustive modelling performed to develop an architecture model for an organisation's new requirements, scant effort is applied to understand and codify the knowledge that represents the rationale of the *why* behind these architectural modelling decisions.

Requirements in the form of functional system specifications are supported by a number of technologies including UML and formal languages such as B and Z, and various logics. UML can be categorised as structured but imprecise and the formal languages as being precise but generally unstructured or difficult to map to implementation features.

In both cases, these technologies do not support motivational aspects of system development that are often expressed in terms of non-functional properties such as *cost*, *reliability* and *usability*. Motivation or Intention of business requirements, if supported satisfactorily, can provide a means for analysing the relationships between business requirements or needs and IT infrastructure and thus address one of the perennial issues in Information Systems/Information Technology (IS/IT) research, that of business-IT alignment. This relationship between business and IS/IT performance has received much attention from as far back as 1977 (McLean and Soden 1977) and a more recent review of the key issues being identified in Chan and Reich (2007).

Motivation or intention aspects of requirements engineering has resulted in a new branch requirements modelling based around goal oriented requirements engineering (GORE) techniques

(Mylopoulos et al. 1999) such as i* (Yu 1997; Yu and Mylopoulos 1994) and KAOS (Dardenne et al. 1993; Letier and Van Lamsweerde 2004; Van Lamsweerde 2008). GORE based techniques present a variety of options for analysis such as providing a more formal basis of how goals realise other goals, conflict between goals and the positive and negative contributions goals make to other goals. Further, the relationship between the proposed solution and actual need is more clearly delineated.

Requirements Engineering methods such as KAOS and i* aim to address the structured aspect of requirements in terms of goal modelling. Goals capture the motivation behind system design and goal modelling languages provide a mechanism for structuring the goals and linking them to system elements that are responsible for achieving the goals.

In order to be effective goal-modelling must address the following:

precision In the early stages a requirements engineer is likely to have a broad understanding of the required system. Therefore, goals should support informal discursive requirements. However, as requirements are refined, their precision should increase to the point where they can be, in principle at least, processed mechanically.

semantics Whether a goal is informal or formal, it must be possible, in principle, to provide its meaning. In general a goal is a predicate over some features of a system. Behavioural goals are predicates over system executions and therefore, it is important to be able to articulate such executions to the required level of precision. Non-functional goals are typically more difficult to express, however our proposal is that non-functional goals are predicates over *meta-properties* of a system (whether static or dynamic). If, in principle, a non-functional goal cannot be expressed precisely in these terms then it is not measurable and is of limited use in system development.

structure The use of requirements engineering techniques are justified in terms of system size and complexity. Therefore, requirements must have structure that allows them to be decomposed and analysed independently. Decomposition should support the analysis of alternatives. Ultimately, behavioural goals should decompose into system component responsibilities and therefore the goal model structure should support links to design elements that realise the specified behaviour.

LEAP is a technology for constructing and animating architectural models (Clark and Barn 2011, 2012; Clark et al. 2011). It is based on a small collection of features including: components and connectors, messages, operations, operation specifications, information models, events, state machines, and rules. The design of LEAP has been motivated by a desire to provide a simple collection of orthogonal executable modelling features that can be used as a basis for system design from enterprise-wide architectures through to individual IT components. Our approach is to use components as containers of information and behaviour, and to use messages between connected components as a basis for computation. Components can be used to represent both physical and logical features of a system, and the data stored in components and passed in messages between components, may include components themselves. Our claim is that by making components higher-order features of the LEAP language offers a highly expressive basis for system modelling at all levels without the need for a diverse collection of different elements.

LEAP is based on existing languages and approaches including the port-and-connector models of UML, class and object models of UML, state machines, the Object Constraint Language (OCL), functional programming languages particularly higher-order functions, list comprehensions and pattern matching, event driven programming, and KAOS. LEAP uses a functional language in two ways: to implement component behaviour and to abstract over system models. The LEAP

tool supports the leap language and provides textual and graphical editors for constructing and viewing LEAP models. The definitions in this article are LEAP source code and the diagrams (with the exception of Fig. 8) are generated by the LEAP tool.

Existing goal-modelling languages address precision, semantics and structure as described above. However the degree to which this is achieved is limited because the languages are either imprecise (such as BMM) or general purpose (such as KAOS and i*). In particular KAOS provides a formal language for behavioural goals based on temporal logic, however this does not map on to any specific executable system and therefore remains very general. In addition, no existing goal modelling notation addresses the issue of non-functional requirements in a precise way.

LEAP makes a contribution to architectural modelling in the following ways:

- LEAP brings together an integrated collection of orthogonal features that we propose as a basis for the design of component based architecture. Our claim is that these features are appropriate for high-level architectures such as those found in EA and also appropriate for smaller scale system architecture. This article provides an example of a system architecture but see Clark and Barn 2011, 2012; Clark et al. 2011 for other examples.
- LEAP extends component-based modelling with intentional features in the form of goals, this together with the executable features of LEAP makes it unique as a component-based design language. This article provides examples of the use of the intentional features.
- LEAP uses a formal logic to express behavioural goals over component executions, whilst other systems provide such mechanisms, LEAP is unique in that it integrates the logic with a traditional component-based modelling language. This article provides many examples of LEAP behavioural goals and defines the formal language.
- Our proposal is that non-functional goals can be formalised as meta-predicates over extra-calculational system properties. This allows so-called *soft* goals to be precisely defined in LEAP compared to other approaches that require non-functional goals to be expressed in natural language. This article provides a number of examples of non-functional goals in LEAP and describes the technical machinery that allows the non-functional goals to be checked.

This article describes the LEAP approach to goal modelling. We introduce the approach using a case study and then define the languages used. Finally we compare LEAP with other goal modelling approaches.

2 Case Study

Ruritanian General Practitioners (GPs) are required to provide an automated consultation booking system. Patients register with a medical practice in order to use it. When they register they may indicate a particular GP that they wish to see during consultations. The Ruritanian medical system is entirely on-demand: patients walk in to the practice and request a consultation. If the patient is registered with a particular GP then they will see that GP when they are free, otherwise the patient sees any GP. Being a Ruitanian GP is tough since they must always be available in the surgery. A record must be kept of all consultations and the medicines that are dispensed.

Apart from the functional requirements given above, Ruritania defines some non-functional requirements in terms of *fairness*, *cost*, *efficiency*, and *risk*. It requires that its medical practices are *fair* in the sense that no GP is overworked, all patients are seen within a sensible time, and no consultation takes too long. In addition, the *total cost of ownership* for a medical practice should be below a specified amount. The costs will include the development cost of the software, the costs of running the software, and the costs of the GPs. The *risk* of sensitive medical knowledge being leaked is reduced if the system is distributed

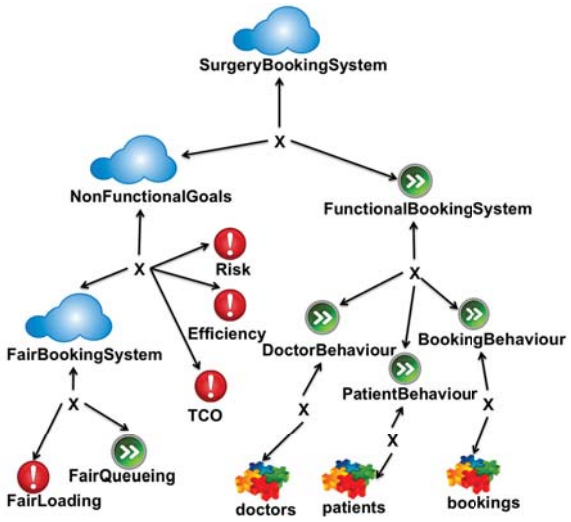

Figure 1: The Surgery Goal Tree

over more than one site. Finally, a system implements the requirement functionality as *efficiently* as possible.

3 LEAP: Goal Directed Models

3.1 Goals

Figure 1 shows the LEAP goal decomposition tree for the GP case study. All goals denote predicates over aspects of a system architecture, for example *SurgeryBookingSystem* are the conditions under which any architecture represents an acceptable system, whereas *Doctors* are the conditions for the correct behaviour of the part of the system that manages information about GPs. Goal types differ in terms of the type of language used to express the condition and in terms of the things that the goal can denote. LEAP supports goals of the following types:

informal An informal goal is expressed using natural language. It is intended to scope out an area of the system that is subsequently refined. A goal decomposition tree usually has an informal goal at its root. Informal goals are expressed in LEAP diagrams as clouds.

behavioural A behavioural goal uses *linear temporal logic* to precisely define the behaviour of some aspect of the system. A LEAP model consists of a collection of components each of which contains a database of terms. LEAP execution occurs in terms of messages that cause changes in component databases; therefore LEAP executions are sequences of states and messages. A behavioural goal is a condition that applies to LEAP executions. Behavioural goals are shown as nodes labelled ».

Behavioural goals are not necessarily limited to the scope of a single component. As a goal-decomposition tree is developed from root to leaves, the scope of behavioural goals nearer the root are likely to be scoped over sub-systems that comprise multiple interactive components. Behavioural goals near the leaves of the tree tend to relate to single components and may even be limited to single operations.

Invariant An invariant goal is something that must be true at all times during system execution. A behavioural goal that specifies a condition that must hold in all states of an execution is an invariant. However, the behavioural goals described above are specified using a language that is limited to component messages and states. Other types of invariant relate to meta-properties of a system such as cost, reliability, etc. Therefore, LEAP invariants can be expressed using the following meta-features: **`state`** which is used to reference the current system state in terms of its messages and database terms; **`calc`** that is used to reference the sequence of states in a system execution; **`reify`** that is used to map between model elements and database terms; **`intern`** that is used to map between database terms and model elements. Invariants are expressed on LEAP diagrams as `!`.

component Goals can be linked to specific LEAP components. The goal may be used to specify that the component has particular behaviour or meta-properties. Behavioural goals

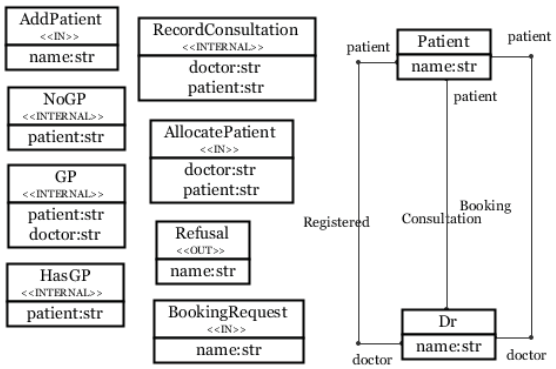

Figure 2: Goal Data

can be used to specify the behaviour of particular component operations and as such will map directly on to the specification contained in the component.

Goal decomposition is shown as nodes and links that connect the goal types described above. Decomposition may be in terms of conjunction (X) or disjunction (+). Decomposition is a mechanism for separation of concerns and for refinement.

The model shown in Fig. 1 has a root goal `SurgeryBookingSystem` that is decomposed into `NonFunctionalGoals` and `FunctionalBookingSystem`. The details of these goals are described in the following sections.

3.2 Behavioural Goals

The behavioural goals are defined with respect to the data types shown in Fig. 2. In LEAP, both the state of a component and the messages that are processed by a component are represented as *terms* whose types are defined by classes. We use UML-style stereotypes to designate the difference between messages and passive data. Figure 2 uses the tags «`IN`» and `OUT` to represent messages that communicate with the system environment. The tag «`INTERNAL`» is used to represent a message that is both produced and consumed by the system. The reason for using data for both active and passive information is that goal-based requirements need not commit to specific distinctions at such an early stage of development.

The `FunctionalBookingSystem` goal specifies a range of system behaviour. Behavioural goals use *Linear Temporal Logic* (LTL) to express constraints, for example:

```
always {
  forall BookingRequest(p) {
   Patient(p) implies
     before(10) {
       Consultation(_,p)
     }
  }
}
```

requires that after a booking request is recorded, if the patient is registered with the practice then a consultation must be recorded before 10 minutes have passed. The following goal requires that a request is politely and immediately denied for customers not registered with the practice:

```
always {
  forall BookingRequest(p) {
    not(Patient(p) implies next { Refusal(p) })
  }
}
```

Finally, the following requires that a patient is eventually seen:

```
always {
 forall BookingRequest(p),Dr(d) {
  Patient(p) and Registered(Dr(d),Patient(p))
   implies
    eventually {
      Consultation(Dr(d),Patient(p))
    }
  }
}
```

3.3 Non-Functional Goals

Functional goals can be expressed in terms of system behaviour that is represented in terms of *calculations* (sequences of run-time states). This may be expressed as pre and post-conditions (one step in the calculation), invariants (every step in the calculation), or as LTL expressions (sub-sequences of the calculation). Our proposal is that non-functional goals are those that require extra-calculational information, i.e., data that relates to any aspect of the system execution, but may not be directly necessary to express the

execution rules. Examples include the cost of resources that are used during execution, whether or not an architecture satisfies given structural guidelines, and the rates of component failure that lead to system exceptions.

Therefore, non-functional goals are expressed in terms of meta-properties of the system. The properties should be made sufficiently precise so that, at least in principle, they can be mechanically checked. Meta-properties may be static or dynamic. Static meta-properties include structural properties of the system models, for example checking how many connections a component has or placing a requirement on the overall number of components. Checking for architectural patterns is a meta-property of a system. In addition, it is important to allow developers to extend the basic meta-types of a system to support their own static meta-information that can be checked in constraints. A typical example of this is the extension of standard UML classes to introduce a new RDBMS table meta-type.

Static meta-properties can be extended to dynamic meta-properties in a straightforward way providing the system has a well defined dynamic semantics. Typically this will involve defining a static structure for the system and then extending it to sequences, trees or graphs of system states. Once these system execution structures are defined it is possible to define measurable dynamic non-functional properties in terms of the meta-properties of the individual states.

LEAP supports meta-access in the following ways. The contents of the current system state is available via the variable **`state`** and the sequence of states in a system calculation is denoted by **`calc`**. Any LEAP value can be transformed into a LEAP term that has a uniform structure and which can be processed using pattern matching, using the meta-operator `reify`. The inverse of `reify` is called `intern`.

In the following example goals, we will make use of some operators that allow a LEAP component to be processed as a LEAP term. The operator `walkComp` is defined below as a standard LEAP operator (all the code in this article is written in the LEAP programming language). The arguments of `walkComp` are `map` that transforms all components in a tree, `cons` that combines a mapped component with its mapped children; `sib` that combines mapped components with their siblings; `base` that is the result of mapping the empty sequence of component siblings.

```
walkComp(map,cons,sib,base) {
 fun (comp) {
  let f = fun(children) {
   case children {
    c:cs →
     sib(walkComp(map,cons,sib,base,c),f(cs));
    [] → base
   }
  } in cons(map(comp),f(comp.children))
 }
}
```

The operator `getComponents` is constructed by supplying `walkComp` with the identity mapping `id` and operators that build lists. The operator `getMess` maps a component to the messages it processes:

```
getComponents = walkComp(id,cons,app,[])
comp2Messages = fun(c) [ m |
  p ← c.ports,
  m ← p.messages
]
getMess = walkComp(comp2Messages,app,app,[])
```

Risk: The Ruritanian Government has identified that public sector systems are at risk if they entirely hosted in one place. Therefore, the goal `Risk` requires that any compliant system must be distributed over at least 2 hosts. The goal needs to reflect on the structure of the system and requires that each component has meta- information defining where the component is hosted:

```
let system = reify(self);
    C = getComponents(system);
    hosts = set([ intern(c).host | c ← C ])
in #hosts > 1
```

Efficiency: The efficiency of a system can be defined in relation to the amount of inter-component communication that is performed. The Ruritanian Government requires that any IT solution to the surgery requirements are *efficient* where

this is defined in terms of a measure of the number of possible messages within the system (recall that *risk* requires at least 2 different components).

The number of messages in a system model is a meta-property. It is found by reifying the system and mapping the resulting term to the number of messages it contains. The size of the resulting set is calculated by the following:

```
let system = reify(self)
in fold(add,0,set(getMess(system))) < 20
```

TCO: The total cost of ownership is defined the Ruritanian Government as the development costs, the hosting costs and the staffing costs of any IT system. The non-functional requirement that the TCO should be less than 1000 Ruritanian Rurs is a meta-constraint that is applied to properties of the model. Both the development and hosting costs are meta-properties of the components in the system. The staffing costs are calculated by mapping the state of all system components to the set of GPs that they contain and then multiplying by `GPcost`:

```
let system = reify(self);
   C = getComponents(system);
   devcosts = [intern(c).devcost | c ← C];
   devcost = fold(add,0,devcosts);
   hostcosts =
     [(intern(c)).hostcost | c ← C];
   hostcost = fold(add,0,hostcosts);
   staff =
     set([d | c←C,Term('Dr',[d])←c.state]);
   staffcost = #staff * GPcost * #calc
in devcost + hostcost + staffcost < 1000
```

FairLoading: The Ruritanian Government expects all GPs to put in a fair and equitable amount of effort. Some Ruritanian patients register with a particular GP in a medical practice and expect to see only that GP for any consultation. However most are happy that all GPs are of similar quality and will see the next available GP when they request a consultation. Therefore, fairness is defined to ensure that the difference in the number of GP consultations is never greater then 2 from the average:

```
let doctors = [Dr(n) | Dr(n) ← state];
   consultations(dr) =
    #([1 | Consultation(dr,p) ← state,
           ?([n | Registered(dr,p)←state]=[])])
   M = [ consultations(dr) | dr <- doctors ]
in (max(M) - min(M)) ≤ 4
```

4 Design

This section outlines the LEAP support for operation specification and for animating architecture designs. It provides a specification, architectural diagram, and implementation of the case study. It also describes how LEAP supports animation and visualisation of data in terms of object diagrams. An object diagram is used to show a snapshot of a running system. A simple interface for the case study is constructed using the LEAP built-in components for constructing interactive GUIs.

4.1 Component Architecture

The goal model shown in Fig. 1 links to leaf components named `doctors`, `bookings` and `patients`. The behaviour of these components is captured in the goal model using behavioural goals together with the non-functional requirements for the overall system.

Figure 3 shows the next level decomposition of the system where the components are connected to support message-based communication. Each component is named and has a collection of ports shown as boxes on the outside of the component box. Each port is named and may be designated for input (white boxes) or output (black boxes). For example the `bookings` component has a port that handles `requests` from the `gui` component and produces `patientRequests` to the `patients` component.

Each connection between ports has an interface type that is shown as text positioned close to the connection edge. The interface type defines the messages that may be sent along the connection. For example, the connection between `bookingCommands` in `gui` and `requests` in `bookings` is labelled with the following interface:

```
interface {
  addGP(name:str):void;
  requestConsultation(name:str):void;
  next():void
}
```

The message return types indicate whether the message is *synchronous* or *asynchronous*. Most messages in the case study are asynchronous and have the return type **void**.

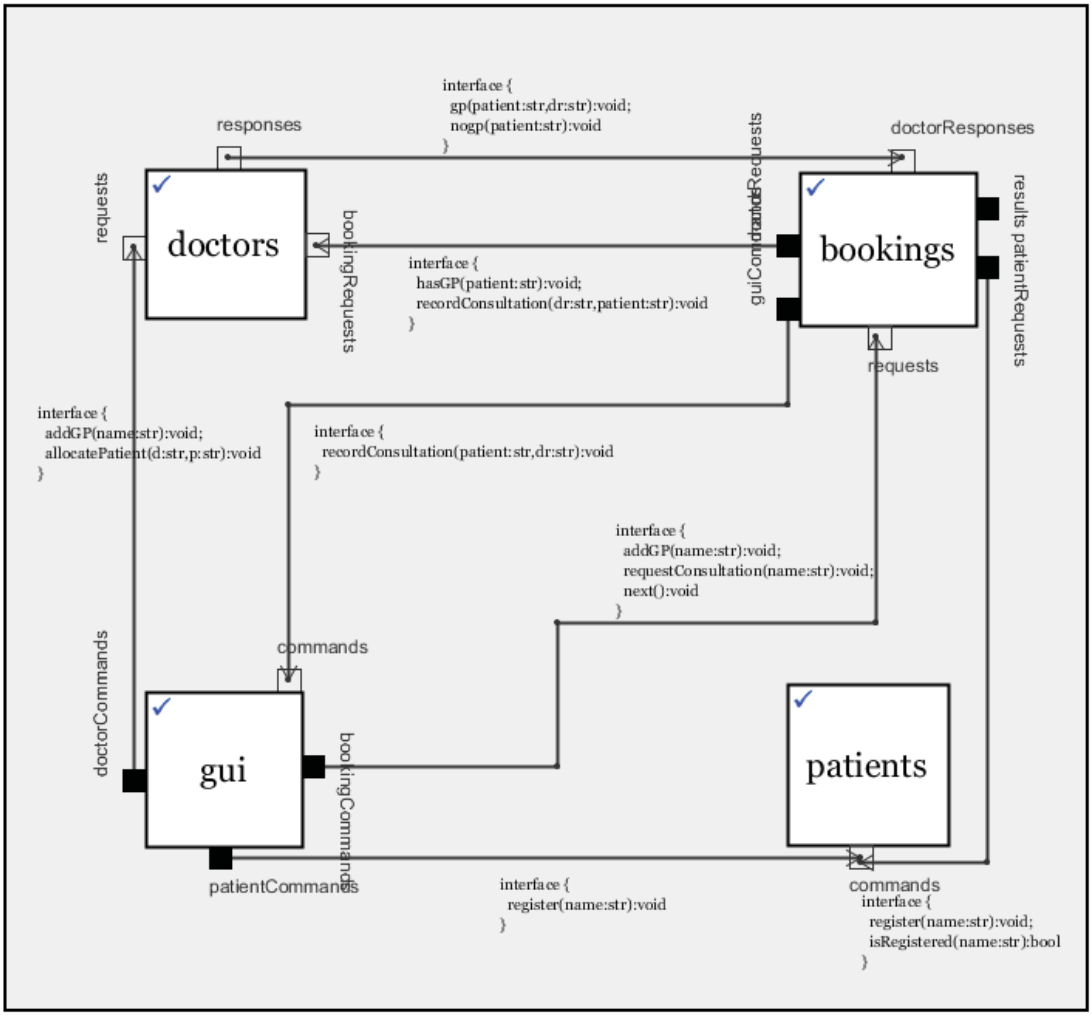

Figure 3: Surgery Component Architecture

4.2 Specifications

A goal decomposition tree should lead to the identification of a collection of components whose individual behaviours are specified by behavioural goals. The goals should identify the information content of each component and also define the messages that the components must support. The designer then has freedom to partition the messages between ports and to specify the behaviour of the components in response to each message.

LEAP provides a *specification clause* in the definition of a component that is used to specify the behaviour the component in response to messages. Individual behaviours are then simulated in LEAP using a variety of mechanisms including state machines, transition rules and operations. This section gives an overview of the specification clause in terms of the case study.

A specification clause contains a collection of message specifications that are defined using three types of sub-clause: pre-condition; post-condition; message-condition. A pre-condition is a predicate that must hold at the time the message is processed in order for the post-condition and the message-condition to hold. Both pre and

post-conditions are expressed in terms of patterns over the state of the component (although both may refer to general boolean expressions `e` via the syntax `?(e)`). The message-condition is a predicate that applies to the output ports of the component and uses pattern matching to determine message membership of the output queue.

A specification clause should follow directly from behavioural goals in the goal model. KAOS uses a similar method to determine patterns in the LTL formulas that can be ascribed to single operation calls. The following shows a fragment of the specification for the `doctors` component and defines the behaviour of the component in response to handling the `addGP` and `hasGP` messages:

```
spec {
 addGP(name:str):void {
  pre not(Dr(name))
  post Dr(name)
 }
 hasGP(patient:str):void {
  pre Registered(Dr(dr),Patient(patient))
  messages responses ← gp(patient,dr)
 }
 hasGP(patient:str):void {
  pre not(Registered(Dr(dr),Patient(patient)))
  messages responses ← nogp(patient)
 }
}
```

When refining the behaviour of a component from that imposed by goals to that provided by a specification clause, it may be necessary or useful to also refine the data model. LEAP goal models are associated with components. The model in Fig. 1 is associated with a component called `surgery` that contains the four components shown in Fig. 3. The data model for `surgery` is shown in Fig. 2 and is therefore used throughout the goal model.

We would like to refine the representation of patient bookings so that LEAP lists are used to implement a queue and therefore impose an ordering on processing the bookings for a GP. The refinement is shown in Fig. 4. It shows an example of LEAP data references that are displayed

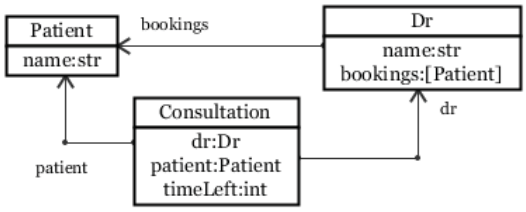

Figure 4: Patient Model

as links between classes with arrows. A reference is shown as a field and an edge, for example `dr:Dr` in `Consultation` and the edge labelled `dr`. This is because the classes are actually term-types and the field order is important, whereas the graphical representation using nodes and edges aids comprehension. A typical consultation term is:

```
Consultation(
 Dr('phibes',[Patient('fred')]),
 Patient('wilma'),3)
```

The specification clause for the `patient` component is shown below:

```
spec {
 addGP(d:str):void {
  pre not(Dr(d,_))
  post Dr(d,[])
 }
 requestConsultation(patient:str):void {
  messages doctorRequests ← hasGP(patient)
 }
 gp(p:str,dr:str):void {
  post Dr(dr,b) ?(exists Patient(p) in b)
 }
 nogp(patient:str):void {
  post Dr(d,b) and exists Patient(patient) in b
 }
 next():void {
  pre Dr(d,Patient(p):b)
      not(Consultation(Dr(d),_))
  post Dr(d,b) Consultation(Dr(d),Patient(p))
 }
}
```

4.3 Implementation

At this point in design, goals have placed behavioural and non-functional requirements on the system, the requirements have been refined into a component architecture including ports and connectors. An initial data model may have been

refined into local data models for each component and an associated specification for each of the messages that the component handles. The final step is to provide an implementation for each message.

LEAP provides three mechanisms for implementing component behaviour. Where a component exists in a number of pre-defined states and where behaviour can be conveniently defined in terms of state transitions, LEAP provides pattern directed state machines that monitor changes in the state of a component and fire transition when guards become satisfied. Our case study does not use state machines.

A less structured form of state-machine behaviour is provided in the form of rules that monitor changes in the state of a component and fire when the rule-condition is satisfied. The `bookings` component uses rules to manage consultations. Finally, a component defines a collection of named operations. Operations can be directly called from within the component and, if the operation name matches a message name, are invoked when a message is processed.

The rest of this section provides examples of the implementation of three of the `surgery` components. The GUI component is the subject of the following section.

patients: The state of a component is modified using **`new`** and **`delete`**. The **exists** operator is used to match patterns over the current state of the component:

```
component patients {
  devcost = 200
  hostcost = 10
  host = 'local hospital'
  operations {
    register(name) {
      new Patient(name)
    }
    isRegistered(name) {
      exists Patient(name)
    }
  }
}
```

doctors: The `doctors` component provides examples of the use of **`find`** that selects an element from the current state of the component that matches a given pattern. the **`replace`** ... **`with`** operator is used to replace a term that matches a pattern. The ← operator sends a message to the named port:

```
component doctors {
  devcost = 500
  hostcost = 10
  host = 'surgery'
  operations {
    addGP(name) {
      new Dr(name)
    }
    allocatePatient(dr,p) {
      new Registered(Dr(dr),Patient(p))
    }
    recordConsultation(dr,p) {
      find Consultations(Dr(dr),ps) {
        replace Consultations(Dr(dr),ps)
        with Consultations(Dr(dr),p:ps)
      } else new Consultations(Dr(dr),[p])
    }
    hasGP(p) {
      find Registered(Dr(d),Patient(p)) {
        responses ← gp(p,d)
      } else responses ← nogp(p)
    }
  }
}
```

bookings: The `bookings` component provides examples of component rules. the rule named `next` has a pattern that matches a `Dr`-term that contains a non-empty queue of waiting patients `p:ps`. The use of **`not`**`(...)` in the rule `next` requires that there is no current consultation for the GP named `d`. The body of `next` creates a new `Consultation`-term, sends a message to record the consultation, and removes the patient from the GP's waiting list.

The rules `consult` and `complete` deal with processing the consultation. The system requires a message `next` to occur in order to start any pending consultations. The rule `consult` then matches any ongoing consultations that have not reached their time limit, and increases the consultation time by 1. The `complete` rule fires when the consultation is over:

```
component bookings {
 devcost = 1000
 hostcost = 100
 host = 'surgery'
 operations {
  addGP(d) {
```

```
  new Dr(d,[])
 }
 requestConsultation(p) {
  if patientRequests.isRegistered(p)
  then doctorRequests ← hasGP(p)
  else results ← refusal(p)
 }
 gp(p,d) {
  find Dr(d,bs) {
    replace Dr(d,bs) with Dr(d,bs+[Patient(p)])
  } else new Dr(d,[Patient(p)])
 }
 nogp(p) {
  find Dr(d,bs) when
       not(exists Dr(d',bs') {#bs>#bs'}) {
   replace Dr(d,bs) with Dr(d,bs+[Patient(p)])
  } else error('cannot allocate gp to ' + p)
 }
 next() {
  new Next()
 }
}
rules {
 next:
   Dr(d,p:ps)
   not(Consultation(Dr(d,_),Patient(_),_)) {
  new Consultation(Dr(d,ps),p,5);
  guiCommands ← recordConsultation(p,d);
  replace Dr(d,p:ps) with Dr(d,ps)
 }
 consult: Next
        Consultation(Dr(d,b),Patient(p),n)
        ?(n>0) {
  delete Next;
  replace Consultation(Dr(d,b),Patient(p),n)
  with Consultation(Dr(d,b),Patient(p),n-1);
  requests ← next()
 }
 complete:
  c=Consultation(Dr(d,b),Patient(p),0) {
   delete c;
   doctorRequests ← recordConsultation(d,p);
   requests ← next()
 }
}
}
```

4.4 State

LEAP is an architecture simulation language. A simulation may be generated interactively, as described in the next section, or programmatically. Component state can be initialised directly by listing a collection of terms, or indirectly by sending components some initial messages to start the simulation. Messages may be synchronous or asynchronous. The ← operator sends a message asynchronously in which case there will often be an issue regarding the relative ordering of groups of asynchronous messages. LEAP provides the **do** construct to place an ordering on message groups: **do** { ms } **then** c sends the messages ms concurrently, but waits for all the messages to be completed before proceeding to command c. Therefore, in the following example, all the GPs are added before the patients are registered and subsequently consultation requests made:

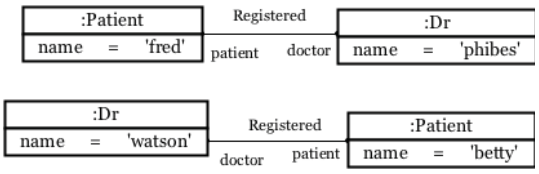

Figure 5: Registered Patients

```
do {
 bookings.requests ← addGP('phibes')
 doctors.requests ← addGP('phibes')
 bookings.requests ← addGP('who')
 doctors.requests ← addGP('who')
 bookings.requests ← addGP('watson')
 doctors.requests ← addGP('watson')
} then do {
 patients.commands ← register('fred')
 patients.commands ← register('wilma')
 patients.commands ← register('barney')
 patients.commands ← register('betty')
 patients.commands ← register('pebbles')
 patients.commands ← register('bam bam')
 doctors.requests ← allocatePatient('phibes','fred')
 doctors.requests ← allocatePatient('watson','betty')
} then do {
 bookings.requests ← requestConsultation('fred')
 bookings.requests ← requestConsultation('wilma')
 bookings.requests ← requestConsultation('betty')
 bookings.requests ← requestConsultation('barney')
 bookings.requests ← requestConsultation('pebbles')
 bookings.requests ← requestConsultation('bam bam')
} then bookings.requests <- next()
```

The state of LEAP components can be constructed or visualised via a tree-browser and using object diagrams. Terms are instances of classes and associations such as those defined in Fig. 2. Terms that are instances of classes are drawn as objects with slots; terms that are instances of associations are drawn as links. The object diagrams are a useful way of vizualising the structure of a component's state.

Figure 5 shows an example object diagram that visualises the state of the doctors component after the allocatePatient messages have

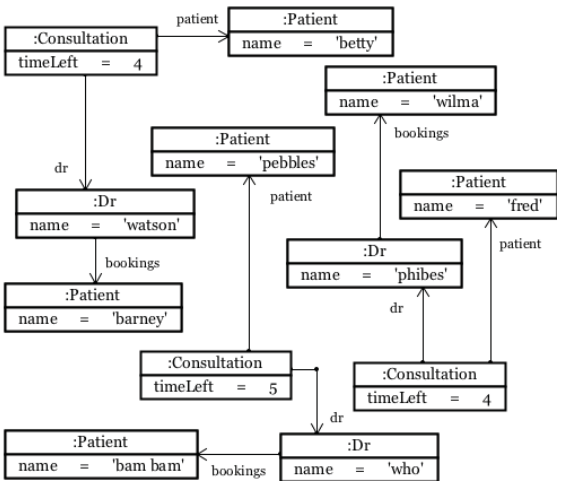

Figure 6: Bookings

been processed. Figure 6 shows an object diagram that visualises the state of the `bookings` component after all the consultation requests have been processed and the simulation has been started via the booking-rules. notice that field references in Fig. 6 are shows as directed links whereas association instances in Fig. 5 are shown as undirected links.

4.5 GUI

LEAP provides a language for constructing simple graphical user interfaces in order to interact with a simulation. The language uses a term representation for a display defined as follows:

```
d ::=
  Table([[d,...],...]) // tables of elements.
| Text(s)              // text.
| Input(s,s)          // a text input field.
| Button(s,f)         // a button with a label.
```

where `s` is a string and `f` is a closure. Each input field is named. When a button is pressed, it is supplied with a table that contains all input names with their associated values.

Figure 7 shows the LEAP tool running the case study. The panel labelled `LEAP` shows a browser view on all components loaded into the tool. The panel labelled `leapsrc` shows a view onto the file system containing leap source code, the file `surgery.cmp` has been selected and its source code is shown in the middle panel on the right. The upper right panel shows the case study user interface. The rest of this section describes how the user interface is defined as a LEAP component.

The surgery `gui` component makes use of a general LEAP feature whereby a user-defined Java class can be dynamically loaded into the system and participate as a LEAP component. The LEAP operator `jcomponent` loads a Java class. The class must implement a predefined LEAP interface that allows it to participate in message passing. The `GUI` class provides an input port `in` that can be send a `display` message. The outline of `gui` is as follows:

```
component gui {
  display = jcomponent('frames.GUI')
  show() {
    display.in ←
      display(Table([[manage()],...]))
  }
  // definitions of ports and operations...
}
```

The definition of the `manage` operation shows how user input is handled for adding a new GP:

```
manage() {
  // Set up a table for managing the surgery
  Table([[Text('Manage')],
         [Table([manageDoctors(),
             managePatients(),
             manageBookings()])]])
}

manageDoctors() {
 // Return a table row.
 // Include dummy text for padding...
 [Text('Name'),
  Input('name',''),
  Text(''),
  Text(''),
  // A button function has a single argument
  // a table containing all input fields...
  Button('New GP',fun(e) addGP(e.name))]
}

addGP(name) {
  // The GUI keeps track of a GP's state...
  new GP(name,Free);
  // Send messages to the other components...
  doctorCommands ← addGP(name);
  bookingCommands ← addGP(name);
  // Update the display...
  show()
}
```

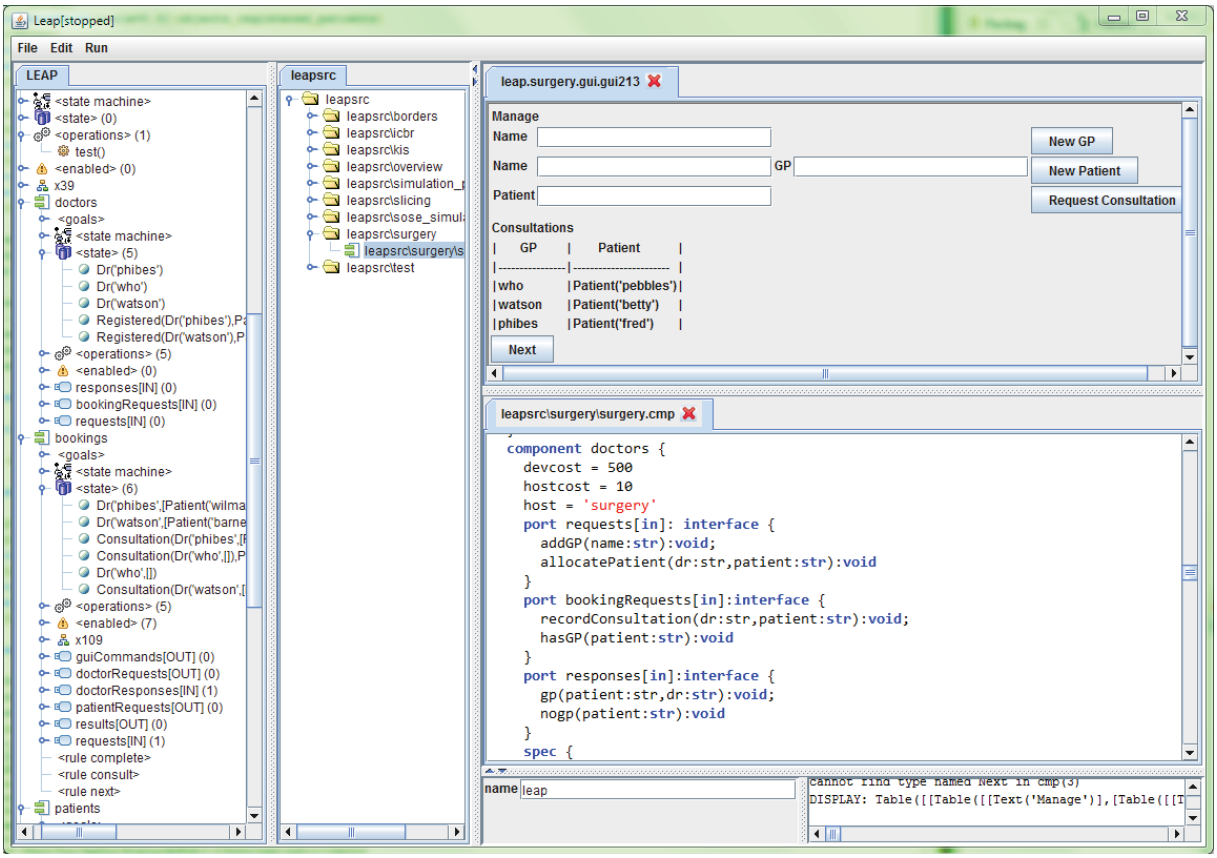

Figure 7: LEAP Showing Case Study Simulation

5 Semantics for Goal Models

We have shown how goal modelling can lead to an architecture model using LEAP. The goals are identified as informal, behavioural and non-functional. Our approach is based on other goal-driven approaches, most notably KAOS. However, whilst KAOS proposes LTL as a language for expressing behavioural goals, it does not provide a link to a language with operational semantics and it does not define how non-functional goals should be precisely expressed. our proposition is that non-functional goals are predicates over meta-properties of system models.

This section describes how LEAP goal models can be given a precise semantics in terms of behavioural and non-functional goals. We define the LEAP value domain and outline the LEAP operational cycle in section 5.1. Section 5.2 describes how *reification* is performed whereby LEAP values at the Java-level are translated automatically into LEAP data at the user-level in order to support meta-constraints. Finally, behavioural goals are written in a LTL that is defined in section 5.3 in terms of the value domain.

5.1 Values

Figure 8 shows the value model for LEAP. Developers can use the LEAP tool to produce diagrams of type: goal; model; component; state; state machine, or can develop their models using the LEAP textual language and then transform the models into diagrams for visualisation.

The execution semantics for LEAP continually removes messages on the queue of component input ports. The messages are matched against

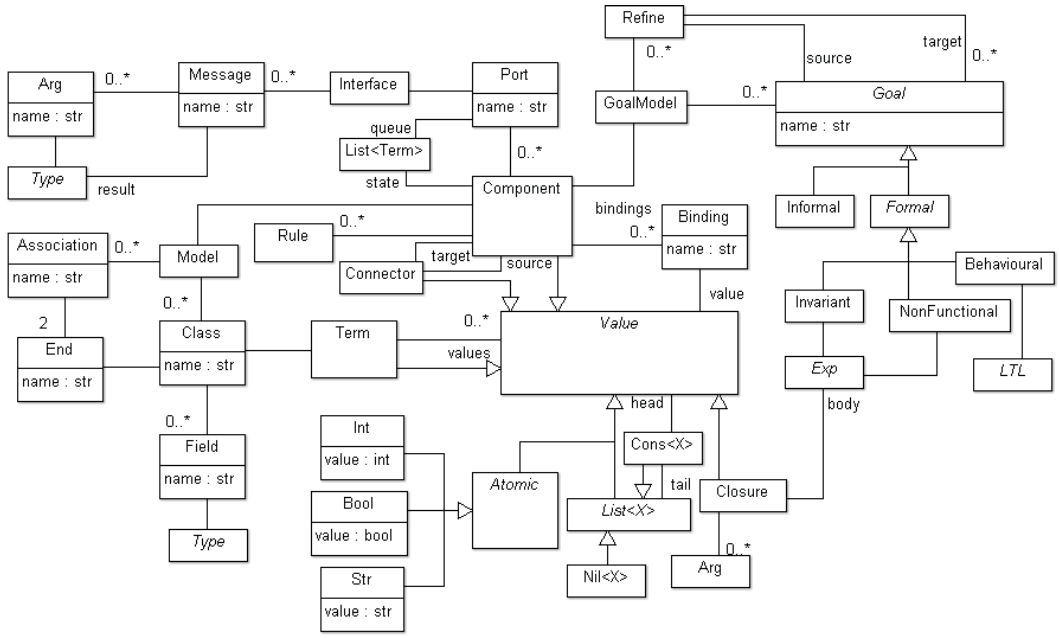

Figure 8: LEAP Values

closures bound by the component and handled by calling the closure with the supplied arguments in the message. The body of the closure is performed and may send messages to the output ports of the component. Messages sent to an output port are transferred to the message queue of any connected input ports. Rules continually monitor the state in a component and when the rule condition matches, the rule body fires.

5.2 Reification

Non-functional goals are predicates over meta-data. LEAP provides access to meta-data using two operators: `reify` and `intern` that are inverses of each other. The `reify` operator maps any LEAP data value to a LEAP term and the `intern` operator maps a suitably encoded term into a LEAP value. A simple example is:

```
reify(10) → Int('values',10)
intern(Int('values',10)) → 10
```

In general, reification of a LEAP value produces a term whose type name is the name of the underlying Java class. The first data element in the term is the name of the Java package containing the class followed by the values of the Java fields in a predefined order. The implementation of `reify` and `intern` relies on underlying Java reflection as described in the rest of this section.

In order for a Java class to participate in the reification process it must provide a Java annotation of type `Descriptor`. A descriptor names the fields that will be included as term-data and names the Java methods to be used as accessors and updaters for the fields. Crucially, the descriptor places an order on the fields:

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Descriptor {
  String[] accessors();
  String[] updaters();
  String[] fields();
}
```

All LEAP value classes provide descriptors. The following shows part of the `Association` class and its descriptor annotation:

```
@Descriptor(fields={"name","ends"},
            accessors={"getName","getEnds"},
```

```
            updaters={"setName","setEnds"})
class Association {
  String name;
  End[] ends;
  public Association() {}
  public String getName() { return name; }
  public void setName(String n) { name = n; }
  ...
}
```

Mapping between different data representations relies on three meta-operations: instantiation; `getContents`; `setContents`.

Instantiation is directly supported by Java through the `newInstance` method of a class and is used by LEAP in a standard way by providing a 0-arity constructor for each value-class. Accessing the contents of a Java object is achieved through the accessors listed in the descriptor:

```
Object[] getContents() {
  Class<?> c = getClass();
  String[] names = getAccessors(c);
  Object[] contents = new Object[names.length];
  for (int i = 0; i < names.length; i++) {
    String name = names[i];
    Method accessor = c.getMethod(name);
    contents[i] = accessor.invoke(this);
  }
  return contents;
}
```

The `getAccessors` method above transitively retrieves the descriptor annotations of the receiver's class and its super-classes and returns the names of the accessors. Updating a Java object is achieved through `setContents`:

```
void setContents(Object[] os) {
  Class<?> c = getClass();
  String[] U = getUpdaters(c);
  for (int i = 0; i < U.length; i++) {
    String u = updaterNames[i];
    Method um = c.getMethod(a);
    Object o = coerceValue(os[i], argType(um));
    um.invoke(this, o);
  }
}
```

The method `coerceValue` is used to ensure that LEAP values are mapped to Java values, for example LEAP lists are mapped to Java arrays or vectors depending on the type of the field.

Having defined the meta-access machinery we can now define the reification operations. the `reify` method maps a Java object `o` to a LEAP value. Note that the following code has been simplified by omitting the machinery that deals with cyclic data. The `reify` method has three categories of mapping: objects whose class has a descriptor; atomic values; collections. As shown below, the atomic cases are dealt with by directly translating to instances of `Int`, `Bool` and `Str`.

```
Value reify(Object o) {
  Class<?> c = o.getClass();
  else if(c.getAnnotation(Descriptor.class)!=null)
    return reifyDescriptor(o);
  else if(o instanceof Integer)
    return new Int((Integer)o);
  ... // more atomic cases...
  else if(c.isArray())
    return reifyArray((Object[])o);
  ... // also deal with vector...
  else error(...)
}
```

An object whose class has a descriptor is mapped using `reifyDescriptor` defined below. The value is encoded as a LEAP term and the fields, as defined by the descriptor, are recursively reified:

```
Value reifyDescriptor(Object o) {
  String type = o.getClass().getName();
  int dot = type.lastIndexOf('.');
  String packageName = type.substring(0,dot);
  String typeName = type.substring(dot+1);
  Object[] contents = value.getContents();
  Value[] subTs=new Value[contents.length+1];
  subTs[0] = new Str(packageName);
  for (int i = 1; i < contents.length; i++)
    subTs[i + 1] = reify(contents[i]);
  return new Term(TERMCLASS, typeName, subTs);
}
```

An array is reified to become a LEAP list as follows (vectors are treated in the same way):

```
List<Value> liftArray(Object[] objects) {
  List<Value> list = new Nil<Value>();
  for (int i = objects.length - 1; i >= 0; i--)
    list = list.cons(reify(objects[i]));
  return list;
}
```

The `intern` method performs the inverse mapping. Like `reify` is has three categories of translation: terms, atoms and lists:

```
Object intern(Value value, Class<?> type) {
  if (value instanceof Term)
    return internTerm((Term) value, type);
  else if (value instanceof Int)
    return ((Int) value).getValue();
  ... // more atomic cases
  else if (value instanceof List<?>)
    return internList((List<Value>)value,type);
  else eror(...)
```

```
p ::= behavioural constraint
  always { p }
| eventually { p }
| next { p }
| before(n) { p }
| forall t { p }
| p implies p
| not(p)
| p and p
| t

t ::= term patterns
  N(t*)
| k
| t:t
| v
```

Figure 9: Behavioural Constraints

Translation of terms is shown below. The name of the Java class to be instantiated is encoded as a class name and package name in the term. These are extracted and a new Java instance is created. The other data elements in the term are extracted and recursively translated before setting the values in the new Java object using `setContents`:

```
Value internTerm(Term term, Class<?> type) {
  String typeName = term.getName();
  Object[] contents = term.getContents();
  Str str = (Str) contents[0];
  String pname = str.getValue()+"."+typeName;
  Class<?> c = getClass().forName(pname);
  Value v = (Value) c.newInstance();
  Object[] vs = new Object[contents.length-1];
  Class<?>[] ts = v.getTypes();
  for (int i = 0; i < types.length; i++)
   vs[i]=intern((Value)contents[i+1],ts[i]);
  v.setContents(vs);
  return v;
}
```

5.3 Behavioural Goals

Behavioural goals are written in a formal language that is based on *Linear Temporal Logic* (LTL). The syntax of LEAP LTL is shown in Fig. 9 where `N` denotes a term name, `k` is an atomic constant, and `v` is a variable. The semantics of a behavioural goal are defined with respect to system executions. A system execution is a sequence of states where a state is a set of terms as defined by Fig. 8. Although the state of a LEAP model involves multiple components, ports, messages and terms, it is possible to simplify this by equating messages (which are just terms anyway) with states and by flattening the structure of the components (renaming consistently where necessary).

A LEAP system state is a set of terms s. A system execution is a sequence of states $[s_1, \ldots, s_n]$. At any given time the system is in a particular state i and has a history $[s_1, \ldots, s_{i-1}]$ and a future $[s_{i+1}, \ldots, s_n]$.

A term pattern t contains variables. Variables are bound to LEAP values in an environment θ. An environment is applied to a term pattern $\theta(t)$ to produce a term by substituting the variables in the pattern for values. A pattern t occurs in a state s when there is a substitution θ such that $\theta(t) \in s$. Note that a pattern may occur more than once in a state if there is more than one substitution.

A LTL formula p holds at a given point i in a system execution $[s_1, \ldots, s_n]$ when the following relationship holds: $[s_1, \ldots, s_n], i \models p$. The relationship is defined in Fig. 10. The definitions are as follows: (1) defines that P always holds when it holds for all future states; (2) defines that p eventually becomes true when there is a future state for which is true; (3) defines when p is true in the next state; (4) states that **before**$(n)\{p\}$ holds when p is true in the past by skipping back n states into the history; (5) requires that p must hold for all possible elements in the current state that matches t; (6) states that if p holds in the current state then q must also hold; (7) defined that if **not**(p) holds then p must not hold; (8) states that for p **and** q to hold then p and q must both holds in the current state; (9) allows the variables in a term pattern to be existentially qualified in the current state.

6 Related Work

This section provides an overview of related work in the provision of modelling languages and frameworks that have attempted to address the early stages of requirements engineering.

(1)	$[s_1, \ldots, s_n], i \models \textbf{always}\ \{\ p\ \}$	*when* $[s_1, \ldots, s_n], j \models p$ holds $\forall j \geq i$
(2)	$[s_1, \ldots, s_n], i \models \textbf{eventually}\ \{\ p\ \}$	*when* $[s_1, \ldots, s_n], j \models p$ holds $\exists j \geq i$
(3)	$[s_1, \ldots, s_n], i \models \textbf{next}\ \{\ p\ \}$	*when* $[s_1, \ldots, s_n], i + 1 \models p$ holds
(4)	$[s_1, \ldots, s_n], i \models \textbf{before}(n)\ \{\ p\ \}$	*when* $[s_1, \ldots, s_n], i - n \models p$ holds
(5)	$[s_1, \ldots, s_n], i \models \textbf{forall}\ t\ \{\ p\ \}$	*when* $\forall \theta.\theta(t) \in s_i \implies [s_1, \ldots, s_n], i \models \theta(p)$
(6)	$[s_1, \ldots, s_n], i \models p\ \textbf{implies}\ q$	*whenever* $[s_1, \ldots, s_n], i \models p$ *then* $[s_1, \ldots, s_n], i \models q$
(7)	$[s_1, \ldots, s_n], i \models \textbf{not}(p)$	*when* $[s_1, \ldots, s_n], i \not\models p$
(8)	$[s_1, \ldots, s_n], i \models p\ \textbf{and}\ q$	*when* $[s_1, \ldots, s_n], i \models p$ *and* $[s_1, \ldots, s_n], i \models q$
(9)	$[s_1, \ldots, s_n], i \models t$	*when* $\exists \theta.\theta(t) \in s_i$

Figure 10: LTL Semantics

Requirements modelling is an intrinsic and important element of the processes by which system architectures are designed, implemented and managed. However, architecture modelling approaches and techniques, such as design-by-contract, have until recently focused on what a system should do and how it can be achieved. Scant attention has been paid to the "why", the motivations in terms of goals, requirements, rationales. In (Wagter et al. 2012) Wagter et al make an interesting distinction between 'blueprint' styles exemplified by the engineering based approaches such as Zachman (Zachman 1999), TOGAF (Spencer et al. 2004) and Archimate (Lankhorst 2009) and argue that such a blueprint style does not suffice as interests such as stakeholders, informal power structures and other hard-to-quantify factors cannot be easily represented in such an engineering style. They propose instead, that a yellow-print style (as noted by De Caluwe and Vermaak 2003) is necessary.

Efforts to standardise on the motivational or intentional aspects of enterprise architecture have been consolidated in the OMG Business Motivational Model (BMM) (Group et al. 2005) which provides a structure for representing concepts for developing, communicating and managing business plans such that they can be used to model those factors that motivate a busines plan, the elements making up the business plan and the relationship between these factors and elements. The BMM provides a focus for *motivation* such that activities delivering a business plan are defined by *why* specific activities are performed. Concepts in the BMM include:

Ends: the aspirations of the enterprise expressed as Vision, Goals and Objectives.

Means: the mechanism by which Ends are realised and expressed as Mission in terms of Strategy and Tactics; and Directives such as business policy and business rules.

Influencers: how ends and means can influence each other in either positive or negative ways.

The BMM provides a meta model (syntax only) expressed as a UML model that shows these concepts, their subtyping and their relationships. The model recognises that the complexity of the relationship between BMM model elements and process modelling is not fully developed and proposes that a related standard the Business Process Modelling Notation (BPMN) (BPMN 2.0. Notation 2009) should be used as an additional technology. This raises questions about model integration, consistency and shared semantics issues.

The foundational work for BMM can be traced back to goal oriented requirements engineering (GORE) techniques (Mylopoulos et al. 1999) such as i* (Yu 1997; Yu and Mylopoulos 1994) and KAOS (Dardenne et al. 1993; Letier and Van Lamsweerde 2004; Van Lamsweerde 2008). GORE bases techniques present a variety of options for analysis such as providing a more formal basis of how goals realise other goals, conflict between goals and the positive and negative contributions goals make to other goals. Further, the relationship between the proposed solution and actual

need is more clearly delineated. So organisations can more easily and in a systemised manner, articulate choices between alternative configurations of architectures or explore new possible configurations (Halleux et al. 2009; Jonkers et al. 2004). GORE techniques also have potential in that they can be readily combined with viewpoints oriented requirements engineering approaches such as (Finkelstein et al. 1991; Sommerville and Sawyer 1997).

The Reference Model for Open Distributed Processing (RM-ODP) is a comprehensive framework for open systems specification. It is an ISO/ITU Standard (ITU, 1996) that defines a framework for architecture specification of large distributed systems using viewpoints on a system and its environment: enterprise, information, computation, engineering and technology. The theoretical basis of the RM-ODP model resides in object oriented principles and service oriented specification and the mapping of the levels to implementation objects (Raymond 1995). RM-ODP addresses the motivation or intention aspects by the "enterprise viewpoint" and the inclusion of a concept of "Objective" in its enterprise language. The concept is then related to key elements of the enterprise viewpoint including "community" and the objects belonging to the community such as policy, role and process definition. As Almeida et al point out: "In a nutshell, communities are defined to achieve certain objectives. These objectives influence the definition of the policies and roles in the community, which affect the behaviour of the enterprise objects to favour the satisfaction of community objectives" (Almeida et al. 2010). In the same paper, RM-ODP concepts are interpreted using the Unified Foundation Ontology (Guizzardi et al. 2008) to provide a basis for communication and consensus particularly to address the social behaviour dimension of how the use and change of enterprise objects can strive towards motivation and intentions behind business goals.

The i* framework provides a set of concepts for modelling and analysis addressing the early requirements phase, namely, that focusing on the "why" of underlying systems requirements (Yu and Mylopoulos 1994). Key concepts in the framework are centred around the notion of the *intentional actor* that possesses intentional properties such as a belief or ability. Actors collaborate and depend upon each other and collectively perform tasks and in the process of doing so, consume resources. Actors will acknowledge and adapt so that opportunities and threats are addressed in line with the intentional beliefs. Actors are thus part of an agent-oriented system. The i* framework has a provision for two types of models. Firstly, a Strategic Dependency model that is effectively an Actor diagram that is used to model agreements between actors to fulfil a goal, perform a task or to use a resource. Types of goal - hard and soft (for which there is no clear criteria to be fulfilled) can be represented. The Strategic Rationale model is a means of expressing stakeholder interests and concerns and their relationship to various configurations of an enterprise architecture. The model is effectively a drill down of the SD model and describes the actors' goals and rationales in order to justify the actors' relationships and their adoption of particular plans. Related and directly derived from i* is the Tropos methodology (Bresciani et al. 2004; Susi et al. 2005) which adopts the same concepts for the early requirements stage. The ARIS methodology (Scheer 2000) is an approach that is widely used in industry for business process modelling and also includes a high level set of goal-related concepts that include such elements as Objective, Participant, Critical Factor and Function as a means of modelling intentions. Objectives are used to represent a notion of a goal and Functions can be seen as operations applied to objectives for the purpose of supporting goals. Relationships between goals are also supported. The Critical Factor concept represents aspects which need to be considered in the meeting of a particular objective. The limitations of ARIS with respect to the richness of the modelling language for goals and the lack of process modelling capabilities within Tropos and it's parent i* has

been identified as a requirement for some form of integration between the two approaches and Cardosa et al propose a semantics based integration between goals and process modelling as one such approach for provisioning that requirements (Cardoso et al. 2010).

One major strand of research in goal modelling is KAOS methodology (Dardenne et al. 1993) and subsequently elaborated further by Letier, Van Lamsweerde and others (Letier and Van Lamsweerde 2004; Van Lamsweerde 2000, 2008; Van Lamsweerde et al. 1998). Consistent with the technologies described previously, KAOS is also an agent oriented methodology for requirements engineering. The key concept is a goal as a "a prescriptive statement of intent that the system should satisfy through cooperation of its agents". Goals are defined at varying levels of abstraction through refinement relationships. Goals are specifically satisfied by a system component and are termed a *requirement*, however there is support for a partial satisfaction via an *expection* relationship. These are not enforceable via automated processing. KAOS, like i* and others supports the notion of conflict modelling by *obstruction* and *resolution* by other goals. Perhaps different from others, KAOS allows the modelling of domain hypotheses represented by domain invariants - properties (and values) that are always hold.

Quartel et al (Quartel et al. 2009) provide a useful overview of some of the technologies described here and also make the observations: that BMM cannot be considered a true requirements modelling language; i* while providing a rich expressive language presents on overhead in learning and using the language; KAOS lacks some of the richness of expressivity but counters that shortcoming in its simplicity. In considering these observations they propose a language called ARMOR which provides a goal/intentional modelling capability to the Archimate language and is thus similar to our proposal outlined in this paper. We argue that ARMOR provides both an abstract and concrete syntax but relies on the limited semantics provided by Archimate. In contrast our integrated offering of goal modelling support within the LEAP language provides intentional modelling with semantics. The language offered here as part of LEAP has a similar expressive power in that most of the pertinent aspects of i* are available. We have also tried to optimise the usability available in KAOS and provide the more advanced facilities of a simulation environment. The semantics offered by the use of LTL further enhances the capability of the language. As Cardosa et al have pointed out, semantics based integration between goals and process modelling (the integration of the why and how) are a necessary step. LEAP and its goal modelling element provides that capability.

7 Conclusion

The motivation or goal behind why a particular requirement manifests itself as system function and the traceability of the relationship that explores the *why* remains an area of relative neglect in the system and enterprise architecture domains. Technologies originating as characterisations of goal oriented requirements engineering such as KAOS and i* could perhaps have had limited impact on architecture modelling because of issues such as: complexity leading to usability, lack of semantics and a traceable rigour from goals through to system functions. In particular, non-functional requirements present specific difficulties. In this paper we have proposed a technology LEAP that attempts to address some of these issues. This contribution exists at several levels. Firstly we have provided a formal model for goal modelling that is supported by a technology that allows goals to be checked during execution. Secondly, the executability of requirements presents an opportunity to provide semantics for goal model executability by the use of LTL. Finally by provisioning a tight meta model based integration between concepts that reside at the early stages of requirements analysis with those that are focussed on the engineering aspects of architecture we provide a route from goal models to architecture models that is supported by a prototype modelling and execution environment.

The LEAP approach is based on the hypothesis that architecture design and analysis should use a small and orthogonal set of concepts based around higher-order components. A component-based language can be used to represent both logical and physical elements of a system. We have demonstrated success with this approach in terms of intentional modelling, goal and IT alignment, architecture simulation, representation of both event-driven and service-oriented approaches to architecture, refinement, and architecture refactoring.

However, there are limitations with the current LEAP technology that we have not yet addressed. It cannot represent low-level architectural designs such as those required by embedded and real-time systems. We acknowledge that the current support in LEAP for specifying behaviour in terms of invariants, pre and post-conditions and LTL expressions is insufficient to express complex component interactions involving time and/or concurrency.

LEAP has been implemented and therefore has an operational semantics given by its implementation, our next steps will include an abstract semantic description of LEAP that integrates with the LTL semantics given in this article that will allow us to study issues such as the complexity of execution and therefore the limitations on the sise of LEAP models. The usability aspects of LEAP tooling, including debugging complex configurations of higher-order components, has yet to be addressed. Part of the strength of LEAP is that it is based on a relatively small number of orthogonal concepts and provides a richly expressive language through higher-order features. However, in order to be usable it is necessary for developers to be able to make distinctions between different categories of elements, for example goals that apply to the system and those that apply to the actors that use the system. A form of domain-specific syntactic sugar may be a suitable way to support these distinctions.

Another area that we feel will be fruitful, is using LEAP as a migration tool from an as-is architecture to a to-be architecture by using the Java interfaces of LEAP to simulate the to-be architecture in terms of the as-is architecture and thereby providing a basis for incrementally replacing the LEAP simulation with new components.

Our proposal for goal modelling has been evaluated with an experiment using a restrictive but representative case study example. We note the limitations of such experiment and as a consequence further research will continue to validate our approach as we attempt to use our tools to evaluate new case studies from both existing literature and from industrial practitioners. We are now engaged in a relationship with leading technology research lab from the commercial sector who will be using LEAP as part of their research activity. We expect this relationship to provide new evaluatory data to support the proposal presented in this article.

References

Almeida J., Cardoso E., Guizzardi G. (2010) On the Goal Domain in the RM-ODP Enterprise Language: An Initial Appraisal based on a Foundational Ontology. In: Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International. IEEE, pp. 382–390

Bresciani P., Perini A., Giorgini P., Giunchiglia F., Mylopoulos J. (2004) Tropos: An agent-oriented software development methodology. In: Autonomous Agents and Multi-Agent Systems 8(3), pp. 203–236

Cardoso E., Santos Jr P., Almeida J., Guizzardi R., Guizzardi G. (2010) Semantic Integration of Goal and Business Process Modeling. In: IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2010), Natal-RN, Brazil

Chan Y., Reich B. (2007) IT alignment: what have we learned? In: Journal of Information Technology 22(4), pp. 297–315

Clark T., Barn B. S. (2011) Event driven architecture modelling and simulation. In: Gao J. Z., Lu X., Younas M., Zhu H. (eds.) SOSE. IEEE, pp. 43–54

Clark T., Barn B. S. (2012) A common basis for modelling service-oriented and event-driven architecture. In: Aggarwal S. K., Prabhakar T. V., Varma V., Padmanabhuni S. (eds.) ISEC. ACM, pp. 23–32

Clark T., Barn B. S., Oussena S. (2011) LEAP: a precise lightweight framework for enterprise architecture. In: Bahulkar A., Kesavasamy K., Prabhakar T. V., Shroff G. (eds.) ISEC. ACM, pp. 85–94

Dardenne A., Van Lamsweerde A., Fickas S. (1993) Goal-directed requirements acquisition. In: Science of computer programming 20(1-2), pp. 3–50

De Caluwe L., Vermaak H. (2003) Learning to change: A guide for organization change agents. Sage Publications, Inc

Finkelstein A., Goedicke M., Kramer J., Niskier C. (1991) Viewpoint oriented software development: Methods and viewpoints in requirements engineering. In: Algebraic Methods II: Theory, Tools and Applications, pp. 29–54

Group B. et al. (2005) The business motivation model-business governance in a volatile world. Technical report

Guizzardi G., Falbo R., Guizzardi R. (2008) Grounding software domain ontologies in the unified foundational ontology (ufo): The case of the ode software process ontology. In: 1th Iberoamerican Workshop on Requirements Engineering and Software Environments (IDEAS 2008)

Halleux P., Mathieu L., Andersson B. (2009) A method to support the alignment of business models and goal models. In: Proceedings of BUSITAL 8, p. 121

Jonkers H., Lankhorst M., Van Buuren R., Hoppenbrouwers S., Bonsangue M., Van Der Torre L. (2004) Concepts for modeling enterprise architectures. In: International Journal of Cooperative Information Systems 13(3), pp. 257–287

Lankhorst M. M, H.A. Proper H., Jonkers J. (2010) The Anatomy of the ArchiMate Language. In: International Journal of Information System Modeling and Design 1(1), pp. 397–409

Lankhorst M. (2009) Introduction to Enterprise Architecture. In: Enterprise Architecture at Work. The Enterprise Engineering Series. Springer Berlin Heidelberg http://dx.doi.org/10.1007/978-3-642-01310-2_1

Letier E., Van Lamsweerde A. (2004) Reasoning about partial goal satisfaction for requirements and design engineering. In: ACM SIGSOFT Software Engineering Notes. 6 Vol. 29. ACM, pp. 53–62

McLean E., Soden J. (1977) Strategic planning for MIS. John Wiley & Sons Inc

Mylopoulos J., Chung L., Yu E. (1999) From object-oriented to goal-oriented requirements analysis. In: Communications of the ACM 42(1), pp. 31–37

OMG BPMN 2.0. Notation 2009. In: Object Management Group, p. 496

Quartel D., Engelsman W., Jonkers H., Van Sinderen M. (2009) A goal-oriented requirements modelling language for enterprise architecture. In: Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International. Ieee, pp. 3–13

Raymond K. (1995) Reference model of open distributed processing (RM-ODP): Introduction. In: IFIP TC6 International Conference on Open Distributed Processing. Brisbane, Australia, Chapman and Hall, pp. 3–14

Scheer A. (2000) ARIS–business process modeling. Springer Verlag

Sommerville I., Sawyer P. (1997) Viewpoints: principles, problems and a practical approach to requirements engineering. In: Annals of Software Engineering 3(1), pp. 101–130

Spencer J. et al. (2004) TOGAF Enterprise Edition Version 8.1

Susi A., Perini A., Mylopoulos J., Giorgini P. (2005) The tropos metamodel and its use. In: INFORMATICA-LJUBLJANA- 29(4), p. 401

Van Lamsweerde A. (2000) Requirements engineering in the year 00: A research perspective.

In: Proceedings of the 22nd international conference on Software engineering. Acm, pp. 5–19

Van Lamsweerde A. (2008) Requirements engineering: from craft to discipline. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. ACM, pp. 238–249

Van Lamsweerde A., Darimont R., Letier E. (1998) Managing conflicts in goal-driven requirements engineering. In: Software Engineering, IEEE Transactions on 24(11), pp. 908–926

Wagter R., Proper H. A. E., Witte D. (2012) A Practice-Based Framework for Enterprise Coherence. In: Practice-Driven Research on Enterprise Transformation. Lecture Notes in Business Information Processing Vol. 120, pp. 77–95

Yu E. (1997) Towards modelling and reasoning support for early-phase requirements engineering. In: Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on. IEEE, pp. 226–235

Yu E., Mylopoulos J. (1994) Understanding ẃhyín software process modelling, analysis, and design. In: Proceedings of the 16th international conference on Software engineering. IEEE Computer Society Press, pp. 159–168

Zachman J. (1999) A framework for information systems architecture. In: IBM systems journal 38(2/3)

Tony Clark, Balbir Barn

Middlesex University
The Burroughs
London
NW4 4BT
United Kingdom
{t.n.clark | b.barn}@mdx.ac.uk

Marco Kuhrmann, Georg Kalus, Alexander Knapp

Rapid Prototyping for Domain-specific Languages

From Stakeholder Analyses to Modelling Tools

Today, modelling is a widely acisecepted technique in Software Engineering (SE). Many problems can be expressed using general-purpose modelling languages such as the UML. For more specific problems, the definition of a specialised domain-specific language (DSL) may be required. The definition of a domain-specific language is a time-consuming task that requires knowledge in (modelling) language design, deep understanding of the domain and, to be useful and usable, user assistance and tool support. In this paper, we present an approach to derive a domain-specific language from the description of instances of the domain under consideration: Stakeholders describe model instances from which the metamodel (the DSL) and a suitable modelling tool are derived automatically. We describe a tool that we used to experiment with this approach, its current state and the future work.

1 Introduction

A 'model' has several advantages over free-from sketches, as it has some degree of syntax and semantics and that it can be used to generate or derive other artefacts from it. However, defining a modelling language and corresponding modelling tools is laborious. If models are used mainly to clarify a particular domain, i.e., while analysing a customer's domain or a project's requirements, defining an appropriate modelling language is often not worth the effort. In addition, stakeholders not familiar with (or not interested in) modelling languages may not see the immediate benefit of the investment. A statement by a tool vendor highlights this problem: 'Nobody wants to perform *real* modelling, but only drawing pictures…' And in fact, tools such as Microsoft Visio or Omni Group's OmniGraffle, and even PowerPoint can be regarded as some of the most popular general-purpose 'modelling' tools.

On the other side of the spectrum, powerful modelling tools were developed: The Unified Modeling Language (UML) became a standardised modelling language and notation, and various UML dialects, such as SPEM (OMG 2008) or BPMN (OMG 2010), were created. Also, for certain domains specialised and comprehensive modelling approaches (including formalisms, notations, and tools) were developed, e.g., ARIS (Davis, R. 2010) or Focus (Schätz, B. 2001). Such dialects and specialised modelling approaches are well suited for their particular domain.

A popular approach for creating precise, specialised and easy-to-understand modelling languages are *domain-specific languages* (DSL). A domain-specific language facilitates (1) easy communication by using well-known and accepted domain objects in an appropriate notation, while keeping the (2) precision that enables further processing of the domain models, e.g., to generate code, data models, and so on. A domain-specific language is usually designed according to specific domain requirements and, therefore, a concrete domain-specific language is difficult to apply in other environments than the one it was developed for. The development of a specialised domain-specific language that may be used in just one project therefore may not be economically feasible as long as rapid language development — similar to rapid prototyping — is not well supported for domain-specific language development.

1.1 Problem Statement

Providing stakeholders, i.e., analysts, designers, and other project roles with appropriate modelling languages and tools beyond standard solutions is a challenging and costly undertaking. Domain-specific languages are a way to define special-purpose modelling languages. Current tools to develop DSLs, such as Eclipse EMF/GMF, Meta-Case, or the Visual Studio DSL-Tools require deep understanding of the tool itself and conceptual and technical knowledge. For specific problems, the effort necessary to develop a suitable domain-specific language therefore often seems too high compared to the potential benefit. The benefits of domain-specific languages could, however, be leveraged if there were means to quickly and iteratively develop a domain-specific language, similar to the rapid prototyping development paradigm.

1.2 Contribution

At the ICSE 2011 workshop on 'Flexible Modeling Tools' (Kuhrmann 2011) we discussed an early idea about how to make the language development process for domain-specific languages easier. The core of this idea was to interactively develop a domain-specific language by deriving it from an exemplary instance, which was 'drawn' during a stakeholder workshop.

In the paper at hands we take this idea one step further by reporting on an implementation of a DSL-based platform which generates a concrete domain-specific language from an instance-model scribbled on a 'virtual white board'. The instance modeler works in a drag-and-drop-style, similar to free-form drawing tools such as Microsoft Visio. The result is more than just pictures, but a concrete modelling language.

1.3 Outline

The remainder of the paper is organised as follows: In Sect. 2 we discuss related work, especially with regard to modelling of domain-specific languages and corresponding tools. In Sect. 3 we briefly describe the 'traditional' domain-specific language development process using our DSL-platform. In Sect. 4 we present our understanding of instance modelling and its impact to domain-specific language design. The presented approach is applied to a small case study in Sect. 5. Continuing with Sect. 6, we summarise the extended domain-specific language development method that facilitates rapid, instance-based language design. We conclude the paper in Sect. 7 and formulate the need for further research tasks.

2 Related Work

The field of modelling and meta-modelling is too wide to be covered exhaustively here. We therefore focus on basic concepts, current tools for meta-modelling and domain-specific language design, and new emerging ideas w.r.t. the advancement of meta-modelling.

Domain-specific Languages

We can roughly distinguish between the *general-purpose* approach, such as the UML (OMG 2011b), specific techniques for certain domains, e.g., Focus (Schätz, B. 2001), and the concept of *domain-specific languages* somewhere in between (Fowler and Parsons 2010; Kleppe, A. 2008). A domain-specific language is, essentially, a metamodel, which can be discussed from different perspectives. With regards to language design, some good definitions can be found in (Cook et al. 2007). We understand a metamodel to be a formalism to describe (domain-specific) languages, which can be understood by a computer.

Standard Meta-modelling Tools

In Eclipse-based language modelling tools (Steinberg et al. 2008), metamodels are represented by so-called Ecore models, which are based on the OMGs Meta Object Facility (MOF) hierarchy (OMG 2011a). The definition of a metamodel (a domain-specific language) is done using a UML-like notation subset. For instance, the Eclipse

Modeling Framework (EMF) provides rich support for the definition of metamodels (Steinberg et al. 2008), which is shown by many concrete EMF-based languages (e.g., the OMME tools (Folz and Jablonski 2010) and the considerable number of concrete EMF-based tools). Another example for such a tool is the Meta Case environment (MetaCase: Company's homepage and samples). Such a comprehensive support is important for language engineers to adjust all aspects of a modelling language (structure and semantics). However, for quickly capturing a domain, many of the powerful features are not required. The same can be said about the Microsoft Visual Studio DSL-Tools (Cook et al. 2007; Greenfield and Short 2004), which we used to develop PDE (Kuhrmann et al. 2010b). The Visual Studio DSL-Tools are not based on UML but also use a structured, XML-based approach to define data models and offer the possibility to add semantics and behaviour using source code.

New Ideas in Meta-modelling

The development of modelling languages and modelling environments is a widely discussed topic. Kimelman and Herschman (2011), for instance, discuss the need for ways to support formal modelling tools in a flexible manner. Similar to Cho et al. (2012), they argue the design of a modelling language should not be regarded as a Waterfall-like process, and highlight the necessity to dynamically move forth and back between informal (free) and formal models. Challenges resulting from this view are discussed by Cho et al. (2011). They discuss an approach that captures model instances by demonstration, and note that most domain-specific modelling is initially done using 'creativity' tools such as word processors, drawing tools or presentation tools. They conclude that the creation process of a domain-specific language has to take into account that (1) free form shapes have to be formalised, that (2) a metamodel needs to be formalised from model instances, and that (3) captured model instances have to be enriched by semantics.

We were facing similar challenges when designing DSL-based meta-modelling tools (Kuhrmann 2011). In Cho et al. (2012) a first implementation of the concept described in Cho et al. (2011) is presented. This implementation results, however, in a 'drawing' tool. Beyond sketches and ideas, Volz et al. (2011) present a multi-layer modelling environment that not only supports meta-modelling but also the connection of models at different levels of abstraction. They motivate their approach with the observation that users often use different tools and that a solution for bridging the gap could be to integrate all models using one modelling language and creating connections among the models.

Discussion

The design of a domain-specific language needs support in at least two areas: (1) to provide language 'end users' with a modelling tool that supports them in creating and handling concrete model instances and (2) to assist language engineers during the definition of a domain-specific language.

Almost all DSL tools address the first aspect. Eclipse EMF/GMF for instance supports textual as well as visual domain-specific languages and provides corresponding Eclipse-integrated editors. With our work on PDE we followed an alternative approach where an editor is generated from the domain-specific language. The domain-specific language is 'baked into' the resulting editor. The end users are provided with a specific modelling tool (stand-alone or IDE-hosted) according to their needs (Kuhrmann et al. 2010b).

Comprehensive support for language engineers to define a domain-specific language is only partially addressed. Eclipse, MetaEdit, and Visual Studio provide comprehensive support for the language engineers if the domain of action is known and analysed. If the language engineer should capture a domain and derive a domain-specific language, no adequate support is given — especially for scenarios such as a domain analysis workshop, which is done with the stakeholders.

3 PDE Language & Tool Development

A result of the research described here is the Process Development Environment (PDE). The platform is published as an Open Source project and can be accessed at `http://pde.codeplex.com`. PDE provides an infrastructure for the design of domain-specific languages in general, and process languages and process authoring tools in particular. The core functionality is based on the Microsoft Visual Studio DSL-Tools (Cook et al. 2007). PDE adds several features, such as:

- Improved visual design
- Model visualisation
- Metamodel modularisation
- Hooks for validation functions

3.1 The PDE Platform

The PDE framework consists (Fig. 2) of two parts: The first part is PDELanguage, which is an extension of the Visual Studio DSL-Tools. The second part is PDE Framework that serves as the shell for the editors that are generated from a domain-specific language. The PDE Framework consists of a ToolFramework and a concrete PDE-based language that is an instance of the PDELanguage (Fig. 1). Such a language consists of a ViewModel and a DomainModel. The DomainModel is the executable language in which functions, such as validation or serialisation are configured. A concrete PDE-based language might contain different view models, e.g., a tree view, a graphical editing pane, or a property grid. The ToolFramework is a comprehensive *Windows Presentation Foundation*-based application frame, which uses the .NET Framework. Using the MEF interfaces (Managed Extensibility Framework 2010) the application frame is extendable, i.e., by new views extending the ViewModel or additional functionality provided by separate plug-ins.

A concrete language (a metamodel) is a domain-specific language, which is based on the PDE extension of the Visual Studio DSL-Tools. The PDE Language, which is itself a 'meta-meta model[1]' (Fig. 1), is the basis for the metamodel, which is merged with the PDE Tool Framework into a concrete modelling tool (stand-alone or hosted in Microsoft Visual Studio) for language engineers or modelers respectively. A comprehensive description of PDE can be found in (Kuhrmann et al. 2010a).

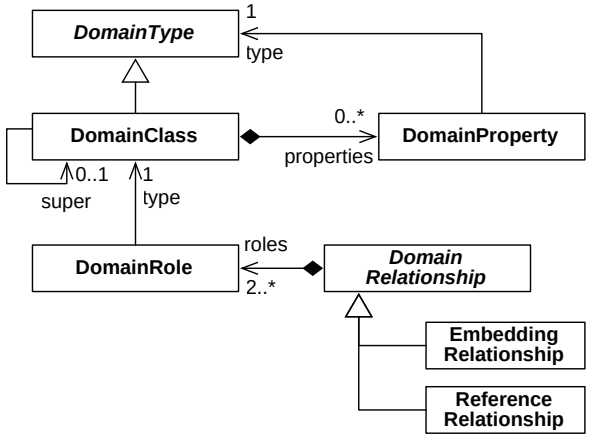

Figure 1: Concept meta model of PDE-based languages.

3.2 Creating a PDE-based Language

In the following we describe the typical language development process using PDE. This process is usually gone through only once per metamodel. However, if the metamodel is updated, parts of the process have to be repeated to incorporate the changes. Figure 3 shows the (classic) language development process consisting of up to five steps.

Step 0 In the step 'PDE Language Development' the PDE Language itself can be manipulated or extended. This step influences the behaviour of the whole platform. It should not be done without deep knowledge of the platform. For a language engineer who just wants to define a new domain-specific language for a customer, this step is usually unnecessary.

[1]Note that the meaning of meta-meta model is aligned with the definition provided by the MOF (OMG 2011a).

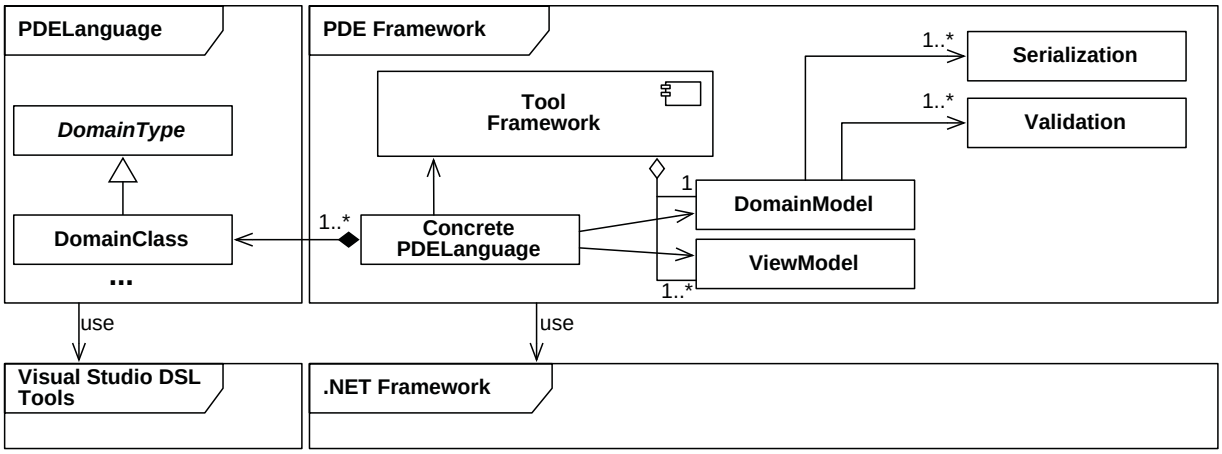

Figure 2: Architecture of the PDE platform (simplified).

Step 1 The step 'Implement a Metamodel' involves the implementation of the metamodel (the domain-specific language) within the specification of the PDE Language. Elements and relationships of the domain-specific language are transformed into *domain classes* and *domain relationships* of the concrete metamodel. Additionally, the domain-specific language can be extended to provide multiple views (graphical notations) to present different aspects of the model.

Step 2 In the step 'Transform' the transformation process of the platform utilises T4 (Sych 2007) templates to generate the source code representing the domain-specific language for integration in the PDE Editor Framework.

Step 3 The third step covers two aspects: the extension (step 'Extend') and the customisation (step 'Customise') of the transformed domain-specific language. In this step the generated code can be extended or customised for different purposes, i.e., serialisation methods can be overridden to define a custom serialisation format or new validation methods can be provided to check for specific constraints. New views etc. can also be provided in this step, i.e., sophisticated views that combine certain aspects of the underlying model to ease the modelling or to foster the discussion with stakeholders.

Step 4 Step 4 is the last step in which the final editor is created. Depending on the audience of the editor and the initial PDE-template, either a stand-alone tool is created, or an editor, which is hosted in the Visual Studio environment.

The language development process is designed to allow short development cycles if a domain-specific language needs to be changed and evaluated (as discussed by Kuhrmann et al. (2010b) and demanded by Cho et al. (2012)). Besides the above listed steps there is a standard path for the language development consisting of the steps 1, 2, and 4.

Step 3 can be skipped if the features that are initially provided by the platform fit (almost) all requirements. In this step additional components can be introduced to the tool, i.e., specialised visualisation components or additional logic.

Besides this static binding of additional components, PDE also provides a MEF-based (Managed Extensibility Framework 2010) plugin interface to discover and load separately developed plugins at runtime, e.g., additional logic for runtime validation, or features that cannot or can only

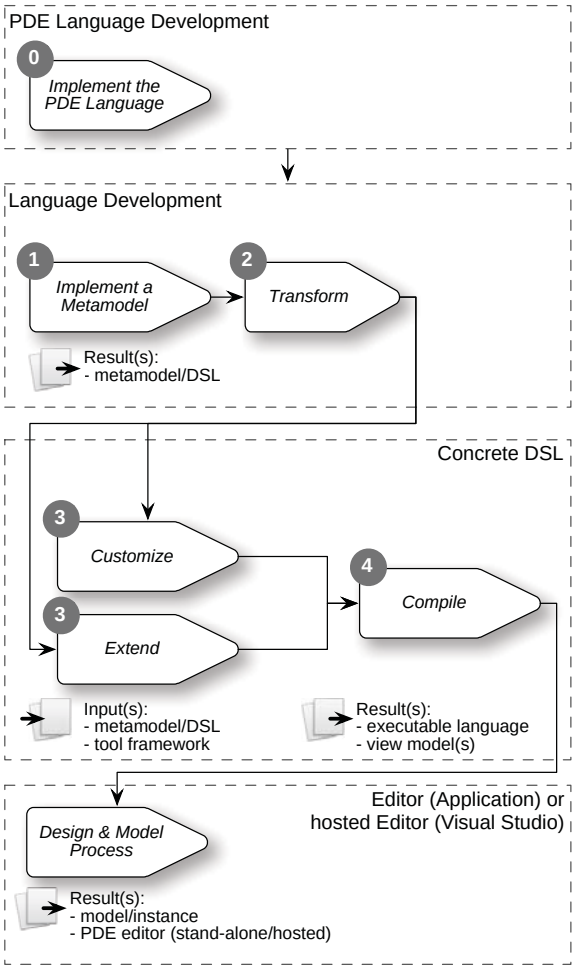

Figure 3: The language development process for creating a new domain-specific language using PDE.

hardly be expressed in the domain-specific language itself.

4 Instance Modelling

The outcome of the aforementioned language development process are the modelling language (for the engineers/designers) itself and corresponding modelling tools (for engineers and authors), which are either a stand-alone or a Visual Studio-hosted tool. The experiences with assistants like the built-in one of Visual Studio showed that the PDE-based process does make the language definition easier, especially if a comprehensive editor for models based on the language should be available.

However, as the PDE Language designer is integrated with Visual Studio, its application requires some technical understanding. The language designer component is far away from being easy to understand for non-expert stakeholders and supports the definition of a domain-specific language on a fairly technical level.

4.1 Instance Modelling — The Idea

In a cooperative and iterative modelling approach creative and formal tasks overlap to a certain extent (Cho et al. 2012; Kuhrmann 2011) — especially if a new domain needs to be 'explored' in a stakeholder workshop.

Therefore, the idea of *instance modelling* can be described as follows: A stakeholder workshop to understand and capture the domain is done 'as usual' — but instead of using a classical whiteboard, a digital 'informal' modelling pane is used. This pane collects domain entities, which are represented visually, and simple associations. In the workshop, entities can be collected, structured, combined, and so on.

The goal is to express the domain using prototypical model instances as representatives for the domain under consideration. Stakeholders are able to describe the domain from their perspective and experience. Thus, instance modelling currently aims at complementing the elaboration of a new domain-specific modelling language in cases where the domain still has to be explored. Changes to existing domain-specific modelling languages in the sense of model evolution have not been considered.

4.2 The Approach

In an instance modelling workshop, a prototypical instance of the envisioned domain-specific modelling language is drawn. A domain element, like a role or an artefact, is captured by an *element* class with some *attribute* slots filled; relationships between domain elements, like a role

being responsible for an artefact, are either represented by a *reference*, expressing a directed connection, or by an *embedding*, expressing a whole-part relationship (cf. Fig. 4, step 1).

Behind the scenes, the language-modelling tool captures the model prototype and translates into (or derives) PDELanguage constructs to prepare the definition of the target domain-specific language. The domain-specific language is mostly the result of the informal design and builds the basis to rapidly create the new DSL.

PDE infers the language using the following mapping:

- Element class ↦ DomainClass
- Reference ↦ ReferenceRelationship
- Embedding ↦ EmbeddingRelationship

Furthermore, attributes associated with elements are mapped to DomainProperties; also a set of similar properties can be mapped to one domain property, e.g., a property 'Name'. Primitive types, e.g., Int or String, are directly mapped to predefined domain types of PDE.

The modelling process is triggered by user interactions that are caught by the ViewModel (Fig. 2). The editor realises, e.g., drag and drop events and the framework calls methods that, for instance, create new domain entities. Figure 4 shows an example: When dragging an image onto the modelling pane (step 2 of Fig. 4) the ViewModel catches the assigned event and creates a new instance of a DomainClass (Fig. 1). Having added the new entity to the instance model, the language engineer can edit the entity, e.g., editing name, adding attributes, create relationships to other entities, and so on.

The resulting domain-specific language can be used to create or generate various tools, in particular, for modelling. Currently, only a PDE export is implemented, but other meta-modelling tools, such as EMF/GMF, could be targeted. Therefore, an appropriate SerialisationFormatter has to be added to extend the PDE Framework (Fig. 2).

The stakeholders use the resulting modelling tools. The style of modelling, the notation and the semantics comply with the drafts made during the language creation workshops.

4.3 DSL Optimisation

So far, we only changed the 'input channel' for the design of the domain (see language development process in Fig. 3, step 1). One could say, this is just another front end to the PDE Language designer. However, we have to take into account that we consider two different ways of creating domain-specific languages:

1. The first option to create a domain-specific language—creating a language based on solid information gathered beforehand—works fine if the language engineer has knowledge about the domain. In consequence a frequent interaction with the stakeholders might be unnecessary, and the design of a domain-specific language is close to, e.g., UML architecture design.
2. The second way to create a domain-specific language is a more 'exploratory' approach. It is applied in settings, where language engineers need to get initial information about the considered domain. At this point instance modelling replaces domain analysis workshops by an interactive design of the domain, represented by exemplary instances.

Figure 4 gives an idea of the second scenario of designing a domain-specific language: A language engineer asks a stakeholder for a domain entity, i.e., a role, drags a picture that symbolises this entity onto the modelling pane, and starts to refine this entity (i.e., adding attributes) during the interview with the stakeholder. From a technical point of view, this way is rather 'pragmatic' and does not result in an optimal domain-specific language.

Therefore, optimisation is performed before finalising the language. PDE analyses those captured domain entities, derives domain types, and analyses them for optimisation opportunities, e.g.,

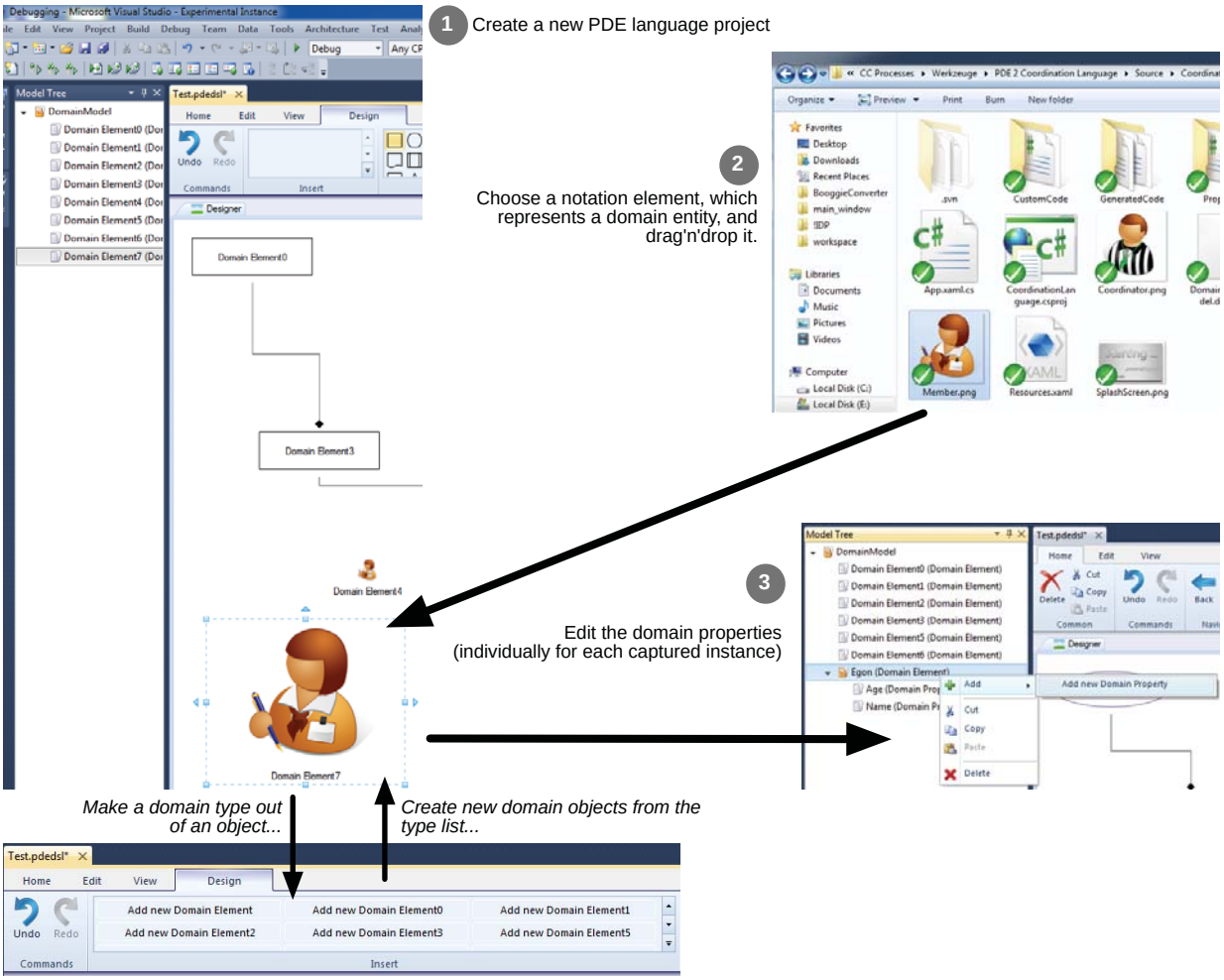

Figure 4: Concept of instance design: A PDE-based Visual Studio-hosted editor pane is used to capture model instances. A model explorer allows for the precise definition of types, domain properties, and, finally, for the derivation of a concrete modelling language.

common attributes that can be extracted into a base class. This approach is based on refactoring (Fowler et al. 1999) and is applied on tentative languages. Currently, the platform allows for optimisations triggered by properties and relationships. Figure 5 shows two examples: In the left part of the figure a first optimisation strategy is shown. Starting with a number of designed domain entities, PDE analyses those entities for potentially 'sharable' attributes. The analysis criteria are the name of an attribute as well as its domain type. If there were any attributes meeting those criteria, PDE asks, whether those should be refactored using a shared base class for the shared attribute. In the second optimisation opportunity, PDE analyses the captured model for similarities of relationships (Fig. 5, depicted on the right). Reference relationships are deemed similar if their roles are named equally and are of the same type. If the platform finds candidates, it asks, whether a shared base class should be created that realises the extracted relationship.

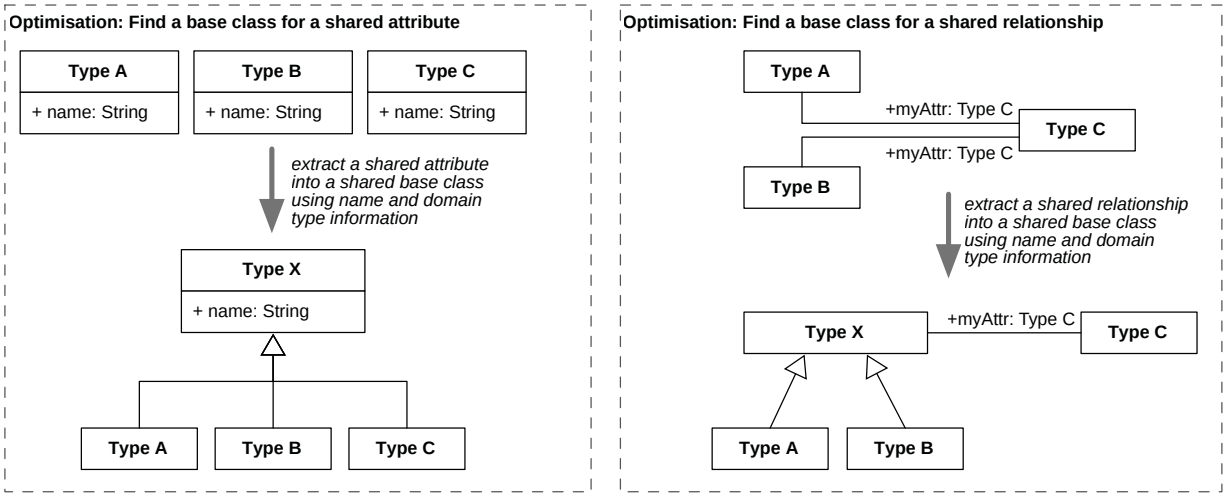

Figure 5: Language optimisation: A captured instance is analysed and optimised in order to create a 'real' domain-specific language. The platform proposes possible optimisations.

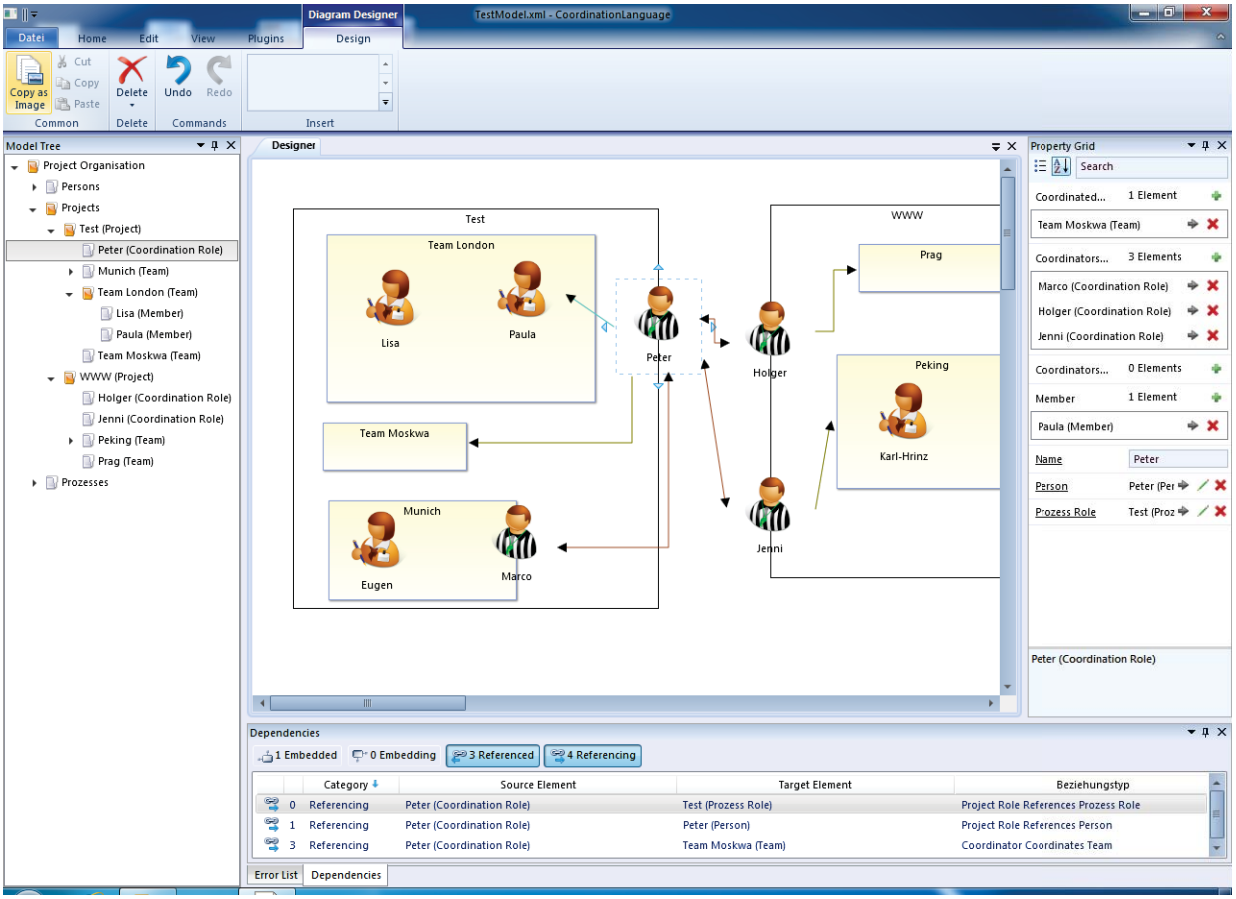

Figure 6: The classically designed DSL and the generated tool according to (Kuhrmann et al. 2010b).

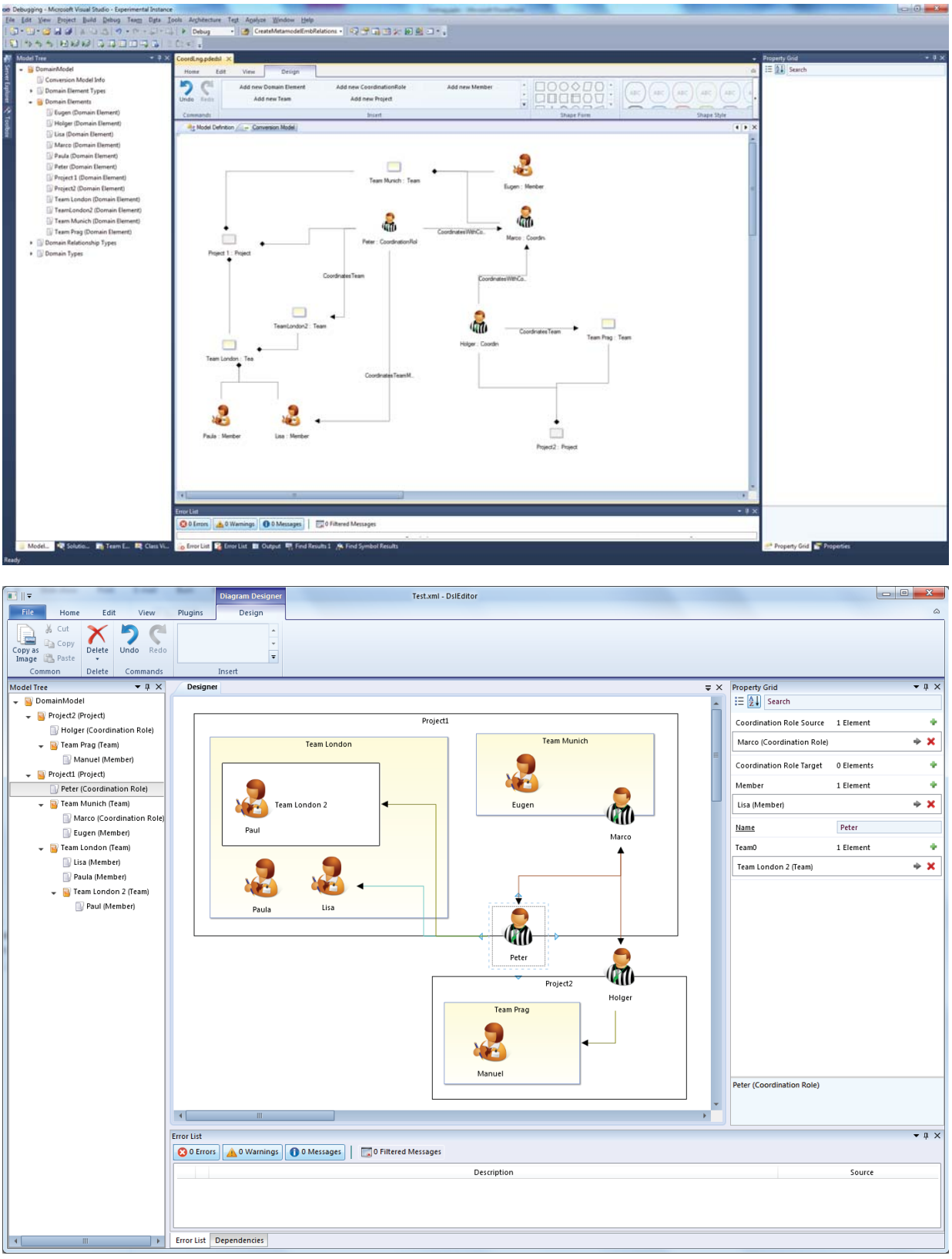

Figure 7: The modeled instance that derives the team modelling DSL (upper part) and the generated editor for the team modelling DSL (lower part).

5 Case Study

Since our research was exploratory, we opted for a case that gave us a 'reference point': In (Kuhrmann et al. 2010b) we discussed PDE using a small domain-specific language for modelling teams at different sites in a globally distributed development project. For the realisation of the instance modeler the primary requirement was that the inferred domain-specific language (including the resulting modelling tools) was at least as powerful as the classically designed one.

Figure 6 shows the editor and an exemplary model of the so-called *team coordination language*[2], which was developed using the 'classic' PDE DSL development process (cf. Fig. 3). According to our key requirement, a domain-specific language which is created based on the instance modelling approach has to contain all the domain types, the domain attributes, and so on. Furthermore, the resulting editor should have the same appearance as the 'classic editor'. Consequently the new editor has to open and read concrete models that were designed using the old editor.

Figure 7 shows in the upper part the Visual Studio-hosted instance modeler and a captured instance of a team model, including different sites, relationships, and different kinds of team members. According to the aforementioned development process (instance capturing, optimisation), the instance modeler add-on to PDE infers a domain-specific language from the instance. The inferred language is the input (see Fig. 3) for the generation of an editor (lower part of Fig. 7). The selected simple case shows that the key requirement was completely achieved. The domain-specific language that was created using the instance modelling approach was, finally, a 'clone' of the originally designed one. Even the models, which were created with the old editor, could be opened with the new editor.

[2]Based on the keynote 'Speculations on Coordination Models' by Len Bass at the International Conference on Global Software Engineering in Princeton, 2010.

6 Extended Language Development

Based on instance modelling we re-define the language development process. Figure 4 shows a series of screenshots that illustrate the extension. Beside the 'classic' approach as described in Sect. 3.2 a language engineer can open a new PDE-DSL-Project (1). The editor pane is hosted in a Visual Studio environment.

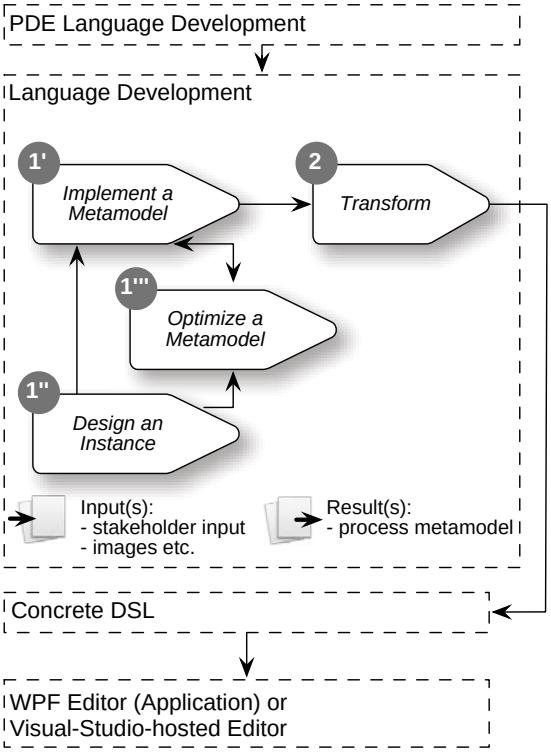

Figure 8: Extended language development process w.r.t. instance modelling.

To create a new graphical language element (domain type), the language engineer only needs to drag and drop, e.g., an image onto the pane (2). The picture immediately becomes a domain type to which corresponding attributes can be added (3). Furthermore, the domain type is also placed in the design ribbon and can be used to create new domain objects of the type. Having drawn the instance of interest, the mechanism described in Sect. 4 comes into play to (1) generate a PDE-based metamodel, and to (2) optimise the inferred metamodel.

Figure 8 shows the modification of the language development process, which was described in Fig. 3. Instead of simply modelling a domain-specific language, another way to create a domain-specific language is added. The first step is now to directly *Implement a Metamodel* (step 1′) **or** to *Design an Instance* (step 1″), which is transferred into a domain-specific language. Additionally, step 1‴ *Optimise a Metamodel* can be executed to optimise a designed or a directly implemented metamodel. The outcome of those steps is, itself, input for the transformation step, which leads to the classical language development process.

7 Conclusion & Future Work

We presented an extension to our PDE platform for instance modelling to capture domain models and to transform them into a domain-specific language. The user-centered part of domain modelling is similar to approaches known from drawing tools, such as Microsoft Visio and therefore easier to learn and understand for non-technophile stakeholders. The PDE platform creates domain-specific languages from such drawn figures in the background and provides language engineers with some refactoring-like optimisation capabilities.

Summarised, the instance modelling extension allows users to draw a figure of the currently considered domain, and creates a domain-specific language from the drawing.

In (Kuhrmann 2011) we already discussed first ideas, concepts, and prototypes. We also discussed some challenges — similar to Cho et al. (2011) — e.g., semantics of pictures, language derivation and respective mappings, or structuring of complex languages. We furthermore discussed, if 'the modelling pane is just another domain-specific language' and 'and to what extent a user-defined domain-specific language can be derived automatically?'

Currently, we have a first prototype that allows to derive a domain-specific language from one particular instance (Sect. 5). Also, we decided to realise the PDE extension as a domain-specific language, too. The mechanism behind the prototype is, therefore, a model transformation at the PIM to PIM level (Kleppe et al. 2003).

Future Work

This paper outlined some (promising) steps to support rapid language design. In ongoing research we have first to improve the capabilities of domain capturing and the language derivation techniques. Here, we need to extend our prototype to be able to extract a domain-specific language from different instances instead of only one. Furthermore we have to improve the usability. Although the user interface is already quite simple and easy to understand, even for non-technophile stakeholders, the working process is, still, complex and requires expertise. Without guidance of a PDE-trained language engineer 'ordinary' users are certainly not able to perform domain capturing.

Finally, we validated our idea against only a few key requirements and provided a proof of concept. We need, however, to intensively evaluate the feasibility and the economic impacts of our ideas to answer the questions: Is the instance modelling approach faster? Is the quality of the generated (and optimised) domain-specific languages comparable to those that are developed the classic way? How much money can be saved using this approach? To this end, the most recent release of the PDE platform, including the instance modelling add-on, is available at `http://pde.codeplex.com` to everybody for testing and evaluation purposes.

Acknowledgments

We want to thank Eugen Wachtel and Manuel Then for their essential support during the development of PDE. We also thank Sebastian Eder for reviewing this paper.

References

Cho H., Gray J., Sun Y., White J. (2011) Key Challenges for Modeling Language Creation By Demonstration. In: ICSE Wsh. Flexible Modeling Tools

Cho H., Gray J., Syriani E. (2012) Creating Visual Domain-Specific Modeling Languages from End-User Demonstration. In: Int. Wsh. Modelling in Software Engineering (MiSE)

Cook S., Jones G., Kent S., Wills A. C. (2007) Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley

Davis, R. (2010) ARIS Design Platform: Advanced Process Modeling and Administration. Springer

Folz B., Jablonski S. (2010) OMME — A Flexible Modeling Environment. In: SPLASH Wsh. Flexible Modeling Tools

Fowler M., Beck K., Brant J., Opdyke W., Roberts D. (1999) Refactoring: Improving the Design of Existing Code. Addison-Wesley

Fowler M., Parsons R. (2010) Domain-Specific Languages. Addison Wesley

Greenfield J., Short K. (2004) Software Factories. Wiley & Sons

Kimelman D., Herschman K. (2011) A Spectrum of Flexibility – Lowering Barriers to Modeling Tool Adoption. In: ICSE Wsh. Flexible Modeling Tools

Kleppe, A. (2008) Software Language Engineering. Addison-Wesley

Kleppe A., Bast W., Warmer J. B. (2003) MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley

Kuhrmann M., Kalus G., Then M., Wachtel E. (2010a) From Design to Tools: Process Modeling and Enactment with PDE and PET. In: ASE Wsh. Academic Software Development Tools and Techniques (WASDeTT-3)

Kuhrmann M., Kalus G., Wachtel E., Broy M. (2010b) Visual Process Model Design using Domain-specific Languages. In: SPLASH Wsh. Flexible Modeling Tools

Kuhrmann M. (2011) User Assistance during Domain-specific Language Design. In: ICSE Wsh. Flexible Modeling Tools

Managed Extensibility Framework. Online: http://mef.codeplex.com/

MetaCase: Company's homepage and samples. http://www.metacase.com

OMG (2008) Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0. Specification. Object Management Group

OMG (2010) Business Process Model and Notation (BPMN) Version 2.0. Specification. Object Management Group

OMG (2011a) Meta Object Facility (MOF) Core Specification Version 2.4.1. Specification. Object Management Group

OMG (2011b) Unified Modeling Language (UML): Superstructure Version 2.4.1. Specification. Object Management Group

Schätz, B. (2001) The ODL Operation Definition Language and the Autofocus/Quest Application Framework AQUA. Tech. Rep. TUM-I0111. Technische Universität München

Steinberg D., Budinsky F., Paternostro M., Merks E. (2008) EMF: Eclipse Modeling Framework, 2nd ed. Addison-Wesley

Sych O. (2007) T4: Text Template Transformation Toolkit. http://www.olegsych.com/2007/12/text-template-transformation-toolkit/

Volz B., Zeising M., Jablonski S. (2011) The Open Meta Modeling Environment. In: ICSE Wsh. Flexible Modeling Tools

Marco Kuhrmann, Georg Kalus
Technische Universität München
Faculty of Informatics
Boltzmanntstr. 3
85748 Garching
{kuhrmann | kalus}@in.tum.de

Alexander Knapp
Universität Augsburg
Institute of Informatics
Universitätsstr. 6a
86159 Augsburg
knapp@informatik.uni-augsburg.de

Rafael Accorsi and Raimundas Matulevičius

Workshop on Security in Business Processes

A workshop report

The Workshop on Security in Business Processes (SBP'12) was organised in conjunction with the 10th international conference on Business Process Management (BPM 2012). Over 25 attended the workshop to present and discuss 11 papers, the insights they offered and the issues they raised. During one-day workshop, a number of important and emerging issues towards the security in business processes. These were the perspectives of secure business processes, security and compliance, security and Internet services, and engineering secure business processes.

1 Introduction

Despite the growing demand for compliant business processes, security and privacy incidents caused by erroneous workflow specification, implementation and execution are still omnipresent. In fact, often business process management and security issues stand out as separate silos, and are seldom addressed together towards the development of trustworthy, compliant business processes. By combining the successful, past experiences of the First International Workshop on Alignment of the Business Process and Security Modelling (ABPSM'11) and WfSAC - BPM Workshop on Workflow Security Audit and Certification, the Joint Workshop on Security in Business Processes (SBP'12) brought together researchers and practitioners interested in security management of business process models in process-aware information systems.

SBP'12 was co-located with the 10th International Conference on Business Process Management (BPM), which, in 2012, was held in Estonian capital, Tallinn. Over 25 participants attended SBP and enjoyed a day of intensive interaction with other attendees; academics working in the business process and security modelling, practitioners reporting on their experience in using techniques to develop security concerns in business processes.

The SBP'12 workshop received 18 high quality submissions, thereby being one of the most successful, attracting and competitive workshops at the BPM. The programme committee worked very hard to select 5 full papers (acceptance rate 27,78%) and 4 short papers (total acceptance rate 50%) for the presentation at the workshop. Additionally the workshop organizers invited two keynote speakers, representing academia and industry, respectively. This report briefly summarises the papers that were presented. The proceedings containing the papers in full are available from Springer[1].

2 Perspectives of Security Business Processes

The workshop was begun with the keynote speech given by Andreas Opdahl, a professor of Information Systems Development at the University of Bergen, Norway. In his talk *Identifying and Visualising Dependendablity Concerns - Application to Business Process Management,* Andreas reported on the requirements for security project, which had the purpose to evaluate techniques for visualization of security and safety early in the planning of new information systems. Misuse case

[1] La Rosa M., Soffer P. (Eds): Business Process Management Workshops. BPM 2012 International Workshops, Tallinn, Estonia, LNBIP 132, Springer, Heidelberg, 2013.

maps and misuse sequence diagrams were the modelling languages proposed by the project for dealing with security requirements and for depicting attack sequences. Regarding the safety aspect, the project analyzed the failure sequencing diagrams. It was noted that the safety and security techniques are closely related since they both identify what the new system should not do. This resulted in the method for combined harm assessment of safety and security for information systems. The project suggests a number of implications regarding the business process area. For instance, it is important to consider, broader, and handle dependability links, involvement of broad competence, visualisation of key concerns, guidance using the keywords, and investigation of remedies for possible vulnerabilities. This could challenge potentially a broad future research regarding dependability and its various types.

In *A Language for Multi-Perspective Modelling of IT Security: Objectives and Analysis of Requirements* by Anat Goldstein and Ulrich Frank, Anat Goldstein highlighted the importance (i) to assess and reduce risks that originate from within organisation and from its outside, (ii) to overcome and manage the increasing organisational complexity, (iii) to encourage participation of non-technical actors, (iv) to relate security solutions with the cost-benefit analysis, and (v) to design and implement security infrastructure using automated creation of security related policies and coding. To address all these concerns a comprehensive and common conceptual framework to support technical, business and social aspects is needed. The paper suggests a list of general and specific requirements for security risk modelling that potentially could result in a method to support design, realisation and management of system security.

3 Security and Compliance Text Fonts

Towards Compliance of Cross-Organizational Processes and their Changes by David Knuplesch, Manfred Reichert, Jurgen Mangler, Stephanie Rinderle-Ma, and Walid Fdhila give an overview of the requirements and challenges to be addressed in order to ensure compliance with regulations, standards and laws. The David Knuplesch argued that ensuring compliance for cross-organisational processes and their changes it is important to understand what modelling cross-organisational compliance rules are, how changes are propagated and what guidelines should be followed to ensure efficient cross-organisational instance migration. It is also important to ensure compliance regarding the data privacy. Other two challenges also include efficient compliance at change time and the adequate treatment of the user feedback.

Achim D. Brucker presented the paper *Secure and Compliant Implementation of Business Process-driven Systems* authored by Achim D. Brucker and Isabelle Hang. In process-aware information systems (PAIS), manually managing compliance with security and privacy policies generates costs. To address this problem, they present a method for statically checking the security and conformance of the system implementation, e.g., on the source code level, to requirements specified on the business process level. As the compliance is statically guaranteed already at design-time, their method reduces the number of runtime checks for ensuring the security and compliance and, thus, improves the runtime performances. As a result it also reduces the costs of system audits, as there is no need for validating compliance of the generated log files.

A Process Deviation Analysis Framework by Benoit Depaire, Jo Swinnen, Mieke Jans, and Koen Vanhoof address the problem of flexible business process models. Process deviation analysis is becoming increasingly important for companies, as a means to provide agility and to adapt to changes. However, these changes are usually uncontrolled and, in fact, there is not much work dedicated to their understandability. This short paper presents a framework which structures the field of process deviations and identifies new research opportunities. The actual application

of the framework starts from managerial questions which relate to specific deviation categories and methodological steps. The paper sketches a high-level method to detect high-level process deviations, which is being developed in detail as a further work.

4 Security and Internet Services

The session opened with the second keynote speech. Sven Heiberg's presentation of *New Technologies for Democratic Elections* overviewed the technological advances and hurdles towards the provision of e-voting facilities in Estonia. The keynote described the challenges, both technical and ethical, involved in the implementation process of the Internet voting (i-voting). The presenter reported on the recent "i-voting" experience at the Estonian Parliamentary elections in 2011. The paper shows and classifies the security attacks (i.e., manipulation, revocation, and reputation attacks) and argues for the individual verifiability as the control to voters and election officials to countermeasure the determined attacks.

In the presentation of the short paper *Securely Storing and Executing Business Processes in the Cloud* by David Martinho and Diogo R. Ferreira, David Martinho addresses the problem of employing cloud computing to store classified data. Since the service provider can access all data, he, accidentally or deliberately, could leak it or use it for unauthorized purposes. As a countermeasure, the authors propose an architectural solution to securely operate their business processes relying on cloud-based services. This solution is built upon a thick client and thin server architectural pattern, where security constructs such as public-key and symmetric cryptographic systems are used to maintain confidentiality between the participants while keeping the server unaware of their participations and business process instances.

Focusing on the secure enactment of business processes, the short paper *Advanced Protection of Workflow Sessions with SEWebSessions* by Maxime Fonda, Stephane Moinard, and Christian Toinard reported on an approach based upon mandatory access control to authorize various confidentiality and integrity properties for the session state. The suggested SEWebSessions approach prevents the security violations associated with malicious accesses. Besides the technical means, the paper also describes experiments with SEWebsessions illustrates its portability to various platforms.

5 Engineering Secure Business Processes

In presenting the first paper in this session, *Modeling Wizard for Confidential Business Processes* by Andreas Lehmann and Niels Lohmann, Andreas Lehmann looked into the problem of avoiding information leaks within and across business process models. He presents a modelling prototype that integrates the non-interference check into the early design phase of an inter-organizational business processes. It not only helps receive the instant feedback on confidentiality assignments, but also automates completion of partial assignments toward guaranteed non-interference. This is an important tool for constructing secure business processes "by design".

Naved Ahmed in this presentation of *Towards Security Risk-oriented Misuse Cases* by Inam Soomro and Naved Ahmed argued for the necessity to understand business security through the security risk management. The presenter illustrated how misuse case diagrams could be extended to support security risk management by introducing construct for security criterion, vulnerability, risk impact, and security requirements. The approach could be generalised to other modelling languages, too. Potentially such language extensions could lead to the alignment of the business and functional perspectives. It could, potentially, result in a comprehensive and systematic development approach of the secure systems.

A Case Study on the Suitability of Process Mining to Produce Current-State RBAC Models by Maria Leitner, Anne Baumgrass, Sigrid Schefer-Wenzl,

Stefanie Rinderle-Ma, and Mark Strembeck was the last but certainly not the least, paper of the workshop. In her short paper presentation Maria Leitner overviewed techniques to derive role-based access control models (RBAC) from the execution logs of business process models. This paper closes an existing gap between the organizational control mining (a segment of process mining, in particular process discovery) and the security reasoning. This contribution is particularly interesting for industry, where the definition of role structures evolve over time and inconsistencies between these roles definitions could appear.

6 Workshop Design

All the workshop presentations were carried on in the mutual and interaction between the paper presenters and the audience. Before the workshop each long paper was assigned two discussants and short paper - one discussant. The responsibility of the discussant included reading the paper before the workshop and preparing few questions, which initiate and challenge the discussion after the paper is presented. This worked out at the largest extent and stimulated very interesting discussions involving not only the presenters and discussants, but also the overall audience. The organizers were explicitly requested to follow such a workshop format in the future editions.

7 Observed Trends

The provision of security and privacy guarantees is a rapidly growing research area in business process management. One indicator for this is the growing number of submissions (compared to the previous ABPSM'11 and WfSAC'11 workshops) we experienced. Another indicator is the elevated quality of submissions we received for SBP and the topics they approach. Below we report on some trends we observed in SBP considering the overall set of submissions (not only those accepted).

Among the submissions, eight papers address the problem of providing security "by design" - i.e. preventively ensuring that process models and execution environments comply with the requirements - and four papers focusing on the detection of security relevant incidents "after the fact". Less attention has been given to runtime approaches (two papers). This is insofar interesting, as there are powerful mechanisms to enforce processes and even correct them during the execution, thereby guaranteeing compliance. However, these approaches come at the cost of runtime overhead and we speculate this overhead may be prohibitive for practical approaches. Equally interesting is the fact that approaches focussing on the formalisation of requirements are underrepresented. We did not expect that; in fact, the call for papers explicitly mentions this as a desired contribution area, for a non-negligible "expressivity gap" between users and verification tools exist in this point. However, on the one hand there is a plethora of languages to express requirements, on the other hand each analysis mechanism encompasses its own language for the specification of requirements. Hence, works focussing solely on the expression and formalisation of requirements might have become uninteresting. Still, we believe that the whole area of requirements engineering should play a bigger role in the area of business modelling.

Another interesting trend we observed is the use of process mining techniques to address the verification of security properties. This is, in our view, a natural development in the area. Process mining focusses on process analysis in general and provides powerful analysis tool. The security community recognises this potential and shifts the application to security properties. We expect this trend to remain for the next years. However, advances in the area of data-aware process mining will be necessary, otherwise one is only able to reason about the structure of the process. Admittedly, this is not enough for security. Here this is a point where we see the security community advancing process mining.

As for the topics, the workshop sets out to address "security" aspects of business processes. However, one trend we observed regards the fact that the provision regulatory compliance is usually seen in this context. From the viewpoint of analysis, this is not surprising as one is merely changing the flavour of the requirements. Still, we would rather expect papers on privacy (no submissions on this topic) than on compliance. Hence, one interpretation of this trend is that the SBP workshop is understood as a venue to discuss reliability issues of business processes.

See you next time...

The next SBP proposal is planned for the BPM 2013 in Beijing, China (August, 26-30). We would be very happy to see you there.

Rafael Accorsi

Albert-Ludwigs-Universität Freiburg
Friedrichstr. 50
Freiburg i.Br.
Germany
accorsi@iig.uni-freiburg.de

Raimundas Matulevičius

University of Tartu J. Liivi 2
Tartu
Estonia
rma@ut.ee

Enterprise Modelling and Information Systems Architectures

The journal *Enterprise Modelling and Information Systems Architectures* is the official journal of the Special Interest Group on Modelling Business Information Systems within the German Informatics Society (GI-SIG MoBIS).

The journal Enterprise Modelling and Information Systems Architectures is intended to provide a forum for those who prefer a design-oriented approach. As the official journal of the German Informatics Society (GI-SIG-MoBIS), it is dedicated to promote the study and application of languages and methods for enterprise modelling – bridging the gap between theoretical foundations and real world requirements. The journal is not only aimed at researchers and students in Information Systems and Computer Science, but also at information systems professionals in industry, commerce and public administration who are interested in innovative and inspiring concepts.

The journal's editorial board consists of scholars and practitioners who are renowned experts on various aspects of developing, analysing and deploying enterprise models. Besides Information Systems, they cover various fields of Computer Science.

Subscription Information

The journal is distributed free of charge for members of the GI-SIG-MoBIS. Membership can be acquired through the German Informatics Society (http://www.gi-ev.de/verein/mitgliedschaft/). Single issues, priced at EUR 25 each (plus shipment), can be ordered online (http://www.fg-mobis.gi-ev.de/).

ISSN 1866-3621 (Online)

Guidelines for Authors

The journal serves to publish results of innovative research on all facets of creating and analysing enterprise models and information systems architectures. For research papers, it is required to satisfy academic standards in terms of originality, level of abstraction and justification of results. Experience reports serve to describe and analyse success stories as well as practical obstacles and resulting research challenges. Topics covered by the journal include, but are not restricted to the following subjects:

- Languages and Methods for Enterprise Modelling
- Reusable Domain Models (Reference Models)
- Analysis and Design Patterns
- Modelling of Business Processes and Workflows
- Process-Oriented System Architectures
- Component-Oriented System Architectures
- Conceptual Modelling for Component-Oriented Design
- Ontologies for Enterprise Modelling
- Modelling for Enterprise Application Integration
- Modelling for Data Warehouses
- Modelling to support Knowledge Management
- Model-Driven Development
- Aspect-Oriented Design
- Agile Methods for Enterprise Modelling

Authors are asked for electronic submissions, which have to be sent to the editor in chief as e-mail attachment. In case of multiple authors, it is required to name one author who acts as contact person. The submission should include a cover page with the paper's title and the names, affiliations and e-mail addresses of all authors. The first page of the paper starts with the title and does not carry the authors' names. A manuscript must be either in MS Word or PDF format. It should not exceed 5.000 words – this includes an abstract of around 150 words.

Submitted papers will be reviewed within no more than two months. The review process is double blind. Authors who submit a manuscript guarantee that it has not been published elsewhere, nor is intended to be published elsewhere. Papers that were accepted for publication must be written according to the style defined for the journal. A comprehensive description as well as a corresponding Word template is provided on the web portal of the GI-SIG-MobIS (http://www.fg-mobis.gi-ev.de/).

SIG MoBIS Portal

The GI SIG-MoBIS portal provides numerous resources on enterprise modelling research, such as a full-text digital library, a bibliography, conference announcements, a glossary and evaluation reports. It is intended to establish the premier forum for an international community in enterprise modelling. The new version is based on a Content Management System allowing authorized users to conveniently upload content. A BibTex interface allows for conveniently integrating bibliographic data. Information about this journal, such as guidelines for authors, tables of content and full-text access to articles (for SIG-MobIS members only) are also available on the portal.

SIG-MoBIS encourages everybody who wants to participate in the evolution of this community knowledge base to contribute to any of the categories covered by the portal. Please contact Michael Heß (m.hess @uni-duisburg-essen.de) for further information.

http://wi-mobis.gi-ev.de/

German Informatics Society

SIG MoBIS
Special Interest Group
on Modelling Business Information Systems

ISSN 1866-3621 (Online)

German Informatics Society