A Modeling Language for Describing Enterprise Architecture Management Methods

Sabine Buckl, Florian Matthes, Christian M. Schweda

Lehrstuhl für Informatik 19 (sebis)
Technische Universität München
Boltzmannstr. 3
85748 Garching
{sabine.buckl,matthes,schweda}@in.tum.de

Abstract. Methods for managing the enterprise architecture (EA) are presented in different approaches from researchers, practitioners, and standardization bodies. These methods are thereby often described textually – a fact, which we exemplify with the pattern-based approach to EA management. A modeling language for EA management methods can improve the state-of-the-art in documenting the methods and make them more comparable. Aiming towards the establishment of such a language, we discuss the requirements for describing EA management methods. Along these requirements, we show that well-established languages from bordering fields are only partially suited and discuss how the *Business Process Model Notation* can be extended to a modeling language for EA management methods.

Keywords: EA management, Methods, Modeling language, Business Process Model Notation

Classification: procedure models and modeling languages for the development of reference models

1 Introduction

Enterprise architecture (EA) management is a challenging management endeavor targeting the complex and interwoven system of the EA as a whole. As with many management processes, EA management is likely to be enterprise-specific in the steps taken as well as the artifacts created and exchanged. This is especially true, as EA management is intended to address a multitude of different concerns, which distinct enterprises do not necessarily regard to be equally important (cf. Kurpjuweit and Winter in [KW07] or sebis in [Cha10]). This fact is accounted for by the pattern-based approach to EA management [BEL$^+$07, Ern08]. The approach introduces different types of so called EA management patterns, which are developed to address typical management concerns in the context of EA management. In accordance to Buckl et al. in [BEL$^+$07] these types are:

Methodology patterns (M-Pattern) which define steps to be taken in order to address given concerns. Further, the methodology patterns provide guidance for applying the method, statements about the intended usage context, and consequences arising from utilization.

Viewpoint patterns (V-Pattern) which provide visual languages used by the methodologies. A viewpoint pattern proposes a way to present data stored according to one or more information model patterns.

Information model patterns (I-Pattern) which supply underlying models for the data visualized in one or more viewpoints. An information model pattern conveys an information model fragment including the definitions and descriptions of the used information objects.

For the later two types of patterns formal description languages or techniques are widely adopted, e.g. the Meta Object Facility [OMG06] for the information model patterns, or have been proposed, e.g. the visualization model [ELSW06] and [Ram09] of software cartography for the viewpoint patterns. In contrast, the methods are typically described rather informally, mostly by giving textual descriptions of the activities to be taken. In some cases, these descriptions have been complemented with informal ad hoc visualizations indicating the sequence of method steps. Such visualizations are notwithstanding helpful for getting an overview about the method, but are not detailed enough to provide actual guidance for applying the method. This is especially true, if the semantics of these ad hoc visualizations is considered, as e.g. aspects of deciding between different steps are not alluded to. Additionally, these visualizations do neither account for the information exchanged between activities nor for actors associated with the execution of a method constituent, e.g. being responsible, accountable etc. for executing the method step.

We regard such detailed information to be beneficial, e.g. for providing in-depth guidance, but also for integrating different methods – methodology patterns or method fragments from different approaches – during the creation of an enterprise-specific EA management process. This integration is necessary to leverage the proven-practice knowledge, as represented by EA management patterns, and apply it to an enterprise in a tailored way. In response to the aforementioned need, we develop a language for documenting EA management methods starting with an elicitation of requirements, which such a language must fulfill, in Section 2. Subsequent Section 3 discusses the fulfillment of the requirements by selected languages for process descriptions, which we regard to be candidates applicable to describing EA management methods. Having discussed the strengths and weaknesses of existing languages, we propose a language similar to the Business Process Model Notation (BPMN) for describing EA management methods in Section 4. Section 5 shows how this BPMN-like language fulfills the aforementioned requirements by giving an application example, i.e. a selected EA management methodology pattern from the EA management pattern catalog [Cha10] is reformulated in the concepts of the proposed BPMN-like notation. Final Section 6 concludes this report and provides a critical reflection on the proposed language.

2 Requirements

Based on an extensive survey on the utilization of viewpoints for EA management [BELM08] and the evaluation of the pattern-based approach to EA management in student projects [Böh08, BMSS09, Die08, Pfl08], we were able to elicit a set of requirements for a modeling language for EA management methods. These application-purpose-specific requirements were further complemented with more general requirements for a modeling language elicited by Frank in [Fra09] and specific considerations on methods and method fragments of Kühn in [Küh04]. Subsequently, we provide the resulting requirements:

R1. The modeling language must support to model *named activities* and the *information flows* connecting the activities. Thereby, the language must provide means to express the direction of *information exchange*, *decisions* in the information flow and optional or alternative activities, i.e. it must be possible to model that different activities are executed if different circumstances apply. These decisions must be complemented with rules for deciding which activities should be executed.

R2. The modeling language must support to model *named actors*, which are associated to corresponding activities. By this association, the languages expresses that the actor participates and/or drives the execution of the step.

R3. The modeling language must support to model *named artifacts*, which are used by the actors that participate in an activity. The method modeling language must support to specify the conceptual *modeling language* used for an artifact.

R4. The conceptual modeling languages should also be usable to specify the *input* and *output types* of an activity, i.e. to make explicit the step's *interface*. The method modeling language should further supply techniques to detail the conceptual modeling language's *notation*, *syntax*, and *semantics*[1] as well as their corresponding utilizations for certain actors, roles, and artifacts.

R5. The modeling language must support to model *hierarchic activities*, i.e. to model that an activity on a higher level of abstraction is detailed with a set of activities on a lower level of abstraction.

R6. The modeling language must support to model *named exceptions*, i.e. it must allow making explicit, that one or more activities may fail. Based on these exceptions, the language must support to model information flows that are *triggered* by the corresponding exception.

R7. The modeling language should supply mechanisms to state the *concern*[2], i.e. the *problem*, which is addressed by an activity (on a distinct level of abstraction). Complementing the description of the concern, the language should also support to model the *consequences* of applying the activity.

[1]The terms *notation*, *syntax*, and *semantics* are utilized as defined in [Küh04].

[2]The term *concern* is used in accordance to its definition in the standard ISO 42010 [Int07].

R8. The modeling language should supply mechanisms to optionally describe the *execution context* of an activity, i.e. to make explicit environmental factors (*forces*) that influence the execution of the step in some way.

R9. The concepts as introduced in the modeling language should correspond to concepts, which modeling experts from the modeling domain are familiar with. Similarly, the graphical notation of the language should correspond to prevalent graphical notations in this modeling domain (cf. requirements *U1* and *U3* of [Fra09]).

R10. The modeling language should facilitate the development of tools for the creation of and for providing execution guidance for the described methods. (cf. requirement *A4* of [Fra09]).

R11. The modeling language must offer all the concepts needed to describe methods in the context of EA management, but must support restriction to exactly these concepts to prevent the introduction of accidental complexity (cf. requirements *A1* and *A2* of [Fra09]).

Within the above list, the key words ”MUST”, ”SHOULD”, and ”OPTIONAL” are to be interpreted as described in RFC 2119 [Bra97]. This interpretation yields a distinction between *mandatory* and *optional* requirements, which can be exemplified with requirements R7 - R10. These have strictly optional character, while most of the other requirements have both optional and mandatory constituents.

3 Evaluating selected process modeling languages

The requirements from preceding Section 2 give rise to the assumption that process modeling languages might be appropriate for modeling EA management methods. Therefore, we would identify the activities of a method with the processes or process steps as contained in a process modeling language. This nevertheless points to a subtle complexity associated with this procedure, as process modeling languages are designed to describe the sequence of process steps on a rather concrete level, while method descriptions only describe information flows but do not a priori assume an order of execution. Nevertheless, we review some prominent examples for process modeling languages and discuss how far they satisfy the requirements.

Extended event driven process chains (eEPC) [KNS92] are widely used for modeling business processes. Due to their focus, they are quite well suited for describing a set of named process steps together with the corresponding actors and the information objects, which are created and exchanged during the process. Nevertheless, in respect to modeling information flows a complexity exists; two process steps might be connected via an information object, but have to additionally be connected with at least one triggered/triggering event. Furthermore, the eEPCs do not provide means to specify all language aspects (notation and semantics) of the artifacts. Process hierarchies are supported by eEPCs, named exceptions are not, but could be realized using specialized events. The same is true for a dedicated execution context concept, which can be realized by a textual comment.

The business process modeling notation (BPMN) [Gro09] is an open standard for modeling business processes for some years now and has become a widely accepted and used technique. The language provides means for modeling named process steps, named actors, named artifacts, and process hierarchies. The language aspects of the artifacts can be annotated textually – notation, syntax, and semantics are not directly considered. Notes can be used to supply the execution context for a process step; named exceptions are not directly present in the modeling language but can be modeled utilizing *errors* or *signals*. A process modeled in the BPMN can be made executable, e.g. as a mapping to BPEL (see below) exists. Furthermore, open source tools for web-based BPMN modeling exists (cf. Decker et al. in [DOW08]).

The business process execution language (BPEL) [Com09] has been developed as a language for describing executable business process based on web-services. We subsequently discuss the language concepts of BPEL, as they might be semantically interesting for our considerations; it has to be noted, that BPEL does not comprise a standardized graphical notation. BPEL supports to model named process steps, named and typed artifacts, as well as named actors, although these are initially designed to be realized as web-service agents. While XML-Schema is used as the rich type-system for the artifacts, neither semantics nor notation for the artifacts is especially alluded to. Process hierarchies are not directly supported by BPEL, although certain extensions as BPEL-SPE exist. With the focus on realizing executable business processes, the concept of the named exception is prominently present in the specification. Further, comments provide means to supply the execution context of a process step. BPEL is backed by strong industry support, although the user community is mostly centered on developers and people from technical provenience.

Also of a rather technical origin is the UML notation of activity diagrams [(OM04]. Their language allows specifying process steps and named information objects, while named actors are not supported. Discussions on the language, which the artifacts are modeled in, are only partially undertaken, although a binding to other UML concepts exists. This could be employed to model the artifacts' syntax. Named exceptions can be specified, the execution context can be described using comments.

Petri-nets (cf. e.g. [Bau96]) have a long history in describing processes, especially if questions of concurrency are concerned. Their mathematical background supports a formally defined execution semantics, which Petri-nets are well accounted for in the mostly technically oriented environments, they are used in. Nevertheless, without extensions, these nets do not well satisfy the requirements. In particular, neither named process steps, nor named actors are supported. Via the extensions of colored or Object Petri-nets [MA00] a typing of artifacts could be achieved. Finally, Petri-nets have become less frequently used, albeit their undisputed facilities for simulation.

The integrated definition for process description capture method (IDEF3) [MMP^{+}95] is a scenario-driven language for modeling processes with a long usage and development history. The language is somewhat inspired by the Petri-nets technique, such that named states are used instead of process steps. Named actors are not part of the specification; named artifacts can be realized. A language specification for the artifacts is not alluded to; named exceptions are also not supported. Notes allow supplying information on the execution context of a process step textually.

	Activity diagrams	BPEL	BPMN	eEPC	IDEF3	Petri nets	YAWL
R1	+	+	+	+	o	o	o
R2	-	++	++	++	-	-	-
R3	–	–	–	o	–	–	–
R3	+	+	++	++	o	-	-
R5	+	+	++	++	-	-	+
R6	++	++	+	-	-	-	-
R7[1]	o	o	o	o	o	–	–
R8	o	o	o	o	o	–	–
R9	+	+	+	++	-	o	–
R10	++	++	+	+	-	+	+
R11	+	++	+	++	-	-	o

[1] Concern and execution context can mostly be supplied only by textual annotations.

Table 1: Fulfillment of requirements of the selected process modeling languages

Table 1 summarizes the extent to which the process modeling languages fulfill the requirements from Section 2. The language YAWL [BD07], which was assessed by us, is not discussed in detail in the article as it is not widely used. The symbols used throughout the table range from ++ indicating complete fulfillment over o indicating medium fulfillment to – indicating total lack of support. Based on the evaluation results, we present a language, which on the one hand reuses BPMN notations where possible but on the other hand extends the notation if necessary for modeling EA management methods in the subsequent section.

4 Developing a modeling language for EA management methods based on BPMN

In response to the requirements as discussed in Section 3, we introduce the modeling concepts, which we regard necessary to support modeling for EA management methods. These concepts are meant to provide the basis for the language's meta-model. If a corresponding concept is available in the BPMN specification 2.0 a mapping is given in brackets.

Activity (activity) An EA management method consists of different steps, so called activities. The step exposes an interface to its environment, can be decomposed to child activities and may employ notations to present the utilized artifacts to the participating or driving actors.

Activity decomposition (atomic and compound activities) An activity can participate in an activity hierarchy, e.g. can specify, that it is part of a more detailed description of an activity on a higher level in the hierarchy.

Activity interface An activity has input and output information objects (including its *syntax* and *semantics definition*). The input and output objects form the interface of the corresponding activity, specifying, which *type of input* is demanded and which *type of output* can be expected. The type of input or output can be described by referencing an *I-Pattern*. Whereas, no direct concept representing an activity interface is given in BPMN, the concept of a *data objects* is introduced, which defines the information an activity requires to be performed and/or what it produces.

Actor Activities are executed by acting systems or persons, of which the later – in a well defined method – act in a distinct role, e.g. CIO, project manager, etc. The *pool* element of the BPMN can be used to represent the actor.

Artifact An artifact is a representation of information, e.g. a visualization or a questionnaire. The artifacts are presented to the activity's associated actors in an appropriate fashion and are considered by them during the execution of the activity. The artifacts are further modeled in accordance to a selected modeling language.

Exception The execution of a process step can lead to exceptional situations, in which the execution cannot be completed. In such a situation, an exception is raised. Although exceptions are not directly contained in BPMN but can be modeled using special types of triggers, as e.g. *signal* or *error*.

Information flow The different activities constituting a method are connected via information flows. An information flow specifies that information objects are handed over from a source to a target. The *sequence flows* of BPMN can be re-defined to model information flows. Two specializations of the information flow concept exist:

[Deciding information flow (gateway)] The information flow is not necessarily linear, but may contain points, where decisions take place. The subsequent activities, which receive information, are decided based on optional *guards* associated with the different options. If the guard is omitted, it is assumed to be *true*.

[Merging information flow (gateway)] Information flows originating from different activities are integrating via a merge.

Notation definition In accordance to the understanding of modeling languages as presented in [Küh04], a modeled artifact uses a certain notation to display its contents in the context of an activity to a dedicated actor.

Semantics definition An artifact is modeled using a distinct modeling language. The semantics of this language's concepts is provided in the semantics definition of the modeling language.

Syntax definition A modeling language for artifacts provides a defined set of concepts together with rules, how these concepts can be related. The syntax definition presents this set and the corresponding rules for the modeling language as associated with an artifact. The BPMN specifies a syntax, which must at certain points be extended in order to address the aforementioned requirements.

In the above list of concepts, the constituents of a conceptual modeling language – notation, syntax, and semantics – have been introduced separately, as they might supply different relationships to actors, artifacts, and activities, respectively. This becomes especially obvious, when the explanation of the notation definition is revisited. A notation definition forms a ternary relation concept, which determines,

- how a *distinct information* is presented as an artifact
- in the context of *one activity*
- to a *specific actor*.

Together, notation, syntax, and semantics form the different EA modeling languages used throughout the EA management process. These languages are built on the idea of patterns (cf. Section 1), especially the V- and I-patterns, of which the former make up the notation descriptions. The latter comprise the syntax description and provide a textual semantics described in the glossary of terms referenced in the I-patterns. Complementing the theoretic discussion of an BPMN-like language, we present an example illustrating the applicability of the language in the subsequent section.

5 Exemplary application of the method modeling language

In order to show the suitability of the proposed BPMN-like language for modeling EA management methods, an application example is given in the following, which explicates an existing method for addressing standardizations concerns, which was initially proposed by Buckl et al. in [BEL$^+$07] and further developed in [Cha10]. The method, which serves as an application example addresses the following concern:

How can standards for the application landscape be defined and enforced?

In order to address the above stated concern, different method steps as illustrated in Figure 1, which utilizes the presented method language, need to be executed. The different method steps as well as the exchanged information flows, the participating actors, and the used artifacts are detailed on in the following.

The method starts by gathering information about the current state of the architecture. This information can e.g. be extracted from specialized EA management tools, existing application documentations, or elicited via interviews. In our example the information is gathered via revisiting existing documentation and filling a questionnaire. Thereby, the enterprise architect has to decide, which information should be gathered, e.g. information about applications and the used technology in our example where homogenization from a technology perspective is regarded. In order to validate the gathered information, the filled questionnaire needs to be reviewed by the application owner.

Based on the information gathered about the current landscape the technology homogeneity is analyzed. Thereby the enterprise architect is the responsible role and uses the viewpoint *Standard Conformity (V-5).* The analysis results are used in the following phase by the standard manager to create, update, or delete standards. Thereby, i.e. bar charts like e.g. *Effects of Project Proposals on Technology (V-38)* are used. The subsequent process step, applying standards, is concerned with defining which business application should conform to one of the aforementioned standards. Thereby, the enterprise architect utilizes viewpoints like *Clustering by Standards (V-6)*, which detail on the environment of the used technology, e.g. the using business application and their usage context. The enforcement of standards, the next process step, can be performed either via vertical or horizontal escalation (see [BEL$^+$09]). A typical decision criterion for horizontal vs. vertical escalation is the estimated budget of the project. In our example the vertical integration is chosen, if the project budget is more than 100000 €, then the decision about the future of the application is handed over to a EA board, which uses reports like *Technology by Architectural Standard (V-23)* or *Standard Conformity Exception (V-67)*. In the case where horizontal escalation is used, the enterprise architect is allowed to impose obligations e.g. budgeting (see [BEL$^+$09]). These obligations are documented utilizing reports or word documents. In case of a horizontal escalation defined errors may occur, e.g. if the imposed obligations are not followed.

6 Critical reflection and outlook

In this paper, we discussed the importance of a language for modeling EA management methods and explained how such language can be used to complement the pattern-based approach to EA management presented in [Ern08]. Based on the analysis of the method patterns, we elicited requirements, such modeling language has to fulfill and evaluated to which extent current process modeling approaches can be re-used in this context. Having discussed the advantages and shortcomings of the current process modeling languages, we found none of these languages fully satisfactory in respect to the requirements and proposed notation, syntax, and semantics for a dedicated EA management method language. Therefore, we propose an extension to the BPMN language. This language extension was exemplarily applied on a selected method pattern from the EA management pattern catalog [Cha10]. Although the exemplary application succeeded, a validation of the modeling language in practice is yet still to be undertaken. Further, the documented method pattern comprehensively covers different typical EA management activities for a given concern, ranging from documentation over analysis to planning and enacting. Therefore, the underlying proven-practice is not designed as a composed activity, whereas the decomposition performed during the creation of the method model lacks practical validation. Nevertheless, the existence of method pattern purely designed for one EA management activity, such as documentation [MJBS09], advocates for the applicability of the decomposition approach.

In describing the make-up of the artifacts used in and the kind of information exchanged between the single method steps, we prominently relied on the viewpoint and information model patterns presented in the EA management pattern approach. While therefore, both the information flows and the visualized information throughout the described method are clearly consistent, one cannot expect this to be the case, if the method models for the dif-

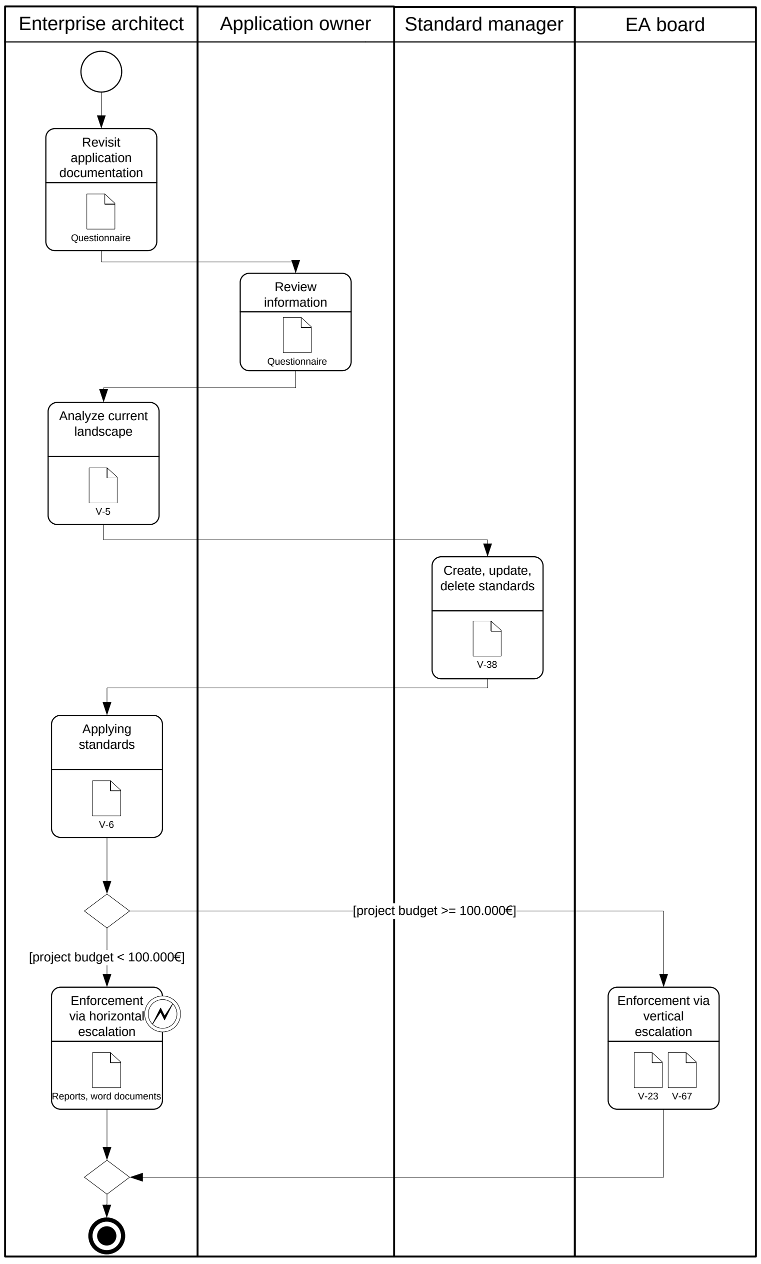
Enterprise architect
Application owner
Standard manager
EA board
Revisit application documentation
Questionnaire
Review information
Questionnaire
Analyze current landscape
V-5
Create, update, delete standards
V-38
Applying standards
V-6
[project budget >= 100.000€]
[project budget < 100.000€]
Enforcement via horizontal escalation
Reports, word documents
Enforcement via vertical escalation
V-23
V-67

Figure 1: Exemplary method model

ferent activities are developed independently. This raises an interesting question for future research: how can the method modeling language be complemented with a tool-set for ensuring that a composed EA management method is consistent in respect to information exchange and utilization. Similarly, questions how the modeling language can support such consistency checks should be analyzed further. Here, the works of [Küh04] and [Der06] provide interesting insights, which might prove helpful in answering the questions.

Finally, the methods as described in terms of the modeling language, presented above, remain on an intermediate level of generality: they are concern-specific, i.e. somehow redundant in respect to general EA management activities as documentation. But, they are also not tailored to the specific needs of an applying enterprise, i.e. there remain alternative and optional information flows, which have to be decided upon, if the method is implemented to an EA management function. We nevertheless see two reasons, why the language supports further development to both an increased level of generality and to a more detailed process description. On the one hand, generalization of activity descriptions may be possible by converting syntax and semantics of the modeling language used by the corresponding method to free variables of the method model. Thereby, method templates are created, which can work on arbitrary languages and support arbitrary concerns. On the other hand, further the specialization of the activity descriptions to concrete and enterprise-specific process models can be supported based on the method modeling language, by introducing configuration and adaption mechanisms as e.g. proposed in [Del06].

References

[Bau96] Bernd Baumgarten. *Petri-Netze. Grundlagen und Anwendungen.* Spektrum Akademischer Verlag, 2nd edition, 1996.

[BD07] Lindsay Bradford and Marlon Dumas. Getting Started with YAWL, 2007.

[BEL$^+$07] Sabine Buckl, Alexander M. Ernst, Josef Lankes, Kathrin Schneider, and Christian M. Schweda. A pattern based Approach for constructing Enterprise Architecture Management Information Models. In *Wirtschaftsinformatik 2007*, pages 145–162, Karlsruhe, Germany, 2007. Universitätsverlag Karlsruhe.

[BEL$^+$09] Sabine Buckl, Alexander M. Ernst, Josef Lankes, Florian Matthes, and Christian M. Schweda. State of the Art in Enterprise Architecture Management 2009. Technical report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany, 2009.

[BELM08] Sabine Buckl, Alexander M. Ernst, Josef Lankes, and Florian Matthes. Enterprise Architecture Management Pattern Catalog (Version 1.0, February 2008). Technical report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany, 2008.

[BMSS09] Sabine Buckl, Florian Matthes, Christopher Schulz, and Christian M. Schweda. Teaching Enterprise Architecture Management – A Practical Experience. Technical report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany, 2009.

[Böh08] Martin Böhme. *Softwarekartographie – Analyse und graphische Visualisierung von Teilen der Anwendungslandschaft des Klinikums der Universität München.* Bachelor's thesis, Fakultät für Informatik, Technische Universität München, Munich, Germany, 2008.

[Bra97] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels, 1997.

[Cha10] Chair for Informatics 19 (sebis), Technische Universität München. EAM Pattern Catalog Wiki. http://eampc-wiki.systemcartography.info (cited 2010-02-25), 2010.

[Com09] OASIS WSBPEL Technical Comittee. WS Business Process Execution Language Version 2.0, 2009.

[Del06] Patrick Delfmann. *Adaptive Referenzmodellierung - Methodische Konzepte zur Konstruktion und Anwendung wiederverwendungsorientierter Informationsmodelle*. PhD thesis, Universität Münster, Faculty of Economics, 2006.

[Der06] Gernot Dern. *Management von IT-Architekturen (Edition CIO)*. Vieweg, Wiesbaden, Germany, 2006.

[Die08] Thomas Dierl. *Models, Methods, and Visualizations for Complicance Management*. Bachelor's thesis, Fakultät für Informatik, Technische Universität München, 2008.

[DOW08] Gero Decker, Hagen Overdick, and Mathias Weske. Oryx – An Open Modeling Platform for the BPM Community. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pages 382–385. Springer, 2008.

[ELSW06] Alexander M. Ernst, Josef Lankes, Christian M. Schweda, and André Wittenburg. Using Model Transformation for Generating Visualizations from Repository Contents – An Application to Software Cartography. Technical report, Technische Universität München, Chair for Informatics 19 (sebis), Munich, Germany, 2006.

[Ern08] Alexander Ernst. Enterprise Architecture Management Patterns. In *PLoP 08: Proceedings of the Pattern Languages of Programs Conference 2008*, Nashville, USA, 2008.

[Fra09] Ulrich Frank. The MEMO Meta Modelling Language (MML) and Language Architecture (ICB-Research Report). Technical report, Institut für Informatik und Wirtschaftsinformatik, Duisburg-Essen, Germany, 2009.

[Gro09] Object Management Group. Business Process Model and Notation (BPMN) – FTF Beta 1 for Version 2.0, 2009.

[Int07] International Organization for Standardization. ISO/IEC 42010:2007 Systems and software engineering – Recommended practice for architectural description of software-intensive systems, 2007.

[KNS92] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Heft 89, Universität des Saarlandes, January 1992, 1992.

[Küh04] Harald Kühn. *Methodenintegration im Business Engineering*. PhD thesis, Universität Wien, 2004.

[KW07] Stephan Kurpjuweit and Robert Winter. Viewpoint-based Meta Model Engineering. In Manfred Reichert, Stefan Strecker, and Klaus Turowski, editors, *Enterprise Modelling and Information Systems Architectures – Concepts and Applications , Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8-9, 2007*, LNI, pages 143–161, Bonn, Germany, 2007. Gesellschaft für Informatik.

[MA00] Zapf Michael and Heinzl Armin. Ansätze zur Integration von Petri-Netzen und objektorientierten Konzepte. *Wirtschaftsinformatik*, Vol. 42(1), 2000.

[MJBS09] Christoph Moser, Stefan Junginger, Matthias Brückmann, and Klaus-Manfred Schöne. Some Process Patterns for Enterprise Architecture Management. In *Software Engineering 2009 – Workshopband*, pages 19–30, Bonn, Germany, 2009. Lecture Notes in Informatics (LNI).

[MMP$^+$95] Richard Mayer, Christopher Menzel, Michael Painter, Paula deWitte, Thomas Blinn, and Benjamin Perakath. Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report, 1995.

[(OM04] Object Management Group (OMG). UML 2.2 Superstructure Specification (formal/2009-02-02), 2004.

[OMG06] OMG. Meta Object Facility (MOF) Core Specification, version 2.0 (formal/06-01-01), 2006.

[Pfl08] Katharina Pflügler. *Evaluation and Extension of the EAM Pattern Catalog in a German Insurance Company*. Bachelor's thesis, Fakultät für Informatik, Technische Universität München, 2008.

[Ram09] René Ramacher. *Entwurf und Realisierung einer Viewpoint Definition Language (VDL) für die Systemkartographie*. Diplomarbeit, Fakultät für Informatik, Technische Universität München, 2009.