TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Informatik XIX

Development of Organization-Specific Enterprise Architecture Modeling Languages Using Building Blocks

Christian M. Schweda

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. M. Bichler
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. F. Matthes
	2. Prof. Dr. M.J. van Sinderen, University of Twente, Enschede / Niederlande

Die Dissertation wurde am 19.04.2011 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.07.2011 angenommen.

Zusammenfassung

Unternehmen stellen heute mehr denn je komplexe und stark vermaschte Systeme dar, die sich in einem beständig verändernden Umfeld mit globalisierten Märkten, gesetzlichen Neuerungen und technologischen Entwicklungen behaupten müssen. Dabei rückt das ganzheitliche Management der Unternehmensarchitektur von den Geschäftsfähigkeiten und -prozessen bis zur IT-Infrastruktur immer mehr in den Mittelpunkt der Betrachtung. Die notwendigen Beschreibungen der Unternehmensarchitektur stellen in diesem Zusammenhang eine besondere Herausforderung dar, welche in der Literatur und Praxis uneinheitlich angegangen wird. Verschiedene de-facto Standards und Ansätze aus der Forschung bieten eine Vielzahl von Sprachen, welche in der Praxis jedoch selten unverändert zum Einsatz kommen. Daneben setzen nicht wenige Unternehmen auf selbstentwickelte Beschreibungstechniken. Diese Tendenz ist nicht zuletzt auf die hohe Unternehmensspezifizität der Sprachen sowie der von ihnen berücksichtigten Konzepte zurückzuführen. Der Gesamtumfang des Unternehmens als Betrachtungsgegenstand gebietet dabei im Speziellen gemäß dem Grundsatz ökonomischen Handelns eine Beschränkung auf wenige, notwendige Konzepte, welche speziell an die Problemstellungen des Unternehmens angepasst sind.

Diese Arbeit präsentiert eine Methode, mit Hilfe derer Unternehmensarchitekten spezifische und problemadäquate Sprachen für die Beschreibung von Unternehmensarchitekturen entwickeln können. Die Methode nutzt aus praxiserprobten Sprachen extrahierte Sprachbausteine, welche in einer Bibliothek gesammelt und gemäß eines konzeptionellen Rahmenwerks für die Unternehmensarchitektur organisiert werden. Dabei sind drei Arten von Bausteinen zu unterscheiden: Informationmodellbausteine (IBBs), welche die abstrakte Syntax, Viewpoint-bausteine (VBBs), welche die konkrete Syntax, sowie Glossarbausteine, welche textuelle Definitionen der relevanten Konzepte beschreiben. Durch die Vernetzung der Bausteine untereinander sowie durch die Verbindung mit typischen Problemstellungen des Unternehmensarchitekturmanagements können Benutzer der Methode die für ihr Unternehmen am besten geeigneten Sprachbausteine identifizieren. Mit Hilfe konkreter Techniken, welche auf zwei domänenspezifische Metasprachen aufbauen, unterstützt die Methode eine konsistente Integration der Sprachbausteine. Daneben werden Techniken vorgestellt, die es erlauben, die vereinheitlichende Terminologie der Methode an das Begriffsverständnis der Sprachgemeinschaft des Unternehmens anzupassen. Somit wird die nachhaltige Nutzbarkeit der Gesamtmethode zur Weiterentwicklung von Beschreibungssprachen im Unternehmen gefördert. Ergänzend werden Handlungsempfehlungen und Techniken dargestellt, mittels derer die Bibliothek von Sprachbausteinen fortgeschrieben und ergänzt werden kann.

Abstract

Today's enterprises are complex and highly interconnected systems that must transform in order to sustain in a continually changing environment of globalized markets, novel legal regulations, and emerging technologies. In response to these challenges, enterprises aim at a holistic management of their enterprise architecture (EA) which defines key elements as well as relationships between and within strategy, business, and IT. Such a holistic management builds on EA descriptions, which have to be based on well-defined textual and graphical modeling languages. Although many different de-facto standards and academic proposals for EA modeling languages exist, enterprises tend to develop their own customized languages based on these approaches to match their already established conceptual model of the EA or parts thereof. The EA management stakeholders build on information explicitly captured by process models, information systems models, or IT infrastructure models, as well as on information implicitly present in the vocabulary used within the enterprise. No established method for developing organization-specific EA modeling languages exists, although the development process is of high practical relevance.

In this thesis we present a method for developing organization-specific EA modeling languages to support EA management addressing the challenges of enterprise transformation. This method builds on a library of language building blocks extracted from practice-proven approaches and organized according to a conceptual framework of the EA. We distinguish three types of building blocks: *information model building blocks* (IBB) describing the abstract syntax, *viewpoint building blocks* (VBB) describing the applicable concrete syntax, and *glossary building blocks* (GBB) providing textual semantic definitions for the concepts. The method further builds on a set of formalisms to compose IBBs into a comprehensive information model, for applying VBBs to this information model, and for consistently evolving the GBBs to a terminology aligning to the terms used in the organization. We establish these formalisms based on the formal underpinnings of two domain-specific metalanguages. We further provide guidelines how to consistently evolve and adapt existing EA description languages developed by this method in response to changing environmental conditions. In addition, we supply an administration method for developing and evolving the organized library in a consistent and coherent manner.

Acknowledgment

There are many ways of helping some with a doctoral thesis: proofreading, discussing, and being there. Many people that I had the pleasure to meet during the time I was working on the thesis helped me in one or all of the above ways. The subsequent page is far to short to express my gratitude to all of you, but I will give it a try.

I would like to thank my primary supervisor, Prof. Dr. Florian Matthes. Without the freedom to pursue my own ideas and to make my own mistakes that you gave me as well as without your gentle reminders to stay to the 'pragmatic things', this thesis would not be the same. I would also like to thank Prof. Dr. Marten van Sinderen for being the second examiner of this thesis, as well as for our interesting discussions in Valencia.

My thanks go to my 'antecedants' in EA management research at the chair for Software Engineering for Business Information System, namely Dr. André Wittenburg, Dr. Josef Lankes, and Dr. Alexander Ernst. Your work was more than just a good starting point, but inspired me to continue. I am proud of having had the chance to go a part of the way together with you, as a student assistant, as a researcher assistant, and always as a respected peer. I would like to express my gratitude to the colleagues at the chair that accompanied my way in preparing and organizing lectures, conducting lab courses, and keeping the wheels of bureaucracy turning—thanks to Thomas Büchner, Ivan Monahov, Christian Neubert, Sascha Roth, Christopher Schulz, and Alexander Steinhoff.

My thanks go to the students, whose theses I supervised over the years. I hope, I was able to teach you something that proves useful. For myself, I learned with every thesis—thanks to Philip Achenbach, Florian Balke, Markus Bauer, Thomas Dierl, Johannes Ehm, Krzysztof Gröhl, Jakob Mund, René Ramacher, Martina Ruf, Martin Stange, Maren Steinkamp, Victoria Struck, Jan Wiegelmann, and Nicolai Zaidmann. I would also like to thank the students assistants that supported our EA management research over years: Philip Achenbach, Florian Balke, Ingrid Prem, René Ramacher, Michael Schätzlein, Alexander Schneider, and especially Thomas Dierl, who spent almost whole of his 'student life' as member of our research team.

I would like to thank my family, who encouraged me to 'do my thing'. You instilled me with a multi-faceted interest in science, which ultimately led to this thesis; you reminded me to take a rest sometimes and you believed in me all the time, even if I was in doubt myself. Thank you also for listening to my weird ideas.

My thanks go to (Dr.) Sabine Buckl who was a colleague working on BEAMS with me and is a friend for me. More than once, your optimistic nature lit my way, and your cunning opinion engaged us in discussions that proved useful. I owe you much, much more than I can say in a few lines. So I resort myself to saying "thank you".

Finally, I would like to thank all the people that I have lost and found over the years of my research. Thank you for being there, for teaching me patience, for tolerating my quirks, or perhaps for liking them. Thank you for the feelings that you shared with me. Last but not least thank you for accompanying me on a part of the way to the next transition in my life.

Garching b. München, 04-17-2011 Christian M. Schweda

Contents

List of Figures

List of Tables

List of Examples

I offer a toast. The undiscovered country...
the future

Chancellor Gorkon, Star Trek VI

CHAPTER 1

Introduction and Motivation

Enterprise Architecture (EA) and its management are topics of ongoing and increasing interest from practitioners. In recent years, a changing economic, regulatory, and technological environment [La05, Wa05, RWR06] motivated companies to introduce an EA management function. Thereby, the companies seek to realize the benefits that are widely alluded to as connected to EA management, namely "consistent strategic IT planning", "increased business/IT alignment", "business process optimization", and "architectural guidance for projects" (for a comprehensive list see Aier et al. [ARW08b]). Different consultancies list EA management as a top challenge for company executives in the (near) future [Pe09a, Pe09b, BA10c]. The high practical relevance of the field is also reflected in the large number of contributions from EA practitioners. The approaches of Dern [De06], Keller [Ke07], Niemann [Ni06], or Schekkerman [Sc06d, Sc08a] exemplify such contributions and provide 'pragmatic' guidelines for managing the EA.

Not only practitioners approach the topic of EA management, but also academic research and standardization bodies show increasing interest in the field. In [My11] Mykhashchuk et al. investigate the increase in publication activities in EA management, especially in comparison to the literature analyses of Langenberg and Wegmann [LW04a] as well as Schönherr [Sc08b]. Standardization bodies, as The Open Group[1], develop and evolve EA and EA management frameworks. A prominent example for such framework is *The Open Group Architecture Framework* (TOGAF), which has matured to a well-known and frequently quoted de-facto standard for EA management. It presents guidelines for managing the EA and is developed in cooperation between the 300 organizational members[2] of The Open Group.

[1] For a self description of The Open Group, see `http://www.opengroup.org/overview/what-we-do.htm`, last accessed 04-05-2011.

[2] Member list of The Open Group available at `http://www.opengroup.org/overview/members/membership_list.htm`, last accessed 04-05-2011.

The ongoing interest that EA management received from diverse directions has lead to a broad variety of proposals for guidelines, methods and techniques for EA management. These proposals differ with respect to the employed terminology but also with respect to the addressed aspects of managing EAs. This is not surprising as the proposals originate from largely different backgrounds. The literature analysis of Aier and Schelp in [AS09] draws a more consistent image of the discipline. The compared scientific EA management frameworks and approaches[3] differ with respect to their terminologies but agree on a central dichotomy in the field. All approaches supply one or more **EA modeling languages**[4] or *meta-models*, and a **method** or *procedure model*. Surprisingly, this fact is not reflected in the definitions for the term EA management. Moreover, as Schönherr describes in [Sc08b], there is not only a lack of a widely accepted definition for EA management, but also a majority of approaches abstains from giving a definition at all. For our work, we define EA management as a general management function targeting the EA.

Definition: EA management
The EA management function in an enterprise documents, analyzes, plans, and enacts the EA. The EA describes the current state of the enterprise (*descriptive aspect*) and makes prescriptions for its planned and target states (*normative aspect*).

In [BMS10j, Bu10a] we detail both the method constituent and the language constituent of an EA management function, thereby introducing the concepts of the **information model** and the **viewpoint**. These concepts are of central importance for this thesis. The viewpoints are used by the stakeholders of EA management to access and define architecture information, that is relevant to them. From this perspective, a single viewpoint shows characteristics of a modeling language. The information model interrelates the different viewpoints and establishes a common conceptualization for the EA. A multi-viewpoint approach to EA modeling ensures inner consistency via the common information model, which has to incorporate all concepts relevant to the different stakeholders of the EA management function, i.e., has to reflect their understanding of the EA. EA Management frameworks and approaches do not provide a definition of the term EA either, but define the EA by coarse-grained constituting **areas-of-interest**. Matthes et al. introduce in [Ma08, page 29] three "architectural layers" ranging from the **business & organization** over the **application & information** to **infrastructure & data**. The concepts on these layers constitute the core of the overall architecture, but are influenced by further concepts from cross-cutting domains as **visions & goals**, **strategies & projects** and **principles & standards**. Finally, the EA comprises **questions & *key performance indicators* (KPIs)** describing the EA in a quantitative way. Figure 1.1 shows the structure of the EA.

The concepts constituting the different layers and domains form the management body for EA management, in which additionally the relationships between the concepts are of high importance. This management body differs from organization to organization, depending on the EA management-related problems that the organization and its stakeholders seek to address. With this respect, an EA modeling language has to be problem-specific. Further, the language must align with the terminology for EA concepts already used in the organization,

[3] In the paper, the frameworks are imprecisely alluded to as *EA frameworks*, although the frameworks focus on activities related to or being part of managing the EA.
[4] For information on the conventions about highlighting terms, please refer to Section 1.4.

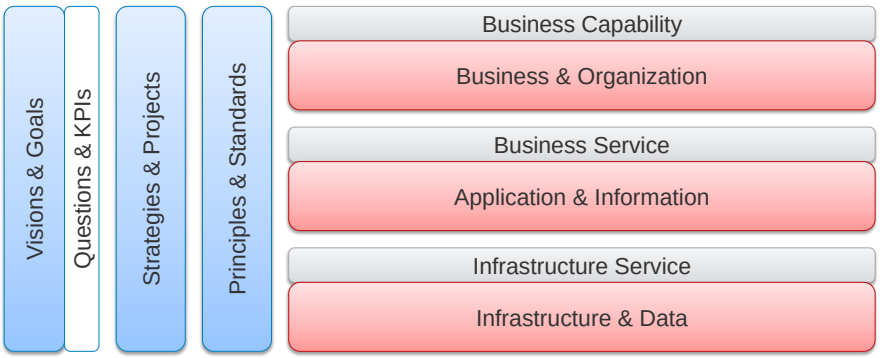


Figure 1.1: The overall structure of the EA

i.e., has to be terminologically compatible. In the following, we subsume problem-specific and terminologically compatible under the term **organization-specific**.

1.1 Motivation

The development of an organization-specific EA modeling language is a complex task. Buckl et al. in [Bu07b], Aier et al. in [Ai08a] as well as Ernst in [Er10] discuss three different approaches to this task, namely

the *greenfield approach*, where a company designs a management function 'from scratch', possibly relying on experiences from EA-related fields in the enterprise,

the *customization approach*, where an enterprise selects a prominent EA management approach from literature and adapts the approach to satisfy the organization-specific requirements, and

the *integration approach*, where an enterprise selects different EA management approaches from literature and integrates them into an organization-specific EA management function.

The *greenfield approach* provides the maximum degree of freedom for designing an EA management function that optimally addresses the specific EA management-related problems of the enterprise. Nevertheless, this flexibility does not come without costs. In [Bu07b, pages 148–152] we discuss several disadvantages of the greenfield approach:

"giant models" to which concepts are introduced without explicit account for their utility in the management function but on interest of the different stakeholders involved,

"stockpiling useless information" which is not needed to address the EA management-related problems of highest relevance, and

"missing methodologies [sic!]"[5] for gathering architectural information, communicating architecture descriptions, and creating architecture plans, which have to be developed in the using organization.

In this vein, a greenfield approach 'reinvents the wheel' over and over again.

[5] More precisely, we should have talked about missing **methods**.

The *customization approach* adapts an EA modeling language from literature to the specific needs of the using organization. Aier et al. give some examples on adaptation difficulties in [Ai08a, pages 559–560], of which the first two ones are interrelated and present themselves as variants of what is colloquially called the 'not-invented-here-syndrome':

"terminological disruption" by the terms supplying with the literature approach, which are different from the terms already used in the enterprise,

"missing stakeholder acceptance" as the EA modeling languages do not reflect the information demands of all relevant stakeholders in the enterprise, and

"unreflected model utilization" by which the enterprise is fit against the model instead of shaping the model to address the EA management-relevant problems.

Aier et al. further highlight in [Ai08a] that only a minority of the approaches encompasses methods or provides guidance for adapting the approach. We raise a similar argument in [Bu07b] and illustrate the difficulties of adapting EA modeling languages, especially when it comes to omitting concepts not needed by the using organization.

The *integration approach* supplies **building blocks** for an EA modeling language. The using organization selects the ones relevant for the EA management-related problems as well as the corresponding stakeholders and integrates these building blocks into a consistent set of EA modeling languages. An integration approach building on the EA modeling prescriptions made by today's EA modeling approaches falls for three difficulties:

terminological plurality as the different approaches supply different terms for the same concepts (synonyms) and supply the same term for different concepts (homonyms),

consistency of the information model as this model is composed from different building blocks that can make inconsumerable syntactic prescriptions, and

applicability of the viewpoints as these have to be consistent against the background of the configured and integrated information model. The viewpoints must further be designed that they together address all concepts in the information model.

Guidelines for integrating different approaches or parts thereof to an organization-specific EA modeling language are scarce in literature. An exception in this context is our work [Bu09b], where we describe how to complement TOGAF with EA management patterns originating from the sebis EAM *pattern catalog* [Ch10].

Both the *customization approach* and the *integration approach* help the using organization to design an EA modeling language based on practice-proven knowledge in the field. They are nevertheless not without drawbacks that complicate the development of an organization-specific EA modeling language that

- is suitable to address the organization-specific EA management-problems as reflected e.g. in distinct information needs,
- confines itself to the organization-specific EA management problems in order to avoid the collection of information not necessary, and
- uses the organization-specific terminology in describing the EA and in communicating relevant EA information.

We detail aforementioned preliminary requirements in Chapter 4 and define quality criteria for EA modeling languages. In the following, we outline the research objective of this work and delineate the different contributions. Together, these contributions enable the using organization to develop an organization-specific EA modeling language.

1.2 Research objective and contributions of the thesis

In the field of EA management different approaches bring along special-purpose viewpoints and modeling languages that are largely independent from each other. In addition, no guidelines exist on how to integrate views conforming to these viewpoints into a consistent description of the EA. Also, yet prescriptions on how to customize an integrated approach to avoid "terminological disruption" [Ai08a] are missing. This raises the central research objective of this thesis seeking to address this experienced gap.

> ***Research objective:*** Develop a method for designing and re-designing organization-specific EA modeling languages to support multi-viewpoint EA management based on redundancy-free best-practice knowledge documented in current EA management approaches.

The key contribution of this thesis are *building blocks for EA management solutions* (BEAMS), which offer a development method for organization-specific EA management functions based on best-practice building blocks. Different types of building blocks target the different aspects of an EA management function:

method building blocks provide the re-usable and integrable activity descriptions specifying EA management tasks together with responsible as well as informed participants as well as triggering events.

language building blocks describe re-usable and integrable fragments for the syntax of the language (**information model building blocks**), the notation (**viewpoint building blocks**), and for textual semantic specifications (**glossary building blocks**).

BEAMS is presented in two theses. In one we focus on the method part, in the other one (this thesis), we discuss the language aspect. Via a joint conceptualization of the domain, we ensure consistency between the two parts of BEAMS. Therefore, especially the formal understanding of the syntactical aspect of EA modeling languages as presented in Chapter 5 is of central importance. In detail, this thesis about the language part of BEAMS establishes five contributions:

meta-languages for describing EA modeling languages and corresponding building blocks,

an **organized library** of **language building blocks** that can be used to design an organization-specific EA modeling language,

an **administration method** for creating and maintaining the organized collection, e.g. for adding new building blocks,

a **development method** that supports the integration and customization of language building blocks during the design and re-design of an organization-specific EA modeling language, and

a **technical toolset** implementing the meta-languages and customization method for supporting the design of an organization-specific EA modeling language.

In the thesis we supply another contribution that does not directly relate to the research objective but to the corresponding research activity. We devise a *research design* that allows us to conduct practice-based research in the complicated environment, into which EA management embeds. Figure 1.2 shows how the contributions of BEAMS (except the toolset) integrate into the context of EA modeling and EA modeling (sub-)languages. The notion of the sub-language is employed to delineate that, from a theoretic point of view (cf. discussions in Section 4.3.1), each triple of viewpoint, underlying information model, and glossary can be considered a language on its own. Nevertheless, the different languages used by one organization have to integrate into a consistent whole. This whole shows typical characteristics of a language as it provides a glossary, an information model and at least one viewpoint. In this light, the whole itself constitutes an EA modeling language as well as its constituents do. We will show in Section 5.1 that the question of what to call a language or sub-language is only of minor interest, as long as mechanisms for ensuring consistency between different information models and glossaries are provided. We will see that as long as any two information models or glossaries are consistent, they can also be integrated into a single information model or glossary, respectively.

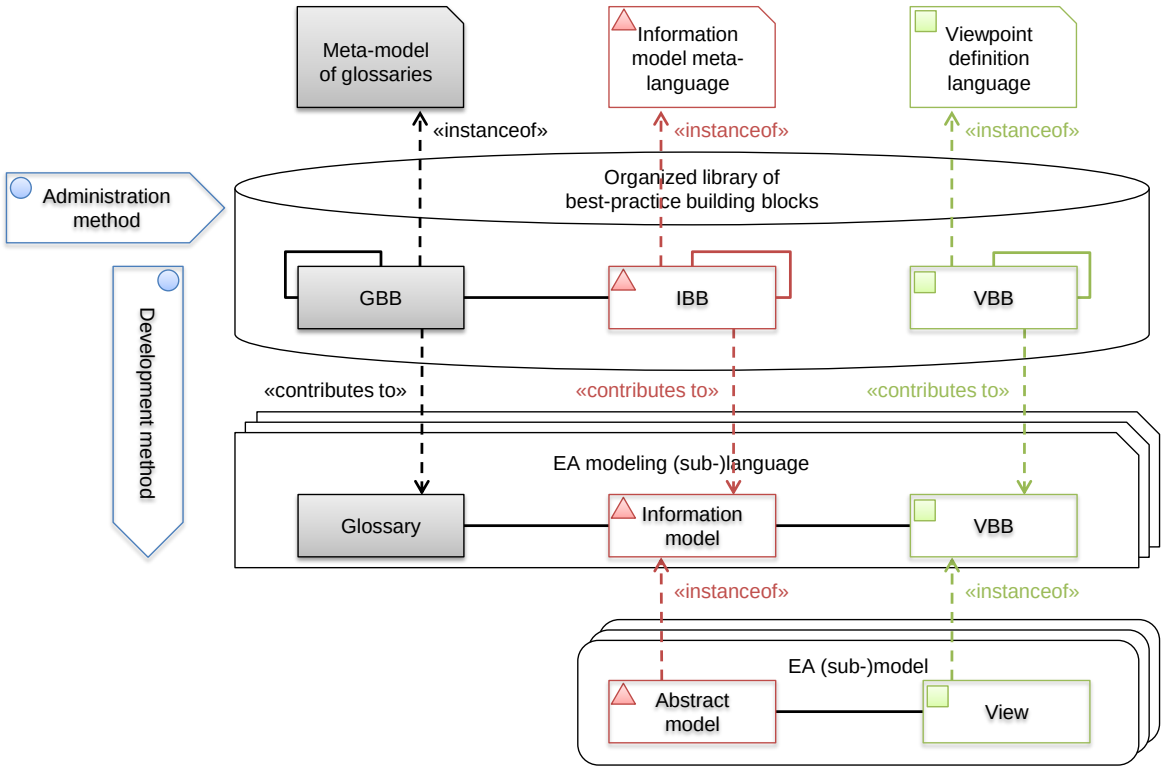


Figure 1.2: Overview about the contributions of the thesis

BEAMS combines a knowledge base on best-practice solutions with techniques and appropriate meta-modeling languages into a comprehensive development method for EA modeling languages. In developing BEAMS, we have to address two challenges. Firstly, we have to find a structure suitable for linking the language prescriptions made in the different approaches. Secondly, we have to accommodate for the fact that EA management is a still maturing discipline. Therefore, we have to provide means to re-organize BEAMS, when new design prescriptions and knowledge becomes available. This raises the first research question of the thesis.

> ***Research question 1:*** How and by which means can language prescriptions from different approaches be interlinked in an extensible artifact supporting the development of organization-specific EA modeling languages?

Prescriptions on how to develop an artifact, e.g. an EA modeling language, can be formulated on different levels of abstraction. Firstly, they can be presented as case-specific prescription in *case studies*, see e.g. [RWR06]. Secondly, they can be generalized from repeated observations to re-usable *patterns* [Bu07b]. Thirdly, they can be abstracted to *design theories* in the sense of Walls et al. [WWES92]. Such theories describe a general way to solve a class of design solutions in a given context. Prescriptions of all three types are subject to contextual factors that facilitate or impede the application of the designated solutions. Nevertheless, only patterns and design theories make these 'contingencies' of the solution explicit. Different EA management approaches present different prescriptions and delineate diverse contexts. These contexts not necessarily distinguish the solution in the sense of contingency factors, but can be independent from each other, i.e., form *asymmetric criteria*. Pries-Heje and Baskerville introduce in [PHB08] the concept of the design theory nexus. Such nexus links complementing and alternative design theories and provides support for selecting the prescriptions admissible in a given context.

In [BMS10k] we have discussed the difficulties of instantiating a theory nexus in an environment as complex and volatile as the field of EA management. In particular, we describe the difficulties in balancing rigorous research with the 'pace' of the practice partners, which are in notwithstanding needed to ensure applicability and relevance of the research outcomes. Responding to the difficulties, we revisit the work of Venable, who presented in [Ve06] a method for theorizing in design science research. We show how this method can be interwoven with the method on nexus instantiation, as proposed by Pries-Heje and Baskerville in [PHB08]. Further, we complement the method with our own findings on pattern-based theory construction [BMS10d], describing a research method for pattern-based instantiation of design theory nexus. In Chapter 2 we adapt this method to the specific context of designing EA modeling languages. Our instantiation of the design theory nexus incorporates best-practice patterns and prescriptions from existing EA management approaches. To prepare the instantiation, we revisit these approaches and explore their understanding of the subject EA. Thereby, we frame the field of EA management and answer the following research question.

> ***Research question 2:*** Which approach describes which activities constituting EA management, which problems do the approaches address, and which problem-specific EA modeling languages are supplied by the approaches?

In Chapter 3 we analyze today's prevalent EA management approaches in a multifaceted way. In particular, we elaborate, whether differences in their EA modeling languages reflect differing EA management-relevant problems or are 'accidentally' employed. From there, we prepare a conceptual framework for reasoning on EA modeling languages and building-blocks thereof. Against the background of this framework, a third research question can be derived as follows.

> ***Research question 3:*** Which requirements does the EA management domain exert on integration and customization artifacts for EA modeling languages?

Research question three depends on the understanding what EA modeling means as devised in response to research question 2. Based on this understanding, we establish a framework for developing multi-viewpoint modeling languages by integrating and customizing different languages. The **meta-languages** presented by the thesis lay the conceptual basis for integrating different building blocks for EA modeling languages into a single consistent language. In detail the thesis supplies a **meta-modeling language** for specifying the **syntax** of an EA modeling language or building blocks thereof. Via dedicated meta-concepts, as *dispersive types* (see Guizzardi [Gu05]), this language accommodates for the specificities of the modeling domain EA, which we will expatiate in Chapter 5. With respect to the **semantics** of the concepts in an EA modeling language or a corresponding building block, we supply building blocks for constructing an organization-specific glossary, i.e., for adapting the terminology to the using organization. These building blocks providethe basis for the language customization techniques. Mirroring the aspect of the **notation** or *concrete syntax*, we provide a **viewpoint definition language** for describing building blocks that allows to specify how the EA modeling language concepts are represented by graphical elements.

> ***Research question 4:*** How do meta-language based artifacts (constructs, models, and methods) and a toolset for integrating and customizing EA modeling languages look like?

The general method taken builds on an organized library of so-called **language building blocks** that are selected, adapted and integrated by users to develop problem-specific and organization-specific EA modeling languages.

The **organized library** of **language building blocks** brings together best-practice design prescriptions for EA modeling languages as found in current EA management approaches. Understanding the prescriptions as design theories, the organized library instantiates the design theory nexus establishing the means for selecting and integrating the theories best suited for a specific organization. The different theories are documented as **information model building block** describing syntactical aspects and **viewpoint building blocks** covering notational aspects. Further, all building blocks are grounded in corresponding **glossary building blocks** that are used to specify semantics by providing a common terminology. Decoupling a layered and structure-oriented perspective ranging from business to infrastructure (cf. Figure 1.1) from cross-cutting aspects applied on the different layers, the library of building blocks is organized to minimize redundancy while providing dedicated integration points. Particularizing these considerations, we discuss on the central notion of PROJECTS as vehicles of organizational change and discuss alternatives on how to incorporate projects into the EA modeling language.

The **administration method** describes steps to be taken and techniques to be used in creating and maintaining the organized library. With the library being an instantiation of a design theory nexus, the creation-specific steps and techniques build on the prescriptions given by Pries-Heje and Baskerville in [PHB08], where they describe how to instantiate a theory nexus. The prescriptions are furthered with respect to specific properties of the EA management domain, e.g. the plurality of terminologies, to foster applicability. Concerning the maintenance of the organized library, special attention is paid to the question how the stratified glossary can be employed to keep maintenance efforts low. In particular, the method discusses how explicit relationships between the building blocks, i.e., 'hard-coded' links, can be avoided. Thereto, we call on the notion of the **implicit relationship**, i.e., on relationships that can be derived between two or more building blocks linking to the same glossary element.

The method **development method** establishes a framework that is used to consistently integrate different building blocks for EA modeling languages. In particular, the framework supplies different customization technique targeting both the level of the information model building blocks, which are combined to an integrated EA information model, as well as the level of the viewpoint building blocks. Latter ones have to be consistently integrated, but also have to be linked to the integrated information model for establishing an executable mapping to the modeling language's notations. Additionally, the aspect of customizing the underlying glossary based on glossary building blocks are given. The method further states which requirements regarding the organization-specific EA management have to be gathered and makes prescription on how to organize these requirements to provide input for language customization. Accommodating for the fact that EA management is a long-running endeavor, whose context, area-of-interest, and goal may change over time, further a method for adapting an existing building block-based EA modeling language is provided. This method not only specifies indicators that can be used to detect a demand for adaptation, but also entails techniques that call on the implicit relationships between the building blocks to identify possible paths of evolution and maturation regarding the EA modeling language. Such mechanisms are used to implement *goal-setting* and *goal-measurement* in the knowledge-intensive activity EA management, in line with the perspective of Struck et al. [St10b] and with our perspective [BMS09b, BMS10e].

In research question 4 the more theoretical artifacts are complemented with a practical implementation thereof in a toolset, more precisely a configuration tool for developing EA modeling languages to support multi-viewpoint EA management. In particular, the toolset supports the users in selecting the problem-specific language building blocks, in adapting them to the organization-specific context, and in integrating them into a consistent set of EA modeling languages. In this respect, the tool guides the users through the steps of the development method and facilitates the use of supporting techniques, all based on the organized library of language building blocks. With this library itself being the basis of the method, a final research question targets the development and maintenance of the library. This is necessary as the field of EA management is still maturing, novel EA-related problems enter the center of attention, and previously relevant problems are reshaped in response to new environmental factors. The fifth research question specifically targets this challenge.

> ***Research question 5:*** What method can be used to facilitate the administration and extension of the organized library of language building blocks?

The answers to research questions 4 and 5 are closely interrelated to each other. In particular, the selection and integration techniques implemented in the toolset have to be maintained and evolved, whenever the library of building blocks is maintained. Having foreclosed the contribution of the work to some extent, the subsequent Section 1.2 gives an overview, on what is achieved by this thesis. Where the above contributions are of strongly theoretical character, a corresponding **technical toolset** shows the practical implementation of the 'triad' consisting of language, techniques and method. The technical toolset has an even broader application area, targeting the design of an organization-specific and problem-specific set of EA modeling languages supporting multi-viewpoint EA management. In this vein, the technical toolset shows that the contribution of the thesis is embedded into a larger work targeting the instantiation of a design theory nexus for designing organization-specific EA management functions. While the nexus instantiation described above focuses on language aspects of EA management, the technical toolset covers both core determinants for an EA management function: the EA modeling languages, which are discussed in this thesis, and the method counterpart, which is described in [BMS10d].

1.3 Outline of the thesis

The thesis is structured into nine chapters, of which the first motivates the problem statement and briefly introduces the readers to the relevant abstractions and conceptualizations. It further summarizes the thesis core contributions, provides an overview on the taken research perspective, outlines the course of the thesis, and delineates the writing conventions guiding the document.

In Chapter 2—**Research Design**—we describe the theoretic underpinnings of the thesis' research. This exposition starts with a discussion on the epistemological, ontological and linguistic positions along the epistemological framework of Becker and Niehaves (cf. [BN07]). Subsequently, we delineate the key concept of the "design theory" [WWES92, GJ07] and related it to the concept of the pattern that is relevant for our research method. Proceeding, we revisit the concept of the "design theory nexus" [PHB08] that can be used to interrelated competing design theories. Finally, we describe the research method taken in this work.

In Chapter 3—**Analyzing the State-of-the-Art in EA Modeling**—we revisit prominent languages for EA modeling. For the analysis we use a classification framework based on the ISO standard 42010 [In07] and on the work of Simon [Si96]. The framework establishes seven dimensions covering a structural perspective on the EA and cross-cutting of the EA. Finally, we analyze to which extent the languages are configurable to an organization-specific EA modeling language.

In Chapter 4—**Towards a Building Block-Based Method for Designing EA Modeling Languages**—we introduce the core idea of the thesis. Firstly, we revisit the *language appropriatenesses* of Krogstie [Kr02] and elicit quality criteria for EA modeling languages. Secondly, we elicit a set of requirements for the development method itself and apply Hevner et al.'s guidelines for design research [He04]. Based on relevant modeling fundamentals, we give an overview on the BEAMS and of their constituents. Therein, we identify the underlying organized library of building blocks with an instantiation of a design theory nexus and understand the different types of building blocks as the design theories.

In Chapter 5—**BEAMS: Building Blocks for EA Management Solutions**—we detail the development method and its constituents. Therefore, we concretize our understanding of EA management functions and describe how the method-language dichotomy is reflected in our solutions via two distinct types of building blocks: **method building blocks** and **language building blocks**. Latter type of building blocks, which is of key interest for developing EA modeling languages, is further specialized into three types of building blocks. We identify the syntax-describing *information model building blocks* (IBB) with **areas-of-interest** in the EA and describe how these areas relate to each other in a way usable for configuring EA modeling languages. We delineate how notation-describing *viewpoint building blocks* (VBB) can be used to develop EA viewpoints grounded in a consistent glossary constituted from *glossary building blocks* (GBB). The development method uses building blocks of all three types to constitute an organization-specific EA modeling language. The **administration method** describes how new building blocks can be added to the knowledge base of our approach and how the organizing relationships between the building blocks are derived.

In Chapter 6—**Foundations of BEAMS**—we establish the formal foundations for BEAMS. Central thereto is a conceptual framework for multi-viewpoint EA modeling that describes how different EA modeling languages interrelate based on a common **conceptualization** of the EA. We detail on the **consistent embedding**-relationship that has to be accounted for, while different EA modeling languages are integrated. A dedicated meta-language for describing EA information models, the *information model meta-language* (IM2L), supplies the formal underpinnings needed to operationalize the consistent embedding-relationship. It further provides domain-specific language concepts for EA modeling. The *viewpoint definition language* (VDL) establishes a formal basis for describing and integrating viewpoints and VBBs. Further, it supplies the relevant links to understand how the viewpoints' underlying viewmodels embed into the organization's embracing information model. A dedicated meta-model for describing glossaries complements the formal foundations of BEAMS.

In Chapter 7—**Implementation Aspects of BEAMS**—we describe use cases for a toolset that supports the building block-based development method. This tool builds on the organized library of language building blocks and supports language development using the guidelines, meta-languages and techniques of our method. We subsequently delineate a prototypic design for such a toolset and discuss issues related to the implementation of the toolset. Both design and implementation establish relationships to existing relevant tools as model repositories, EA management tools, or visualization frameworks.

In Chapter 8—**Evaluating and Applying BEAMS**—we evaluate BEAMS in two distinct ways. Firstly, we revisit the quality criteria for a 'good' EA modeling language and show that these hold for any EA modeling language(s) designed using our development method. In addition, we reflect the requirements for the development method itself and apply the general characteristics of design artifacts according to Hevner et al. [He04] to understand the quality of the provided research outcome. Secondly, we discuss the development method against the background of the state-of-the-art literature. Thereby, we show that using the development method, a skilled enterprise architect can come up with a similar EA modeling language as provided in the literature examples. We also show how the building block-based method supplies innovative additions to the literature approaches. In addition, we describe observations made during the application of the method in cooperation with an industry partner.

Chapter 9—**Summary, Critical Reflection, and Outlook**—provides a summary of the thesis' contribution. We reflect the assumptions underlying the research design and identify possible limitations of the method taken. Furthermore, we derive open questions and describe directions for future research furthering the thesis' contribution. The outlook concludes with reflections on the applicability of the taken research design on other fields of investigation in the area of EA management and in bordering research areas.

1.4 Writing conventions

The writing conventions outlined below are employed to improve overall readability and foster comprehension. A notice on the usage of the personal pronoun "we" is added. The thesis was written by a single author, who also developed the thesis' core contribution. The research leading to this thesis was nevertheless not conducted in isolation but in an active group of researchers. Therefore, I use the term "we" to denote research activities as well as findings and contributions that were created in cooperation with the research team at the Chair for Software Engineering of Business Information Systems at Technische Universität München. Quotations referring to articles and papers written in cooperation with this team are opened with "we". Contrariwise as I do not want to seize merits for articles and papers that were developed in a joint effort with other research groups, these works are quoted using the names of the authors, or the first author, respectively.

Important terms are highlighted **this way**, indicating relevant statements on the specific term. Not all occurrences are highlighted for reasons of an uninterrupted flow of text, although we make exceptions from the former rule, when adding the term without highlighting would result in an overly complex sentence. Central terms in the work are defined in boxes as follows.

Definition: Definition
A definition summarizes the thesis' understanding of a certain term and contributes to the understanding of the research field as assumed in the work.

At some points in the course of the thesis, it might not be possible to give a final definition for a term. The reader might yet not have enough information to fully comprehend the definitions constituting terms. To accommodate for this fact, preliminary definitions for terms are given as follows.

Preliminary definition: Preliminary Definition
A preliminary definition gives a statement describing the meaning of a term in an intuitively understandable way.

As another means of text organization, quotations from related work and primary sources are highlighted. A nearby reference to the authors' names and a citation provide information on the source of the quoted statement. Concerning the length of the quotation, we employ two ways of highlighting: short quotations and single terms are emphasized by "quotation marks". Longer quotations are organized as block-quotes as follows.

> This is an example of a longer quote.

In quotations but also in paragraphs or sections dedicated to paraphrasing the contents of a related work, we cite the original terminology as used throughout that work. Exceptions to this are additions in quotations, enclosed in [brackets]. If the original terminology cannot be mapped unambiguously to the thesis' terminology, a footnote provides additional information on the correspondence of terms. We highlight terms borrowed from the foreign terminology in *this way*.

Examples are used throughout the thesis to illustrate complex concepts and contexts. To improve readability of the text and to unambiguously highlight the examples, they are presented in a box as follows.

Example 1.1: Exemplary example. This is an example.

Conceptual models play an important role throughout this thesis. In order to support an unambiguous reference to model elements or concepts, we highlight termini referring to such concepts THIS WAY. In addition, code and code-like statements, e.g. modeling constraints, use a type writer font face (`ModelElement`). Colloquial terms are used at different points in the thesis to avoid complex circumscriptions, to smoothen the text flow, and to increase understandability. An example of such term is 'gut feel'. These terms are highlighted by single quotation marks. A final remark on writing conventions for job titles or references to persons has to be made. We use plural forms referencing both to female and male representatives of the corresponding job title with no disrespect for either gender intended.

The intent of a question is sometimes as important as the answer

Jeffrey Sinclair, Eyes—Babylon 5

CHAPTER 2

Research Design

In the introductory Chapter 1 we identified the relevant lack of a development method for organization-specific EA management functions. In this chapter, we outline how such development method can be rigorously researched, i.e., we describe our *research design*. In line with Becker et al. [Be03, page 5] a research design consists of a "research method" aiming at certain "research outcomes" and is grounded in well-defined "ontological", "epistemological", and "linguistic assumptions". Becker et al. further state that method(s), outcomes, and assumptions are interdependent, and have to match each other to form a consistent research design that can be executed and validated. Figure 2.1 summarizes the fundamental make-up of a research design.

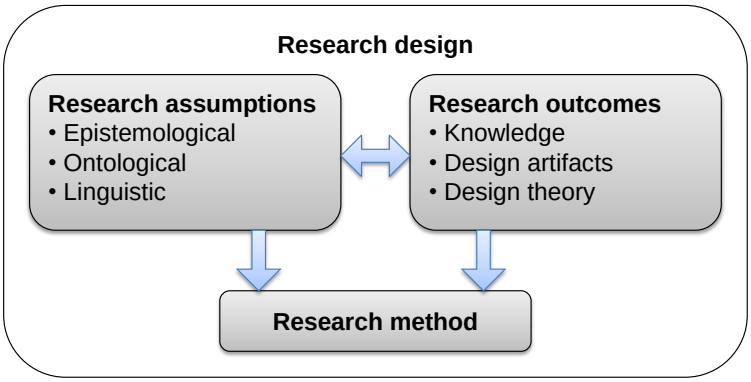


Figure 2.1: Constituents of a research design according to Becker et al. [Be03, page 5]

The research assumptions of this thesis mirror those of the *critical linguistic approach* of Kamlah and Lorenzen [KL67]. In Section 2.1, we discuss these assumptions along the classification framework for the epistemology of IS research presented by Becker and Niehaves in [BN07].

Our research does not seek to develop one organization-specific EA management function (a *design artifact* in terms of Aier et al. [Ai08b], van der Raadt and van Vliet [RV08] as well as Ernst [Er10]), but focuses on the development, i.e., the *design process* for such function.

Section 2.2 discusses the intended research outcome of this thesis, namely a method for developing EA modeling languages based on best-practices. Therefore, we develop methodical support for the development processes with concrete and explicit design prescriptions. In [WWES92] Walls et al. introduce the notion of the *design theory*, i.e., of a scientific theory providing prescriptions on designing effective solutions to *wicked problems* in terms of Rittel and Weber [RW73]. The development of an organization-specific EA managementfunction is a *wicked problem* in this sense, as the typical characteristics [RW73, BJ87] of such problem are fulfilled:

- EA management is subject to unstable requirements and constraints in ill-defined environmental contexts.
- EA management exerts complex interactions among it methods and languages.
- EA management cannot be easily changed and adapted in the enterprise context.
- EA management critically depends on human cognitive abilities, especially for conceptualizing the architecture.
- EA management critically depends on human social abilities, especially for communicating the architecture.

The plurality of EA management-related problems is reflected in the plurality of design theories with the individual theories targeting a specific problem in a particular context. In this sense, users willing to rely on the theories in developing their EA modeling languages face the challenge to find the theories most suitable for the specific application. The 'formalization' of the prescriptions as design theories provides only a partial solution. To facilitate the selection of the theories best suited for the specific context and problem, we interlink them in a *design theory nexus instantiation* (cf. Pries-Heje and Baskerville [PHB08]). Such nexus instantiation relates the different design theories to criteria that promote or impede their application. This instantiation forms the basis of the methodical support provided as part of the research outcome.

Section 2.3 introduces a method for instantiating the design theory nexus. In this method we combine the general design science research method of Venable [Ve06] with the stepwise procedure for constructing a design theory nexus as described by Pries-Heje and Baskerville in [PHB08]. Regarding the relevant aspect of evaluation [He04], we further integrate evaluation and validation techniques proposed by Verschuren and Hartog [VH05] into the method. The final Section 2.4 summarizes the research design underlying this thesis.

2.1 Epistemological, ontological, and linguistic assumptions

We understand a *development process* in line with Hirschheim et al. in [HKL95, page 15] as "a change process taken with respect to object systems in a set of environments by a development group to achieve or maintain some objectives". An *object system* mirrors the perception of

reality by the members of the development group. Therefore, a comprehensive description of any development process has to account for the "philosophical position on the nature of reality and human inquiry" (cf. Hirschheim et al. [HKL95, page 15]). Becker and Niehaves devise in [BN07] an *epistemological framework* that allows more detailed considerations on the philosophical position, i.e., the *ontological* and *epistemological assumptions* underlying a research process. Explicit assumptions are needed for reasoning on research rigor, as different research methods and techniques depend on distinct epistemological assumptions. In the following we answer the five "epistemological questions" of Becker and Niehaves [BN07, page 202] and describe which of the perspectives outlined in Figure 2.2 is taken for which aspect.

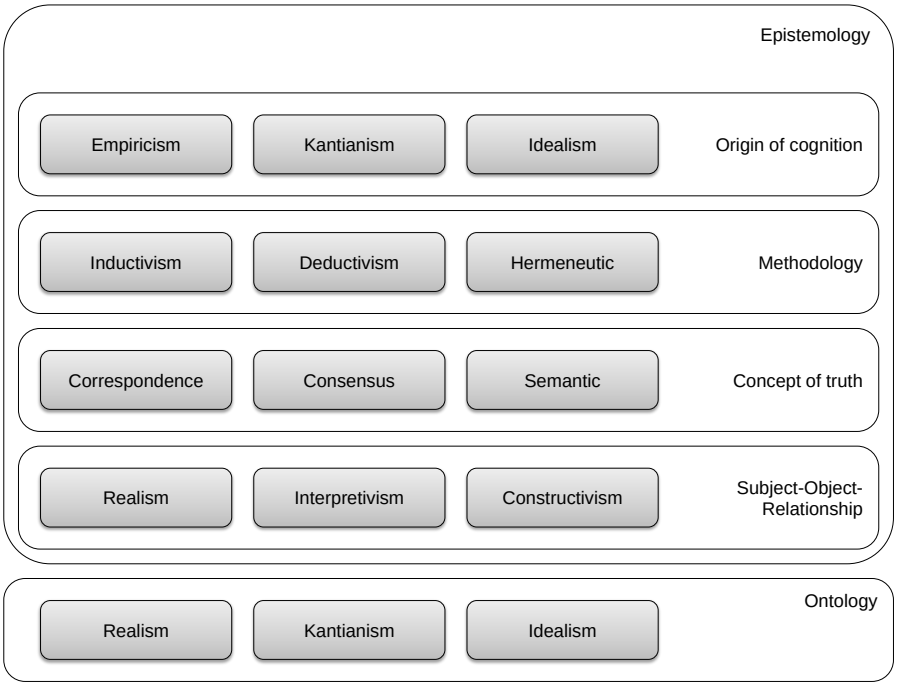


Figure 2.2: Framework of epistemological and ontological assumptions in research [BN07]

2.1.1 Ontological aspect

The framework of assumptions is grounded on the ontological assumption, namely the answer to the question about the existence of a 'real' world, i.e., "what it is" [Fo96, page 366]. Reformulated to the context of a research design, this question reads as: "What is the object of our research?" [Mo03]. Possible perspectives are:

- The *realist* perspective assumes the existence of a world independent from human cognition as well as from human acts to communicate facts. An extreme form thereof, namely *materialism*, assumes that all objects and interactions are matter and physical processes [St98].
- The *idealist* perspective assumes the existence of reality independent from a 'real world', i.e., claims the "primacy of spirit over matter" [BN07, page 203].

- The *kantianist* perspective mediates between the two extremes, introducing a distinction between *noumena*, which are real world entities, and *phenomena*, which depend on human cognition.

Perspective taken in this thesis
Kantianism limits human knowledge to knowledge about phenomena, while the things themselves remain unknowable. This assumption fits the research area of EA modeling languages. For many of the EA-relevant concepts, especially on the application and infrastructure layer, a real world existence as things in themselves can be assumed. Cross-cutting aspects as strategies and goals, or principles and standards present themselves as phenomena, i.e., things dependent on human cognition.

2.1.2 Subject-object-relationship

The epistemological aspect targets the relationship between the subject and the object of cognition. Becker and Niehaves formulate in [BN07, page 203] the question, "whether entities beyond human thoughts and speech can [...] be recognized as objective". Two different assumptions with this respect can be taken:

- The *realist* position assumes that objective cognition is possible and eliminates subject-dependency in cognition, i.e., claims that subject-dependent distortions of reality cognition can be eliminated [Lo01, pages 64–66].
- The *constructivist* position understands every cognition as subjective, i.e., 'private'. Therefore, the subject can explicitly decide on the actual quality of the relationship and can influence the cognition of the object [Lo87].

The epistemological perspective is not independent from the ontological perspective. Reasonably, epistemological realism can only be grounded on ontological realism [BN07, page 203]) with the combination thereof being referred to as *positivism* [CH04]. In their most extreme forms, above epistemological perspectives remain largely incommensurate. According to Becker and Niehaves [BN07, page 203], the spectrum of constructivist perspectives ranges from *interpretivism* to *radical constructivism*. The latter position radicalizes the constructivist idea to the point, that the existence of a thing as an 'objective reality' is neglected. The interpretivist position is more moderate: an 'objective reality' might exist, but is always interpreted by the inquirying subject.

Perspective taken in this thesis
In this work, we take a constructivist, more precisely an interpretivist, perspective. The enterprise and its architecture, are perceived by a multitude of stakeholders (cf. Aier et al. in [Ai08a] and [Ai08b]) that differently interpret and conceptualize the 'reality' based on their subjective backgrounds. As we discuss in [Bu09a, page 8] different stakeholders of the EA have different educational backgrounds. These differing backgrounds cause manifold different interpretations of the reality of the enterprise by the stakeholders.

2.1.3 Concept of truth

Central to any kind of cognition is the concept of truth that the inquiring individual assumes. Paraphrased in terms of Becker and Niehaves [BN07, pages 203–204], this leads to the question "how humans can achieve true cognition", i.e., can obtain knowledge, whose correctness can be verified. Three different positions on the concept of truth can be distinguished:

- The *correspondence theory* of truth builds on the equivalence of *relata*, more precisely of a *statement* and a *fact* from reality. The truthfulness of a statement is determined by the existence of an equivalence relationship.
- The *consensus theory* of truth claims that true statements originate from consensus among the people belonging to a relevant group. This understanding of truth emphasizes the importance of the subject-object-relationship that determines the truthfulness and relevance of a statement.
- The *semantic theory* of truth does not directly provide a definition of truth, see Tarski [Ta44]. In contrast, it gives a *linguistic condition* for truth based on the relationship between a language and its meta-language according to which truthfulness means that true language statements commit to ones in the ontological meta-language.

The linguistic condition embodied in the semantic theory of truth is a priori fulfilled by the utilization, when conceptual modeling languages based on well-grounded meta-languages are used [BN07, page 204]. The semantic theory of truth provides another contribution to the epistemological discussion and especially to considerations on modeling languages. Tarski rises in [Ta44] a constraint on the *self-referentiality* of statements. He discusses that self-referential statements, i.e., ones containing a statement on the itself's truthfulness can lead to a variety of antinomies and paradoxes. On the level of general logic, Tarski's constraint on self-referentiality relates to *Curry's paradox* [Cu42]. He shows that any logic allowing that an expression can contain a statement on the truthfulness of another expression is fallible to implicit contradictions. While we abstain from more elaborate discussions on the aforementioned paradoxes, we shorthand an important consequence from Tarski's constraint:

> Any modeling language must to avoid paradoxes explicitly distinguish between concepts and statements on different levels of ontologic metaization.

The structure of the research outcomes adheres to this rule and provides distinct languages for the different meta-levels.

Perspective taken in this thesis
Two different concepts of truth are applied in this work, similar to many other works targeting the field of conceptual modeling [BN07, pages 206–209]. On the one hand, language-based conceptual modeling builds on Tarski's semantic theory of truth, as the truthfulness of statements in a modeling language can be assessed against the corresponding meta-language. On the other hand, language-based modeling relies on the consensus theory of truth, i.e., understands statements as true in respect to a certain group of model creators and users. In [KL67] Kamlah and Lorenzen coin the term **linguistic community** for the group of people, who consent on the truthfulness of a language model. Further, when constructs as infrastructure devices and technical measures as latency are concerned, consensus can be operationalized towards a *subjective correspondence mechanism* [KL67].

2.1.4 Origin of cognition

People can draw cognition from different sources, which mirror the different ways to perceive and to intellectually conceive concepts. Becker and Niehaves summarize the corresponding epistemological aspect in the question "Where does our knowledge derive from?" [BN07, page 205]. Three answers are formulated between two extreme positions:

- The *empiricist* perspective regards experience as the primary source of knowledge, i.e., imposes a primacy of sensual impressions over intellectual conception. Experience-based knowledge is called *a posteriori* knowledge in line with Alavi et al. [ACB89].
- The *rationalist* perspective regards the intellect as true source of cognition. An object becomes a matter of cognition through a subject's conceptual efforts [BN07, page 205]. Knowledge obtained in this way is called *a priori* knowledge.
- In the *kantianist* perspective, both experience and intellect can and need to be sources of cognition. For the central statement of the perspective, we quote Kant [Ka08]: "experience gives us the objects, while intellect gives us the categories".

Perspective taken in this thesis
Both empirical observations of architectural properties as well as intellectual imaginations are valuable sources of knowledge in EA modeling. Especially, the cognitive nature of categories is very relevant in EA modeling. These categories build the basis for the concepts and types of the EA modeling languages.

2.1.5 Methodology

The final set of assumptions is concerned with the question of "how humans perceive" [BN07, page 205]. The assumptions target the methodology of perception, which is closely related to the selected research method. Further, the methodology is strongly interrelated to the research outcomes such that the research method provides. Three different methodological positions exist:

- *Inductivism* generalizes from individual cases to universal statements. In this way, the formulation of universally applicable laws is preceded by cognition, i.e., perception or intellectual efforts, that means inductivism is a *posteriori* method.
- In *deductivism* one or more universal statements are used as basis to derive less general, i.e., case-specific, statements. Logical conclusions are employed to get from the grounding statements, which are sometimes referred to as *axioms*, to the individual statements.
- Any *hermeneutic* method employs the principle of the *hermeneutic circle* as described by Gadamer in [Ga75]. The hermeneutic cycle originally aims at the comprehension of texts, but can more generally be interpreted as iterative method to interpret an observation against the "background of previous understanding of the entire" [BN07, page 206]. Interpreting the observation conversely shapes the understanding of the entire.

IS research does often not adopt a single methodological position, but combines different methods. Becker and Niehaves [BN07, pages 206–209] see single-methodical research as a prerogative of natural sciences (inductivism) and mathematics (deductivism).

Perspective taken in this thesis
In the light of preceding research in the field of EA modeling languages, a hermeneutic method is appropriate, allowing for a reconciliation of past practical observations. Thereby, the understanding of the field is developed and refined, inductively generalizing the findings from existing modeling languages, see Section 2.2.

2.1.6 Linguistic perspective

Languages, not only modeling languages, play an important role in IS research. In the following, we discuss the linguistic perspective of this work along a framework of Becker et al. [Be03, pages 8–10]. The framework distinguishes three functions of language:

- The *cognitive* function of a language refers to the language's role in the intellectual efforts of its users. In this vein, the cognitive function is purely *intrapersonal*, i.e., refers to 'inner' processes of human cognition. In line with Wittgenstein [Wi53, page 107] we assume that some sort of language is used during cognitive processes. More elaborate discussions on the cognitive function of language are inconclusive as intrapersonal language utilization is not observable.
- The *expressive* function of a language refers to the question how explicit language artifacts, e.g. texts or diagrams, are equipped with semantics. While under the assumption of ontological realism any language artifact relates to an unambiguous semantics, a kantianist or idealist perspective calls for a more differentiated understanding of language semantics. Under the latter assumptions, the semantics of a language artifact can depend on the subject, who uses it.
- The *communicative* function of a language accounts for the *interpersonal* dimension of language use, i.e., refers to the question to which extent language artifacts can be used to convey information to subjects. In the light of subject-dependent semantics of language artifacts, it is of interest to investigate the communicative function of a specific language.

In the context of this work, the expressive as well as the communicative function of a language deserve special attention. In line with the critical linguistic approach of Kamlah and Lorenzen [KL67], Becker et al. discuss in [Be03] the concept of the **linguistic community**, on which our subsequent considerations on language utilization are grounded.

Preliminary definition: Linguistic community
A linguistic community is a group of individuals, may they be researchers or practitioners, that agree on a defined set of terms, i.e., 'frozen' language artifacts, as well as on a shared understanding on the terms' semantics.

We subsequently discuss how linguistic communities are relevant for our work. Firstly, we can in line with Schelp and Winter [SW09] assume that every approach to EA management in literature forms its own community. We must account for this fact, when we explore the state-of-the-art in EA modeling in Chapter 3. Secondly, the use of EA modeling languages in an enterprise relates to the differing linguistic communities that exist within an enterprise.

In this vein, our development method for EA modeling languages, must specifically account for linguistic plurality in an enterprise. This leads to additional contextualizing requirements regarding the research outcome.

2.2 Research outcomes

The notion of the design theory is central to the research outcomes presented in this work. In line with Walls et al. [WWES92, page 37] we paraphrase the meaning of the term as follows: "A design theory is a prescriptive theory based on theoretical underpinnings which says how a design process can be carried out in a way which is both effective and feasible". The prescriptive nature differentiates design theories from *analytical*, *explanatory*, and *predictive* theories according to the theory classification of Gregor [Gr06, page 12]. These types of theories are the theories that March and Smith [MS95] allude to, when they reject theorizing in design science. Design theories nevertheless are relevant outcomes of design science research. In the following Section 2.2.1 we detail the concept of the design theory, proceeding from the earlier work of Walls et al. [WWES92] to the more recent definition provided by Gregor and Jones [GJ07]. In Section 2.2.2 we relate design theories to the concept of *patterns* and explain how patterns can be used to devise new design theories. In Section 2.2.3 we discuss the concept of the design theory nexus introduced by Pries-Heje and Baskerville in [PHB08]. A theory nexus relates different design theories applicable on the same design object in different contexts and facilitates the selection of the most appropriate design theories. Thereto we also revisit the concept of the pattern as well as the closely related conceptualization of the *pattern language* according to Alexander et al. [Al77].

2.2.1 Design theories

In contrast to 'conventional' design science artifacts in terms of March and Smith, design theories do not provide *situated* solutions to specific *problems* in designated *contexts*, but make prescriptions how a *class of problems* can be solved (cf. Venable [Ve06, page 4]). This class of problems forms the **problem domain** of the design theory, whose problems are solved by corresponding concepts from the theory's **solution domain**. In line with Hevner et al. [He04, page 9] we further introduce the **context domain** that describes the environments in which the design theory can be applied. The following extended definition of the term design theory incorporates also the context domain.

Definition: Design theory
A design theory is a prescriptive theory based on theoretical underpinnings which says how a design process can be carried out to address a problem from the theory's problem domain with a solution from the theory's solution domain in a way which is both effective and feasible in the given context from the theory's context domain.

Walls et al. give in [WWES92, page 43] an exposition of seven components that comprise a design theory. These components are classified into two groups, namely those related to the *design product* and those related to the *design process*. We subsequently present the components, starting with the design product-related ones.

- *Kernel theories* govern the design requirements and can be theories of arbitrary type[1], e.g. from natural or social sciences.
- *Meta-requirements* represent the classes of problems which the design theory applies to.
- *Meta-design* represents a class of artifacts able to meet the meta-requirements.
- *Testable design product hypotheses* are statements that can be used to verify, if the meta-design satisfies the meta-requirements. According to Venable [Ve06] these hypotheses can be tested on the meta-level and on the instance level.

Three more components of a design theory are outlined by Walls et al. in [WWES92, page 43] targeting the design method.

- *Kernel theories* are theories of any type used to govern the design method. They often originate from natural or social sciences.
- *Design process* describes the procedure used to develop a particular, situated instantiation of the meta-design, i.e., the procedure for constructing the design product.
- *Testable design process hypotheses* are statements that can be used to verify, if the application of the design process will arrive at a proper instantiation of the meta-design.

Walls et al. graphically summarize in [WWES92, page 44] the components and their relationships. We give a slightly adapted version of their visualization in Figure 2.3, augmented to differentiate between the types of components, namely the design product- and the design process-related ones by a color-cording and by symbols—blue (○) and red (△), respectively. We further account for Venable's discussion [Ve06, page 17] on the nature of design theories. In the course of the discussion, Venable puts emphasis on the activity of "testing". He discusses that design product hypotheses ('blue' hypotheses ○) can be tested on different levels, namely on the meta-level and on the instance level.

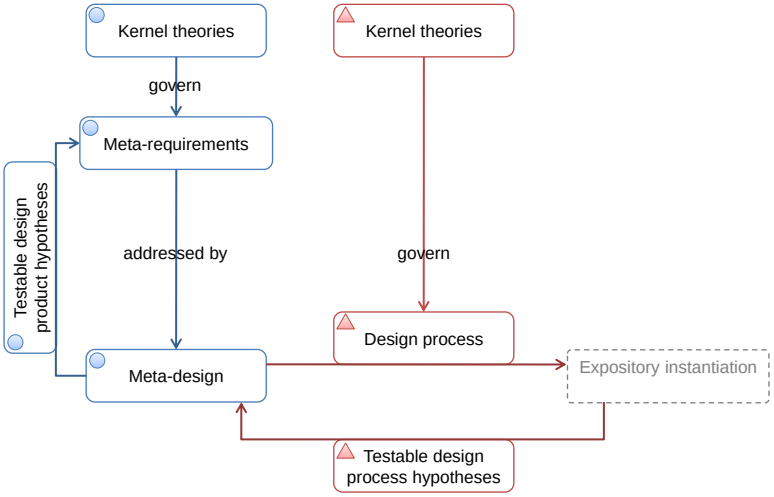


Figure 2.3: Components of a design theory adapted from Walls et al. [WWES92, page 44]

[1] For the classification of theories refer to Gregor [Gr06, page 12].

In a more recent publication [WWES04] Walls et al. discuss that their initial framework for design theories was well-founded in 1992, but retains some potential for improvement. Picking up this argument, Gregor and Jones [GJ07, page 322] present an extended framework. Therein, the term *design artifact* is used instead of *design product* congruently with the discussions of March and Smith [MS95]. Gregor and Jones introduce eight components of a design theory, which are subsequently discussed and mapped to the components as described by Walls et al. [WWES92].

- *Purpose and scope* describe what the design artifact is intended for, i.e., the meta-requirements in terms of Walls et al.
- *Constructs* refer to relevant entities in the context, the problem, and the solution domain, i.e., to descriptions of physical phenomena or abstract terms of interest. Walls et al. do not provide such components for a design theory.
- *Principles of form and function* are prescriptions for the structure of the design artifact, i.e., provide the meta-design in terms of Walls et al.
- *Artifact mutability* describes how the artifact adapts to changes in the context domain. Walls et al. do not account for such concept.
- *Testable propositions* evaluate in a twofold way, on the one hand whether the artifact matches its purpose and on the other hand whether the design method correctly instantiates the artifact. Testable propositions summarize the testable design product hypotheses and the testable design method hypotheses of Walls et al.
- *Justificatory knowledge* describes the base of explanatory knowledge, on which the design theory is grounded. Walls et al. distinguish between knowledge (kernel theories) justifying the meta-requirements and justifying the design method.
- *Principles of implementation* refers to the means by which the design artifact is created. These correspond to the design method of Walls et al.
- *Expository instantiation* describes an exemplary design artifact that helps to present and communicate the theory. The instantiation can also be used to exemplarily validate the theory. Walls et al. do not account for such a concept.

We subsequently summarize the key differences between the frameworks of Gregor and Jones [GJ07] as well as of Walls et al. [WWES92]. Firstly, Gregor and Jones add the *constructs*, which are necessary to build a consistent and shared terminology backing the design theory. The *entity-relationship* (ER) model [Ch76], which is brought forward by Gregor and Jones in [GJ07, page 315] as example of a design theory, is used to further illustrate the importance of constructs. Secondly, Gregor and Jones further an argument of Hevner et al. [He04, page 12] that reads as "design-science research must produce a viable artifact in the form of a construct, model, method, or instantiation". Therefore, Gregor and Jones add an expository instantiation that should be supplied with a design theory. For an overview about the components see Figure 2.4, where the above color-coding and symbols are applied again. In addition, we use another color and symbol (green □) to indicate that components are related to the instantiation and application of the design theory.

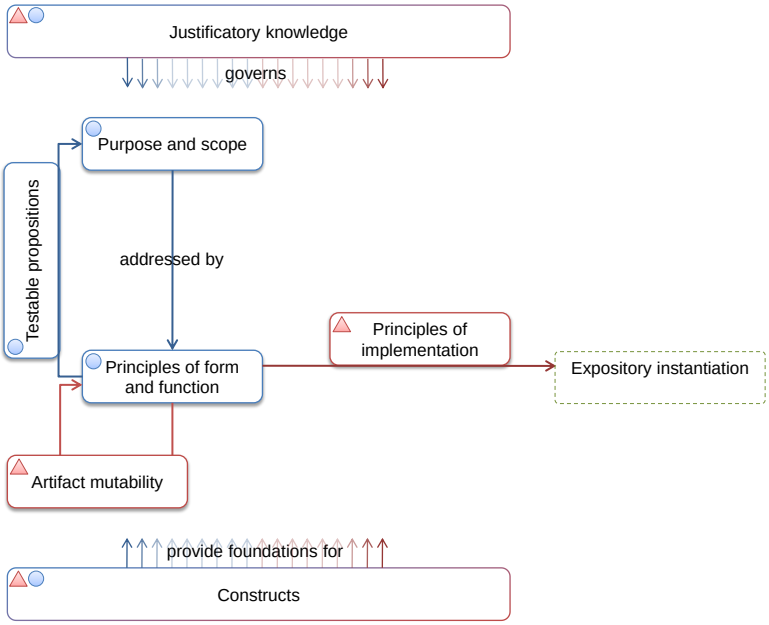


Figure 2.4: Components of a design theory according to Gregor and Jones [GJ07]

2.2.2 Patterns and design theories

In the recent discussion initiated by Winter in [Wi09] different authors reflect on the role of patterns in IS development. Some of the authors highlight that patterns have a long history as documentations of practice-proven design prescriptions in IS and computer science. Gamma introduced the concept of patterns to the latter field in his PhD thesis [Ga92]. Nevertheless, the idea of the pattern is significantly older dating back to Alexander, who introduced in [Al72] patterns as means to document proven solutions to recurring problems in building architecture. As of today, a strong community hosting several regular conferences such as "Pattern Languages of Programs (PLoP)" advances the field of pattern description, thereby staying to the initial definition of the term given by Alexander et al. in [Al77, page x] as follows:

> **Definition: Pattern**
> A pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to the problem, in such a way that you can use this solution a million times over without ever doing it in the same way twice.

Patterns are used to capture and to document proven practices that have been observed as solutions to recurring problems in the real world. The observational nature is highlighted by different authors, foremost Alexander et al. in [Al77], to distinguish patterns from solutions that are invented. Coplien [Co96] puts this particular nature in a pragmatic notion, the so-called *rule of three*. According to this rule, any practical solution can become a pattern, if it has been observed at least three times independently. A solution that has yet not been observed frequently enough is consequently alluded to as *pattern candidate*. Several authors discuss different forms for describing patterns, leading to the distinction between the

Alexandrian form, which is mainly narrative and comes without additional structuring, and the canonical form of Buschmann et al. [Bu96]. This form introduces six components of a pattern description:

- A *pattern name* which identifies the pattern and makes it memorable,
- an *example* illustrating the problem addressed by the pattern,
- a *context description* delineating the environmental conditions in which the pattern can be applied,
- a *problem description* outlining which problem is addressed by the pattern,
- a *solution description* for the solution to the described problem, and
- *links to other patterns*, i.e., references to other patterns (*see also*) solving similar problems potentially in different contexts, or to patterns which help to refine this pattern.

Meszaros and Doble further detail above list of components with a set of meta-patterns for pattern writing [MD97]. They introduce two additional concepts for describing patterns the *forces* as well as an explicit reference to the intended *pattern users*. While latter is self-explanatory, the concept of the forces is defined in [MD97, page 535] as "contradictory considerations that must be taken into account when choosing a solution to a problem". The pattern's solution delineates how these forces can be resolved. Further, the authors list several optional elements of a pattern description: possible *indications* for the existence of the problem, a *rationale* explaining the appropriateness of the pattern's solution, *aliases*, i.e., alternative names, *acknowledgments*, and the *resulting context*. A concept similar to the resulting context is found in other pattern forms, e.g. used by Gamma et al. [Ga94] or by Ernst [Er08]. These contexts describe the situation "that we find ourselves in after the pattern has been applied" [MD97, pages 536–537] and could include one or more new problems.

Above characteristics of patterns indicate a conceptual relationship between design theories and patterns. In [SBK07b] Schermann et al. establish a conceptual structure of pattern-based design theories amalgamating the pattern structure of Buschmann et al. [Bu96] with the structure of a design theory according to Walls et al. [WWES92]. In [BMS10k] we discuss another perspective on the relationship between patterns and design theories. This perspective builds on the principle of *abstraction* discussed by Vaishnavi and Kuchler in [VK04]. Abstraction is applied in different ways to get from situated implementations (solutions) to design theories. The two step abstraction via the intermediary concept of *knowledge as operational principles* is to be distinguished from the one step abstraction, see Figure 2.5.

The knowledge as operational principles provides an intermediary level of abstraction that is of high relevance for design theorizing in practice-driven and practice-based research. It is not sensible to expect practitioners to abstract their implementations towards 'full-blown' design theories. Nevertheless, knowledge sharing between different practitioners almost inevitably leads to abstracted representations of implementations. These abstractions are identified with the abstraction level of operational principles. Abstraction to operational principles can also be applied on case-based research. Eisenhardt and Graebner describe in [EG07] the role of cases in theory building, especially for *phenomenon-driven* research. A relevant factor thereof is to renounce from a focus on representative cases and to embrace 'disruptive' cases that can provide revelations of an unusual phenomenon. Singular cases or infrequent observations contribute to the knowledge base and can be documented as patterns or pattern candidates.

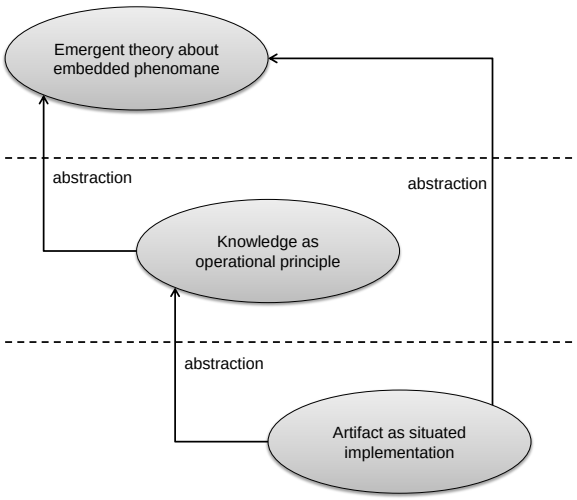


Figure 2.5: Levels of abstraction in theorizing according to Vaishnavi and Kuchler [VK04]

In [BMS10k] we discuss how to use patterns to represent and describe knowledge as operational principles. Thereto, we make use of two characteristics of patterns. Firstly, patterns provide practice-proven design prescriptions in both IS and computer science (cf. Winter [Wi09]), being especially of interest for practitioners. Secondly, the canonic pattern structure according to Buschmann et al. [Bu96] as well as to Meszaros and Doble [MD97] contains the relevant key constituents of a design theory. Our conceptual framework for pattern-based design theories [BMS10k] builds on the anatomy of design theories outlined by Gregor and Jones [GJ07]. Figure 2.6 depicts the framework thereby adopting the color-coding introduced above to distinguish between process-related (red $\triangle$) and artifact-related (blue $\bigcirc$) constituents of the design theory. Green symbols ($\square$) are further used to represent elements resulting from the instantiation of the design theory.

In our considerations on the relationship between patterns and design theories [BMS10k], we emphasize role of the *constructs*. As Carlson describes in [Ca07], design knowledge must be made explicit using a terminology accessible to the intended audience. For patterns, especially ones documenting operational principles, such terminology is the one of the using practitioners in the organization. A design theory must be more general, such that the terminologies of different theories must be aligned. The terminology used therein forms a conceptualization of the context, problem, and solution domain interrelated by the design principle. Seeking to avoid confusion with *concepts* and *constructs* in the sense of March and Smith [MS95, page 253] that would in turn be part of the actual design artifact, we rename the elements of the theory's terminological basis to **meta-concept**.

In Table 2.2.2 we provide a comparison of the terms used to designate the components of design theories. The comparison follows the one provided by Gehlert et al. in [Ge09] and includes the pattern-structured design theories of Schermann et al. [SBK07b]. In the table we further highlight the terms that we adopted in [BMS10k].

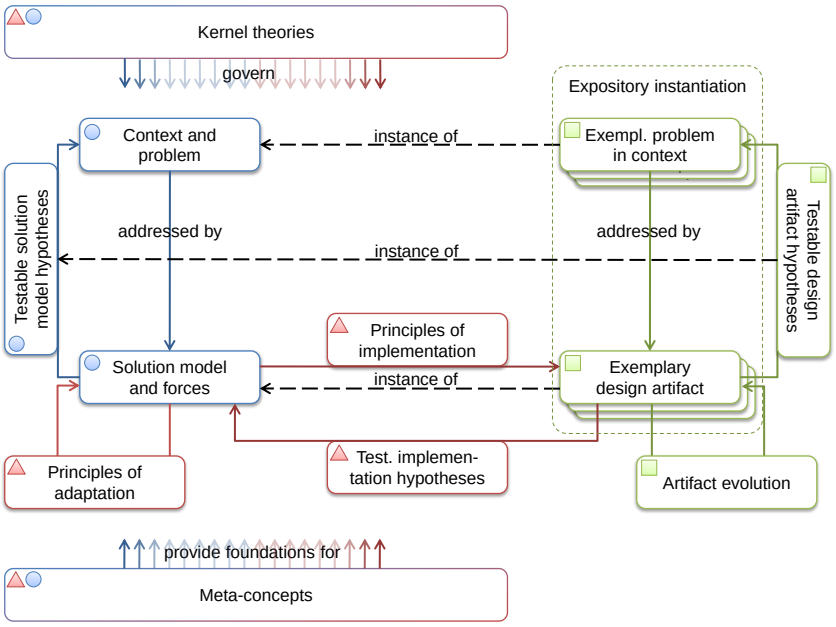


Figure 2.6: Components of a design theory according to Buckl et al. [BMS10k]

Walls et al. [WWES92]	Gregor and Jones [GJ07]	Schermann et al. [SBK07b]
Meta-requirements	Purpose and scope	Context and problem
Meta-design	Principles of form and function	Solution model
Kernel theories	Justificatory knowledge	Theory references
Design process	Principles of implementation	Design method
-[1]	Artifact mutability[2]	Pattern references
Testable hypotheses	Testable propositions	Consequences
-[3]	Expository instantiation	(Instantiation)[4]

[1] Walls et al. do not account for the concept of artifact mutability in [WWES92].
[2] We use the term **principles of adaptation** in [BMS10k].
[3] Walls et al. do not demand an expository instantiation in [WWES92].
[4] Schermann et al. do not provide an explicit name for this concept in [SBK07b].

Table 2.1: Comparison of terms used by the different approaches to design theorizing

2.2.3 Design theory nexus and pattern languages

Pries-Heje and Baskerville discuss in [PHB08] the difficulties of design activities solving problems in environments that are subject to various contingency factors. These factors contextualize potential solutions, i.e., determine, if a particular design solution or theory is applicable. Addressing such design problems, Pries-Heje and Baskerville propose the construct of the *design theory nexus* [PHB08, page 731] defined as follows:

Definition: Design theory nexus
A design theory nexus is a set of constructs and methods [for constructing models] that connect numerous design theories with alternative solutions.

They apply the idea of the nexus to a specific design problem and derive a *design theory nexus instantiation*. Any instantiation employs five types of constructs, see also Figure 2.7: *goals*, *environment*, *competing design theories*, *theory nexus*, and *design solution*.

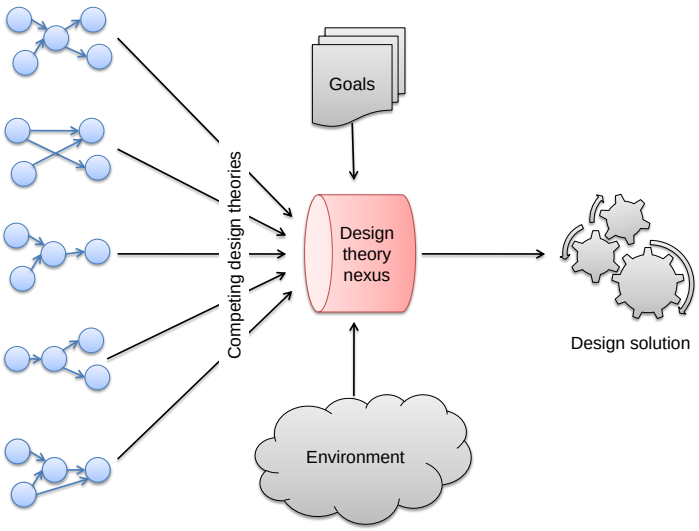


Figure 2.7: Constituents of a design theory nexus

A nexus-based approach targets "ill-structured" design problems [PHB08, page 732]. Ill-structured thereby refers to the criteria that are used to decide about the applicability of a design theory, namely the design goals and the design environment. Thereby, we can in line with Pries-Heje and Baskerville distinguish two different characteristics of a criterion. A criterion can be

- *symmetric*, i.e., can apply to all alternative design theories with different characteristics of that criterion linking to different theories, and
- *asymmetric*, i.e., can apply to only a subset[2] of the design theories with one or only a small number of admissible characteristics linking to the design theories.

For a set of alternative design theories linked only via symmetric criteria different 'classic' multi-criteria decision making approaches apply. Pries-Heje and Baskerville delineate in [PHB08, page 732] how different approaches to support users in determining the optimal solution are bound to fail, when asymmetric criteria are prevalent in the specific design space. The nexus-based approach remedies a key issue of contingency theory [Ba90]. A nexus instantiation does not preclude a static relationship from the structure of the problem and context domain to the solution techniques. This means that approaches based on contingency "pre-program" the solutions [PHB08, page 735] by the criteria shaping the structure of the problem and context domain, and henceforth apply only to symmetric criteria. Structuring principles

[2] In extreme cases an asymmetric criterion applies to only one design theory.

used in a design theory nexus instantiation can provide a basis for combining the different design theories. They do not need to embracingly cover all linked theories with the same basic integration mechanisms.

We distinguish two categories of relationships between the design theories linked to the theory nexus instantiation: *terminological relationships* and *prescription-dependent relationships*. The terminological relationships further the discussions of Carlson [Ca07]. The different design theories interlinked in a nexus instantiation can originate from different backgrounds and do not necessarily build on a common terminology. Therefore, it is relevant to establish terminological relationships interlinking synonymous meta-concepts of the different theories. Prescription-oriented relationships target the context, problem, and solution domains of design theories. With respect to context and problem, two design theories can denote characteristics that are *conflicting*, *independent*, or *subsuming*. Subsuming describes that whenever one characteristic is fulfilled also the other characteristic hold. Further, different theories can relate via their resulting context and a context designating a situation,. Thereby, one theory can be considered a potential prerequisite to another one.

In [Al77] Alexander et al. introduce the concept of the *pattern language*, which represents a system of interrelated patterns. Schermann et al. discuss in [SBK07b] how these relationships can be translated to relationships between design theories, as represented by the patterns. Their discussions nevertheless remain confined to untyped relationships in the sense of *see also* (Alexander et al. [Al77]) or *related patterns* (Coplien [Co96]). In [No98] Noble analyzes the existing pattern relationships in a pattern language and derives a set of semantically richer relationship types. In [BMS10k] we describe how these relationship types provide a contribution in design theory nexus instantiation, i.e., a design theory nexus can be instantiated from a pattern language in a similar way as a particular design theory can be developed in a pattern-based manner.

Walls et al. describe in [WWES92, page 46] the *information system generator*, i.e., of a tool having "the capabilities [...] to construct information systems". An IS generator implements the prescriptions of a single design theory, in particular the conceptualizations of problem, context, and solution domain. The generator can be configured by selecting a specific problem and context from the admissible domains and creates an IS implementing the corresponding solution. A nexus-based IS generator furthers the basic principles of the IS generator in two ways:

- decision support for selecting the appropriate design theories from the admissible ones.
- integration support for combining the solution prescriptions from the selected design theories.

Therefore, the nexus-based IS generator supplies the user with capabilities to specify the relevant characteristics of both problem and context domain. Based thereon, the generator filters the set of applicable design theories, from which the user selects one or more. The user then configures the selected theories, whereby the nexus-based IS generator ensures consistency between the theories with respect to both terminological and prescription-dependent relationships. In addition, the generator automatically integrates the configured theories as far as possible.

2.3 Research method

Multiple research methods for design science research have been proposed in recent years, see e.g. [MS95, He04, SBK07a, Of09]. In the light of the controversial discussions on the role of theorizing it is not surprising that only a few of these methods explicitly target the development of design theories. A relevant method is the one of Venable [Ve06, pages 16–17], which accounts for the iterative character of theory building. Theory building is accordingly understood as linking point between the design activities, *problem diagnosis*, *artifact design*, and *evaluation*, see Figure 2.8.

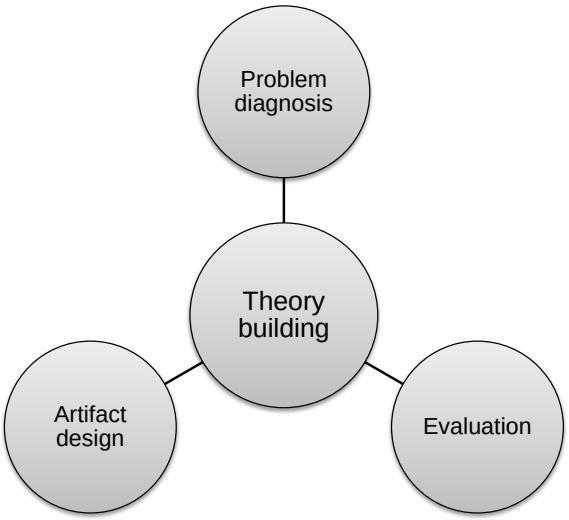


Figure 2.8: Research activity framework of Venable (cf. [Ve06, pages 16–17])

We have proposed in [BMS10k] a research method for pattern-based design theorizing consisting of the steps: observe and document design patterns, elicit pattern terminology, devise pattern-relationships, and derive terminological compatibility relationships. This method on the one hand relates to Venable's research method, but also establishes a link to the method for instantiating a design theory nexus [PHB08, pages 737–738] with the following steps:

1. identify alternative approaches,
2. analyze alternative approaches,
3. formulate assertions for the approaches' criteria,
4. develop a decision making process, and
5. integrate approaches, assertions, and process into a tool.

In the following, we synthesize a research method from the three methods outlined above. Thereto, we identify the steps of nexus instantiation [PHB08] with the activities of Venable [Ve06]. Step 1 and 2 perform the problem diagnosis, steps 3 and 4 form the activity of theory building, and step 5 describes a part of artifact design. The application of the nexus instantiation further contributes to the evaluation, such that for our method the activities *artifact design* and *evaluation* are not considered strictly disjoint. This leads to an

overall research method as shown in Figure 2.9, in which also the relationships to this thesis' chapters are outlined. The color-coding and symbols from Section 2.2 are used to distinguish between chapters that are concerned with method-related (red △) or nexus-related (blue ○) contributions, or the application thereof (green □).

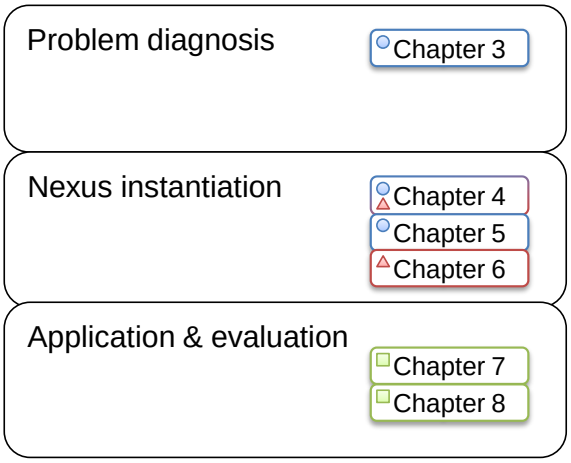


Figure 2.9: Overview on the thesis' research method

It is also the scope and the complexity of the research topic EA modeling that make an iterative approach (analyze, design, evaluate) nearly infeasible to pursue, at least as long as a single doctoral thesis is concerned. This can be explained with the strategic and long-term nature of EA management. While one can analyze the immediate outcomes of an EA-related project after execution, a continuous EA management function does not provide immediate effects. The resulting changes, e.g. in the interaction between different enterprise-level management functions have late effects. In line with Ross and Beath [RB06], we can expect the benefits to manifest after about five years. Our research building on existing design theories and application cases from the knowledge base in the field can deliver results in a shorter frame, making prescriptions for organization-specific EA modeling languages both better accessible and usable.

2.3.1 Problem diagnosis

During the problem diagnosis, we elicit a comprehensive understanding of the class of problems, onto which the development method applies. We further characterize the context domain that constrains the possible application fields. Therefore, we carry out the following three activities:

- In the activity **exploring the problem and context domain** we collect the problems that should be addressed by the development method and analyze existing design theories as well as existing cases in the field. Understanding the cases as *extracting case studies* in terms of van Aken [Ak04, page 232], we rewrite the case descriptions as documentations of pattern candidates. Based thereon, we apply the technique *enumeration of problems* as discussed by Takeda et al. in [Ta90].

- The activity **structuring the domains** establishes conceptual frameworks that help to categorize the concepts from the corresponding domains. These frameworks also provide a high level structure for the domains. According to Nunamaker et al. [NCP91, page 635] structuring principles support problem understanding. During the establishment of the conceptual framework, we apply relevant analytical or explanatory theories in the sense of Gregor [Gr06]. A key contribution to the structuring framework are the relationships between different contexts or different problems. We establish these based on unifying meta-conceptualization of the domains, i.e., in a set of unifying terms.
- In the activity **meta-conceptualizing the domains** we detail the structuring framework with meta-concepts that classify relevant objects from both problem and context domain. The meta-conceptualization provides a clear and consistent terminological basis to express problems as well as contexts and to communicate relevant statements. Thereto, we use a conceptual modeling language for describing the meta-concepts as well as a textual description, in order to homogenize the underlying, potentially conflicting terminologies.

The activities constituting the phase of problem diagnosis have no strict ordering, but are executed in a hermeneutic cycle. We read and interpret the literature prescriptions and cases against the background of our initial knowledge. In doing so, we formulate pattern candidates, where necessary, and extract relevant information to shape both problem and context domain. Therein, we apply inductive techniques to generalize the findings and get to an initial description of the corresponding domains. The intended outcome of **problem diagnosis** are a set of problem statements as well as a set of context statements structured in one or more conceptual frameworks. All statements are further grounded in a well-defined terminology and a conceptual modeling language.

2.3.2 Nexus instantiation

In the phase nexus instantiation, we execute four distinct activities, which derive from the sub-activities of theory building outline by Venable in [Ve06]:

- During the activity **eliciting method requirements** we take a users perspective on the development method and its underlying nexus instantiation. We analyzed the user's expectations with respect to the application against the background of ad-hoc methods in this field. Further, relevant theories and general requirements on methods as well as their underlying knowledge base are revisited to elicit requirements.
- In the activity **exploring solution models** we gather existing solutions described in the design theories as well as in the patterns and pattern candidates. These solution models populate the solution domain, which is further shaped by taking into account relevant kernel theories. Again, we apply the technique of the extracting case study [Ak04, page 232] to evolve the solution domain.
- In the activity **meta-conceptualizing the solution domain** we establish the basis for understanding the solution models and their corresponding assertions with respect to context and problem. The meta-conceptualization provides a unifying basis for all the different models, methods, and guidelines. Key elements in the meta-conceptualization are meta-languages that can be used to unify the different EA modeling languages without causing redundancies.

- During the activity **defining solution assertions** the solutions are embedded in the context of the nexus instantiation. We establish three distinct types of assertions: the **context and goal assertions** that link the solutions to the problems and contexts, the **solution assertions** making explicit relationships between solutions based on the meta-concepts, and the **general assertions** that reflect beneficial and intended characteristics of a general solution, e.g. requirements that any particular solution must fulfill.

The activities constituting this phase have no strict ordering. Again, we apply a hermeneutic cycle to foster the alignment between the exploration and meta-conceptualization. Further, external kernel theories are facilitated by a hermeneutic method that allows to re-read and revise already devised meta-concepts and assertions against the background of additional grounding theories.

2.3.3 Application and evaluation

During the final phase we apply and evaluate the development method. The application builds an expository instantiation of the method, whereas the evaluation targets the method from a more theoretic perspective. The nature of the research outcome, i.e., the building block-based development method, makes a theoretic evaluation necessary. We build our method on the prescriptions from other approaches whose validity cannot be assessed as part of our evaluation process. In addition, we cannot evaluate any of the diverse combinations of design theories that can be selected from the design theory nexus instantiation in practice. Therefore, we assume that the prescriptions from the state-of-the-art literature are correct and confine our evaluation to the development method and its employed techniques. This evaluation is approached from a twofold perspective. Firstly, we implement the development method and its underlying theory nexus instantiation into a toolset. Thereby, we evaluate the technical feasibility of the method. Secondly, we perform an argumentative evaluation, showing that the development method creates EA modeling languages that per design fulfill all general assertions and requirements. In the final step of this phase, we apply the development method in a practice case and document the steps taken as well as the created results. Along this case, we further show the innovative qualities of our development method and delineate which novel insights could be gained.

Developing a nexus-based information system generator

In this activity we develop a tool-set incorporating the development method, its underlying techniques, and the design theory nexus instantiation. Basic constituents of the tool are mechanisms for storing, searching, and accessing the different design theories that are interlinked in the nexus instantiation. The search mechanisms are of particular interest enabling the users of the tool to determine which design theories are applicable to their selected problems in their given contexts. The tool further supports the consistent integration of different design theories in order to develop a comprehensive solution. In the development of the tool we can adopt the system development method outlined by Nunamaker in [NCP91, pages 635–637]. The first step of this method is already fulfilled as we have developed a meta-conceptualization for the method and its corresponding design theory nexus instantiation. Therefore, the design activity can start with the development of a system architecture incorporating components for each of

the above functions. Subsequently, the requirements that the tool should fulfill can be derived from the users' expectations with respect to the development method itself. Afterwards, we derive a feasible system design and discuss critical aspects of its implementation.

Theoretic evaluation of assertions

In this activity we subject the development method and the design theory nexus instantiation to theoretic evaluation. For this evaluation on the one hand general research principles [He04, page 17] are of interest. On the other hand, this type of evaluation has to show that the assertions about the solution models as well as their interplay with the problems and contexts hold. The second type of evaluation is called according to Verschuren and Hartog [VH05, pages 739–751] a *backtracking evaluation* consisting of "plan", "process", and "product evaluation". We subject every assertion on the solution models and their interplay with contexts and problems to an argumentative evaluation process. Thereto the assertions are formalized taking the design theories and patterns used for their compilation as granted. Tracking back from a hypothetical design artifact through the steps of the development method, we consider the different techniques from both a pre-condition and post-condition perspective. We analyze which post-condition is expected to hold for any of the intermediary design artifacts in the development method and reflect on the necessary pre-conditions for the technique applied in the step. Depending on the level of formalization with respect to the technique, the conditions can be described textually or in mathematical and logical invariants. The trace back through the conditions forks in different places, creating a tree-like structure which roots in the assertion to be analyzed. Having explored all leaves of all such trees, it remains to be evaluated, whether the pre-conditions expected from the leaves hold. Thereto, we distinguish two different cases:

- The leaf is to be filled with a particular design theory depending on the selection of problems and contexts. In this case, we can reformulate the condition in terms of the design theory nexus instantiation's underlying consistent meta-conceptualization.
- The leaf represents a particular quality of the nexus instantiation itself. In turn, this quality is rewritten based on the meta-conceptualization as incorporated in the meta-languages complementing the nexus.

Having formulated the fundamental pre-conditions as invariants over the meta-conceptualization and its complementing meta-languages an inductive evaluation approach over the techniques of the methods used for compiling and administering the design theory nexus instantiation is taken. In particular, the analysis targets to show that given a nexus instantiation for which the invariants hold as well as a correct design theory, the combined nexus instantiation remains valid with respect to the invariants.

Supplied with information about possible deviations from the expected output gathered during the theory-driven artifact evaluation, the trace back can find the causes for design anomalies in inappropriate or incorrect steps during the development method or in its underlying design theory nexus instantiation. To do so, unwanted or unexpected properties of the design artifact are described using the meta-concepts. From this point, techniques and principles of implementation targeting the 'suspicious' meta-concepts are derived and their appropriate as well as correct application during the expository instantiation is analyzed along the testable

implementation hypotheses, retrospectively. If the deviations cannot be explained that way, the trace back continues to the solution model as well as the context and problem descriptions targeting the corresponding meta-concepts. These are analyzed along the complementing solution model hypotheses. This can lead to a situation where a single design theory bound to the nexus has to be put into the center of attention. At this point the necessary translation of terms from the original design theory to its representation in the nexus instantiation must be analyzed in detail, seeking for possible translation errors causing the deviation. If none such linguistic issues are discovered, the design theory itself has to be subjected to refutation. In particular, the contexts of the theory can thereby deserve additional investigation, especially, when the theory itself was extracted as a pattern or pattern candidate from an underlying case. As we cannot assume that the developer of the nexus instantiation is able to reproduce and revisit any particular context, the trace back is expected to yield at least one additional assertion, describing that the theory under consideration is not applicable in a particular context—the one, in which the anomaly has been detected.

Practical application

During this phase the development method and the underlying design theory nexus instantiation are applied in a practical setting to guide development processes for organization-specific and problem-specific EA modeling languages in the three activities **nexus-driven problem elicitation**, **nexus-driven artifact design**, and **nexus-driven evaluation**:

- During the activity **nexus-driven problem elicitation** we use the meta-conceptualization of the nexus instantiation's problem domain to elicit requirements for EA modeling languages. This elicitation can take place as a *developing case-study* in terms of van Aken (cf. [Ak04, page 232]) in an enterprise willing to establish an EA management. If this was not possible, one or more EA modeling languages of an enterprise can be analyzed ex-post, i.e., after the fact. Nevertheless, this 'reverse-engineering' is often aggravated by the fact that the EA modeling languages are, as we discuss in [Bu09a], created in an ad-hoc manner without preceding problem elicitation.
- In the activity **nexus-driven artifact design** we select the relevant design theories and enact the principles of implementation as provided by the design theories. At first a conceptual model for the organization-specific and problem-specific solution is created based on the meta-conceptualizations underlying the solution domain. This model is subsequently refined and adapted to the specific environment of the target enterprise. In case that a developing case study in terms of van Aken (cf. [Ak04, page 232]) can be executed, interactive elements in the principles of implementation should be executed in cooperation with stakeholders from the enterprise. If no enterprise willing to implement an EA management function is at hand, the necessary input for the method should as far as possible be reverse-engineered from design documents and the existing EA management function, respectively. Beside the implemented solution, this activity produces an additional outcome, namely a design rationale detailing the decisions and assumptions that led to the final design. This rationale may be of interest for the using company in a developing case-study, but is also helpful for evaluating the design process and its results.

- During the activity **nexus-driven artifact evaluation**, we evaluate the designed EA modeling languages in respect to their design artifact qualities. If no developing case-study (cf. van Aken [Ak04, page 232]) could be conducted, the design rationale becomes highly relevant for evaluation, as along this rationale stakeholders of the EA management function can revisit the design decisions and provide subjective statements on the fulfillment of the requirements. In line with the argumentation of Gehlert et al. [Ge09, page 449] the aforementioned qualities can be regarded as *outcome-oriented* evaluation targeting the utility of the design artifact.

Outcome-oriented evaluation in the sense of Gehlert et al. [Ge09, page 449] directly targets qualities of the actual design artifact, i.e., the result of applying the design theory. Gehlert et al. further argue for a theory-oriented evaluation of the design artifact in respect to its underlying design theory. During this evaluation it is analyzed whether an anomaly has arisen and the theory is 'suspect'. An *anomaly* in the aforementioned sense might be any kind of inconsistency occurring during the application of the theory's prescriptions, e.g. a modeling anomaly in the creation of a situated problem description based on the theory's meta-conceptualizations. An anomaly can also be an ambiguous or infeasible design step described in the principles of implementation.

2.4 Summary

In this chapter, we explored the research design underlying the thesis. In line with Becker et al. [Be03, page 5] we discussed the ontological, epistemological, and linguistic assumptions underlying our research, delineated the intended research outcomes, and outlined the research method taken. Table 2.2 summarizes the ontological and epistemological assumptions underlying this thesis and presents them along the framework that Becker and Niehaves outline in [BN07, page 202]. The set of adopted assumptions matches in large parts the *consensus-oriented interpretivist approach* exemplified by Becker and Niehaves. Two minor differences in respect to the ontological position and the methodology exist. The ontological position of this thesis is thereby 'less realist', i.e., accounts for the intellectual nature of many EA-relevant concepts. The methodological position of the thesis extends the basic methodology of the consensus-oriented interpretivist approach with the methodological 'superstructure' of a hermeneutic cycle. This is needed in the light of the literature-based research regarding the induction of design theories.

Epistemological question	Epistemological assumption		
Ontological aspect	Ontological realism	Ontological idealism	Kantianism
Epistemological aspect	Empistemological realism	Constructivism	
Concept of truth	Correspondence theory	Consensus theory	Semantic theory
Origin of cognition	Empiricism	Rationalism	Kantianism
Methodological aspect	Inductivism	Deductivism	Hermeneutic

Table 2.2: Epistemological and ontological assumptions of this thesis presented along the framework of Becker and Niehaves [BN07, page 202]

In the thesis we contribute to the field of developing organization-specific EA modeling languages and establish a development method based on proven practice solutions for developing EA modeling languages. Further, we implement the method in a corresponding tool-set that helps its users to develop their specific EA modeling languages supporting multi-viewpoint EA management. The practice-proven prescriptions are formulated as **design theories** and integrated in a corresponding **design theory nexus instantiation** in the sense of Pries-Heje and Baskerville [PHB08]. The nexus instantiation is one of the five research outcomes constituting and complementing the development method:

O1 **Meta-concepts** establish the basis for describing context, problem, and solution domains. In the problem and context domains, the meta-conceptualizations present themselves as defined terminologies and frameworks for unambiguously describing the situated EA problem. The meta-conceptualization for the solution domain goes beyond a terminology and a framework. On the one hand, the framework and its constituting concepts are complemented with more formal definitions that e.g. allow to operationalize relationships between solution models towards mathematical expressions. The meta-conceptualization for the solution domain on the other hand presents techniques and meta-languages for specifying an EA modeling language.

O2 **Design theory nexus instantiation** describes the design theories for constructing an organization-specific EA modeling language. Each design theory is presented as solution responding to a distinct EA modeling problem in a specific context. In this presentation the terminology established by the meta-concepts is used as a unified basis, while the meta-conceptualizations' framework is used to achieve a concise and consistent description. When it comes to the solutions presented in the design theories as conceptualization[3] of an EA-related concern, the techniques provided by the solution domain's meta-conceptualization are employed. Further, different solution models are related based on the corresponding meta-concepts.

O3 **Method for administering the nexus instantiation** is constituted of the five steps to instantiate a design theory nexus that Pries-Heje and Baskerville [PHB08, pages 737–738] describe. The method for developing delineates how design theories drawn from various sources are refined and linked in the nexus instantiation. Grounded in the meta-conceptualizations for the three domains, we describe how specific theories' prescriptions are turned into guidelines using the common terminology established in the conceptual basis. These descriptions entail specific indicators that can be used to identify incompletenesses in the conceptual basis that are processed by the maintenance method to evolve the meta-concepts.

O4 **Nexus-based development method** provides guidelines on how an EA modeling language for a situated EA problem can be constructed, or how an existing EA modeling language can be evolved and adapted. Grounding in the terminological basis established by the meta-concepts, the methods can be used for adapting the 'raw' solution models to the specific context. We further describe techniques that can be applied to integrate the design theories into a comprehensive and consistent modeling language. Further the

[3] The term *conceptualization* here actually refers to the conceptualization of an EA and not to a meta-conceptualization for EA modeling.

method describes guidelines on how conceptual relationships between the theories can be utilized to derive possible adaptation and evolution scenarios. Additional techniques are detailed that support the transformation from an old to a new EA modeling language.

O5 **Nexus-based information system generator** presents itself as a configurator for an EA modeling tool. This tool implements the techniques and meta-languages from the meta-conceptualization of the solution domain. EA modeling concerns from the problem domain are used in the tool to identify solution models during the application of the nexus instantiation, as mechanisms for developing organization-specific EA modeling languages.

	O1	O2	O3	O4	O5
Meta-languages	●			●	
Organized library of language building blocks		●			
Administration method			●		
Development method				●	
Technical tool-set	○	○	○	○	●

Table 2.3: Relationships between research outcomes and contribution of the thesis

These general outcomes relate to the main contributions of the thesis as described in Section 1.2. Table 2.4 gives some indications on the nature of the relationships: the symbol ● denotes a direct relationship between the research outcome and the corresponding contribution of the thesis. In contrast, the symbol ○ is used to indicate that the research outcome is beneficial for the according contribution.

In Section 2.3 we presented the research method that is applied to achieve aforementioned outcomes. For the particular purpose, we synthesized a specific research method from the general method of Pries-Heje and Baskerville [PHB08], Venable's method for theorizing in IS research [Ve06], and a method for pattern-based design theorizing presented by us in [BMS10k]. The synthesis results in a three step method that instantiates the design theory nexus for EA modeling languages based on case studies from literature, observed patterns for EA management, and existing design theories in the field.

But I've come today not to ask, but to offer. To offer you the truths that you so desperately sought...

Cigaratte smoking man, Redux II—X-files

CHAPTER 3

Analyzing the State-of-the-Art in EA Modeling

The first research question is of central importance for this thesis. Therein, we ask, whether it is beneficial or necessary to combine different EA approaches and integrate their EA modeling languages during the development of an organization-specific EA management function. This would not be the case, if a single language addressed all relevant aspects of EA management. Rephrasing the question in colloquial terms, we have to ask:

> Are there typical EA management tasks that can only be supported by one EA management approach, but not by another?

Above question has a comparative nature and in detail refers to two distinct types of comparisons, namely:

- comparisons between modeling languages from literature and modeling requirements as well as
- comparisons between different modeling languages from literature.

The second comparison is especially necessary for the practical EA management approaches. They do not present outputs of rigorously conducted design research, also alluding to the requirements and problems. As stated in Section 2.3, the state-of-the-art in EA modeling languages is analyzed with the method of hermeneutic text comprehension [Ga75]. While such a research method as discussed in Section 2.3 well suits the subject of research, it is very inconvenient to report the research outputs in a similar fashion. Therefore, this chapter is organized in a linear way, firstly introducing the relevant fundamental concepts of modeling in Section 3.1. Subsequently, we perform a literature review building on these fundamentals. Following the guidelines for reviews promoted by Webster and Watson [WW02], we

- make the review objectives explicit,
- discuss the scope and limits of the literature included in the review, and
- provide a clear structure for the review.

Research question 2 provides the objective of the review. The scope and limits of the review are discussed in the light of the ongoing and increasing interest in the topic of EA management (see also Chapter 1). The amount of literature in this area proliferates (cf. survey of Langenberg and Wegmann in [LW05] or Mykhshchuk et al. in [My11]). An accompanying change in the used terminology from "information systems architecture" (cf. Zachman in [Za87]) via "business IT alignment" (cf. Luftman in [Lu03]) to "EA management" aggravates the identification of relevant publications. In addition, literature databases, e.g. the web of science[1], the ACM digital library[2], or IEEE Explore[3], provide only minor help, as EA management-related research results are often published as practitioners' books or presented on workshops (cf. Trends in Enterprise Architecture Management Research—TEAR [EDO06]). Therefore, the results are not included in these databases, which typically focus on journal publications. We identified literature relevant for the subsequent review

- based on a selection of important research groups in the state-of-the-art analyses of Aier et al. in [ARW08b] as well as Schelp and Winter in [SW09],
- by searching the DBLP[4] and the websites of the authors for further publications,
- by backtracking the citations of the publications identified in the first two steps, and
- by removing research groups, whose contributions are not available in English.

In [BS11] we identified 22 research groups and individual authors using above approach. For this thesis, we further analyzed these approaches with respect to the focus (method or language) as well as with respect to their proliferation, reducing the number of relevant approaches to 14. These approaches cover a broad range of possible sources, ranging from academic literature over practitioners' works originating in consultancies to the de-facto standard of TOGAF [Th09a]. In the following a framework presented by Fettke [Fe06, page 259] is used to characterize the subsequent review as shown in Table 3.1. We take a practitioner's perspective committing to the intended audience of this thesis' overall research result. The review in general is structured historically, giving an overview on existing development paths.

Type		natural language				mathematic-statistical	
Focus		research results		research method		theory	experience
Target	Formulation	not explicit				explicit	
	Content	integration		criticism		central topics	
Perspective		neutral				position	
Literature	Selection	not explicit				explicit	
	Extensiveness	foundations		representative		selective	complete
Structure		historical		thematically		methodical	
Target group		common public		practitioners		common researcher	specialized researcher
Future research		not explicit				explicit	

Table 3.1: Characterization of our literature review according to Fettke in [Fe06, page 259]

Webster and Watson [WW02, page xiv] as well as Bem [Be95, page 174] highlight the importance of a "clear structure for the review". We establish a "coherent conceptual structuring of the topic" according to Bem in Section 3.2 in terms of an analysis framework for the do-

[1] See `http://www.webofscience.com`, last accessed 04-05-2011.
[2] See `http://portal.acm.org`, last accessed 04-05-2011.
[3] See `http://ieeexplore.ieee.org`, last accessed 04-05-2011.
[4] See `http://www.informatik.uni-trier.de/~ley/db/`, last accessed 04-05-2011.

main of EA modeling languages. In Section 3.3 the EA management approaches are revisited and characterized in the dimensions of the analysis framework. In Section 3.4 we summarize the findings and answer research question 1, but also highlight specific characteristics and recurring challenges in the different EA modeling languages.

3.1 Fundamental concepts of modeling and languages

In the analysis of the state-of-the-art, we take different perspectives on the EA modeling languages to get a comprehensive image of the corresponding contributions. These perspectives are motivated by the nature of EA descriptions as artifacts taking different roles simultaneously. Firstly, EA descriptions are *models* of current or planned architectures, such that the understanding of *general model theory* in line with Stachowiak [St73] applies, see Section 3.1.1. Secondly, EA descriptions are means of communication. They lift subject-dependent conception to an inter-subjectively communicable level for a backing *linguistic community* in terms of Kamlah and Lorenzen [KL67], see Section 3.1.2. Thirdly, an EA description itself can take different perspectives on the subject EA. Different users of the description have different *areas-of-interest* in the overall architecture. The IEEE Standard 1471-2000 provides a meta-model to discuss on the relationships between the different perspectives in a description as well as their corresponding users, see Section 3.1.3. Finally, EA descriptions are vehicles to facilitate EA management, i.e., to the design of the EA. According to this perspective, we discuss how typical constituents shaping a *design space* in terms of Simon [Si96], are present in the EA description and its underlying language, see Section 3.1.4.

3.1.1 General model theory of Stachowiak

Central to the subsequent considerations is the notion of the *(architectural) model*. This term is widely used in many disciplines as well as in various EA management approaches in literature, most notably without being complemented with a definition. To avoid terminological confusion, we define the term in line with Stachowiak [St73] as follows.

Definition: Model
A model is a purposeful abstraction of a defined part of reality at a certain point in time for distinct users.

According to Stachowiak, the following three characteristics are essential, whenever model creation and utilization are discussed:

- **representation**: A model is always a model of something, i.e., represents certain natural objects or models (*universe*).
- **simplification**: A model is always restricted to certain parts of the reality, i.e., simplifies the represented object in respect to certain qualities.
- **pragmatics**: A model is always developed for a certain use, i.e., for a dedicated utilization context and using party.

Under the premise of these characteristics, two architectural models in one enterprise can differ in three distinct ways. Firstly, their underlying universes of discourse can be different

parts of the architecture. Secondly, the models' simplifications can be different, i.e., they can contain representations of the same parts of the architecture, but emphasize different aspects thereof. Finally, the architectural models can target different audiences of model users, while representing similar elements from the universe of discourse (pragmatics). Along an example from conventional architecture, the three ways of distinction read as follows.

Example 3.1: Representationally different viewpoints. Two floorplans of a building, representing first and second floor, respectively, are 'representationally different'. The corresponding parts of the universe of discourse, i.e., of the building's architecture, are completely distinct. Two floorplans of the first floor, one with and one without information on electrical wiring, are different in respect to their underlying simplifications. They target 'intersecting' parts of the universe of discourse, but decide to display or hide wiring information, respectively. A set of plans as well as drawings on the one hand, and a 3D rendering on the other hand both represent the entire building, but differ in respect to their pragmatics. While the set of plans targets users concerned with erecting the building, the 3D rendering may be used to inform (and persuade) prospective residents of the building. This means that the set of plans and the 3D rendering are different in respect to their pragmatics, although they may additionally differ in respect to the employed simplification, as e.g. the omission of information on the wall thickness.

Above example emphasizes a subtlety of comparing two architectural models in respect to their representation and simplification characteristics:

- *representationally different* means that different objects are modeled, while
- different *in respect to the simplifications* means that the same objects are abstracted differently, i.e., different *properties* thereof are described.

This seems to reduce may reduce the question, whether two models differ representationally or in respect to the simplification taken, to a 'logomachy' on what we should call an *object* or a *property*. We will address the underlying conceptual question on the object-nature in the universe of discourse in Chapter 5.

3.1.2 Linguistic perspective on modeling

Stachowiak's perspective on model is constructivist (cf. discussion in Section 2.1), that means he strongly emphasizes on the role of the model creator. This well aligns with the kantianist and interpretivist perspective taken in this work, according to which the subject-dependency of the abstraction plays an important role. One should not misunderstand the term *abstraction* as a mathematical representation function in a strict sense. Abstraction depends on the intended purpose as well as on the subject of the modeler. We can say that the natural

objects (*noumena*) which the model seeks to represent are by themselves unknown, such that a model is restricted to representing their cognitive pendants, the *phenomena*. These phenomena themselves are modeled subjectively. This understanding opposes the representation-centric understanding of a model as "triple of object-system, model-system and homomorphism" (cf. Ferstl and Sinz [FS98, pages 18–20]). We do not regard this as drawback in this work, because this understanding would, as described by Schütte and Rotthowe in [SR98], not align with the ontological position of kantianism and the epistemological position of interpretivism.

An interpretivist perspective nevertheless should not be understood as justification of arbitrariness in modeling. Albeit the subject-dependency of their creation, a 'true' model is not arbitrary. In line with a consensus theory of truth, the 'truthfulness' of a model is linked to model utilization. It is sensible to speak of a 'true model', if the model creators and the group of its aspired users consent on its truthfulness. Model pragmatics [St73] is therefore one key determinant for truthfulness, but also the linguistic perspective is important. The users and the creators of that specific model must use a common system of *speech acts* to be able to reason and agree on the truthfulness of the model, more precisely on the represented phenomena of common interest. Kamlah and Lorenzen establish in [KL67, page 31] at terminology to discuss speech-acts incorporated in the model and exchanged between model creator and model user. Central to this terminology are the notions of *nominator* and *predicator* as follows:

- A *nominator* is a name assigned to a (real world) object, which is conversely represented by the nominator.
- A *predicator* is a name that generalizes similar (real world) objects.

Both nominators and prediators are noumena, i.e., are expressed via speech acts[5], and denote (nominate) or generalize (predicate) other noumena (see Figure 3.1).

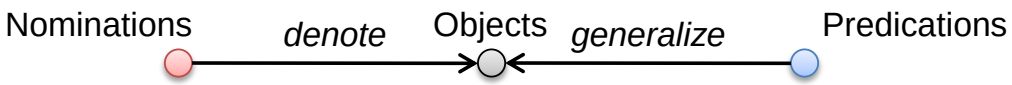


Figure 3.1: Relationship between nominations, predications, and the universe of discourse

Model creators and users which can agree on the nomination and predication for their specific universe of discourse form a **linguistic community** [KL67]. Such community is necessary to be able to agree on the truthfulness of the model by developing consensus on a statement. From this and in accordance to Kamlah and Lorenzen, we can derive the following definition of the term linguistic community as relevant in the context of modeling.

Definition: Linguistic community
A linguistic community is a group of people that in respect to a certain universe of discourse agree on assigning the same linguistic terms to the same objects of discourse.

[5] Kamlah and Lorenzen discuss in [KL67, page 63] different types of nominating and predicating acts, e.g. "dancing steps". For the context of this thesis, the description of EAs, predicators and nominators of this general type are nevertheless only of limited interest.

Successful modeling is inevitably connected to establishing a linguistic community of model creators and model users. This task should preferably be supported by the language used for modeling. In [PVSH05] Proper et al. discuss on the subject-dependency of models. The *Framework of Information System Concepts* (FRISCO) tetrahedron of Falkenberg et al. [Fa98] illustrates the interplay between *objective* and *subjective* elements of world comprehension via speech-acts or models. Figure 3.2 adapts the tetrahedron to the terminology of Kamlah and Lorenzen [KL67] further introducing the *conception* as mental representation of the universe of discourse for the specific viewer. The speech-acts constituting the nomination make explicit this conception.

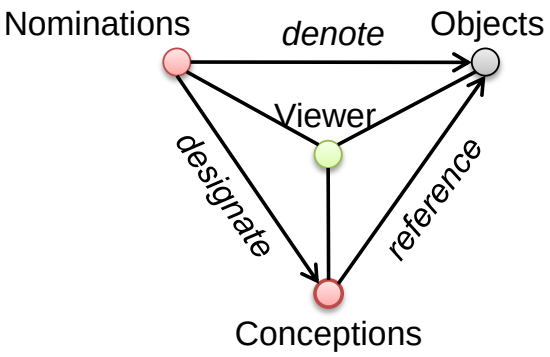


Figure 3.2: Relationship between nominations, conceptions, universe of discourse, and viewer

In the same way the objects in the universe are identified with the noumena of the kantianist perspective, the **elements** of the conception can be identified with the phenomena of the kantianist perspective. Such identification nevertheless neglects that phenomena can be generalized. Example 3.2 illustrates this fact.

Example 3.2: Nominators and predicators. Seeing a car, I can name it "my car", whereby I designate the conception of being my car, or predicate it as "AUDI A3", whereby I express the car being generalizable to a certain type.

Both "being my car" and "being an AUDI A3" are phenomena, while only the former can be designated with a nominator. The latter reflects a generalization and is hence designated by a predicator. Each such generalization, subsequently called a **concept**, represents a phenomenon depending on the viewer's interpretation. Figure 3.3 complements the tetrahedron of conception with the tetrahedron of **conceptualization** to what Aveiro et al. call in [AST10b, page 152] the "ontological trapezium".

A conceptualization describes the concepts, to which a viewer (creator or user) of a model gen-

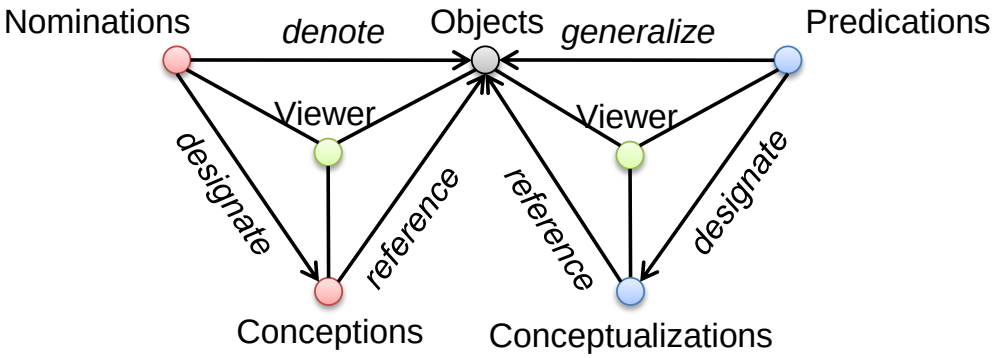


Figure 3.3: Combining the tetrahedra of conception and conceptualization

eralizes the universe of discourse. The members of a linguistic community have to agree on the predicators used to designate the concepts, which they are communicating about. This means that the predicators for the specific universe of discourse have to be 'frozen', or as Kamlah and Lorenzen describe in [KL67, page 70] have to be "standardized predicators", so called *termini*. A terminology should be provided to complement the EA modeling language supplied with an EA management approach, and such that the language supports the establishment of a linguistic community.

3.1.3 Architecture descriptions and their users according to the IEEE Standard 1471

Architectural descriptions usually do not only consist of a single model, but employ multiple models. Therefore, we have to understand how different models relate in an architectural description. In conventional architecture the floorplan is an architectural model, i.e., a purposeful abstraction of the building's architecture. Speaking more precisely, the floor plan represents a part of the architecture of the overall building. What at first sight seems to be linguistic sophistry indicates towards a central and crucial characteristics of model utilization. A user has to know the specific part of reality reflected in the model, i.e., must know the **viewpoint** in order to understand an architectural description. The viewpoint describes the perspective on the architecture that underlies the model. For a floor plan additional information must be supplied to understand the plan in context. A reader must for know that the first floor is depicted, that the plan is drawn with north being top, and that the basic raster points span a half-meter gitter. Such information is often conveyed within the plan, e.g. by a symbol indicating the orientation, but is not part of the model. It is *(meta-)information* on the plan's underlying viewpoint.

The *Institute of Electrical and Electronics Engineers* (IEEE) describes the concept of the viewpoint in the IEEE Standard 1471-2000—*Recommended Practice for Architecture Description of Software-Intensive Systems*[6] [IE00]. This standard conveys a conceptual framework for

[6]The standard was later adopted with minor additions by the ISO as ISO 42010:2007 [In07]. We will come back to the ISO standard later, but for our considerations here, the IEEE standard in its adaptation by Wittenburg in [Wi07] is a more consistent choice.

understanding architecture documentations consisting of architectural models in relationship with their underlying viewpoints. The standard further accounts for the fact that neither a viewpoint nor a corresponding architectural model are 'l'art pour l'art'. For viewpoints, pragmatics is supplied by a **stakeholder** that is interested in the architectural information conveyed in the viewpoint.

> **Definition: Stakeholder**
> A stakeholder is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.

The standard further introduces the concept of a stakeholder's **concern** as follows:

> **Definition: Concern**
> Concerns are those interests [in a system] which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

The standard embeds the concepts and their relationships into a meta-model to display the conceptual framework underlying the description of a system's architecture. Figure 3.4 shows this meta-model in a slightly modified version as put forward by Wittenburg in [Wi07]. This version encompasses the concept of the **modeling language**, which is not part of the original meta-model, and provides clarification with respect to the multiplicities.

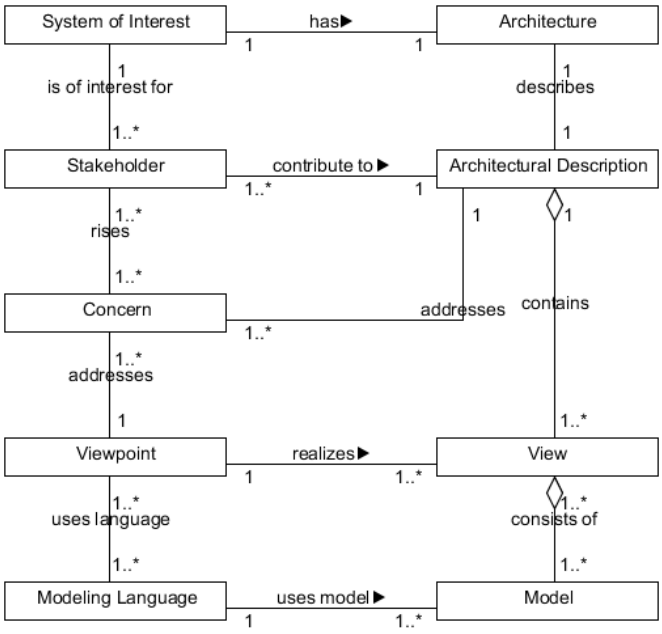


Figure 3.4: Adapted meta-model of the IEEE Standard 1471:2000 in the version of Wittenburg [Wi07]

In its conceptual framework for architectural descriptions, the IEEE Standard 1471:2000 neglects the topic of linguistic communities. For the standard's original context of systems engineering, this is not a critical omission. The prospective stakeholders in this field can rely on a well-established terminology. The standard hence assumes that the model users agree on the meaning of the modeled elements. That such assumption might be overly optimistic can be exemplified once again along the floor plan from Section 1.1. The following example reminds to keep the linguistic perspective in mind, when talking about models in general and architectural descriptions in special.

Example 3.3: Linguistic communities in conventional architecture.
If the plan had been added the indication "first floor", even an architect would have had difficulties to know, which floor is actually meant. This may be ascribed to the differences in floor counting between the British and the American method of counting. The first floor according to American counting would be the ground floor, while the first floor according to British counting would be the second floor in American counting.

3.1.4 Descriptions as vehicles of design

Simon promotes in [Si96] a formal understanding of the activity of design, based on the central notion of *means-end*-relationships. Any design environment sets into place a specific *means-end rationality*. Such rationalities are domain-inherent characteristics and relationships ("natural laws" cf. Simon [Si96, page 3]) that connect dedicated means to corresponding ends. This conversely means that a designer with a specific end (goal) in mind searches for means by which the design will achieve this end. In line with Aier et al. [Ai08b] we understand the management of the EA as a design activity aiming at design goals under the natural laws pertaining to the socio-technical system of the enterprise as well as artificial design restrictions imposed by governance structures. We identify four core constituents that occur in an EA-related design proposition, namely:

- *command variables* describing architectural properties that may be changed by design activities,
- *fixed variables* describing architectural and environmental properties that cannot be changed by design activities,
- *constraints* describing restrictions that pertain to the architecture design space, and a
- *utility function* describing a technique for evaluating a designed architecture in respect to the utility for its stakeholders.

The enterprise is a complex management subject, such that a management approach has to select a relevant part of the overall architecture. In the sense of the IEEE Standard 1471:2000, relevance grounds in the fact that a stakeholder is interested in the corresponding

part of the architecture. The potential stakeholders, as named by current EA management approaches, are diverse, see [Be09], and no common list of EA stakeholders has yet emerged. Therefore, a framework for analyzing the state-of-the-art in EA modeling must draw from alternative sources for structuring the domain. In [Bu10b] we used the design understanding of Simon [Si96] to distinguish the following aspects of modeling EAs:

- the **current** make-up of the EA (command variables),
- the **strategies & projects** affecting and changing the EA (means),
- the **visions & goals** describing a target state of the EA (ends),
- the **principles & standards** guiding the evolution of the EA (constraints), and
- the **questions & KPIs** measuring and evaluating an EA state (utility function).

We further give an overview about the relationships between these concepts, see Figure 3.5. All enterprise architects use **conceptions** and **conceptualizations** of the enterprise to perform their design tasks. Such *mental models* cover a specific area-of-interest in the enterprise from the perspective of a specific linguistic community. The EA stakeholders, whose problems and concerns are to be addressed by means of EA management, have to share the same terminology in order to communicate on design-related aspects. This need leads to more formal conceptions and conceptualizations of the enterprise or, more precisely, parts thereof. The conceptualizations are, as described by Buckl et al. in [BKS10], reflected via conceptual models, so-called **information models** that underly corresponding EA modeling languages. Accordingly, we distinguish in Figure 3.5 between different levels of *operationalization* with respect to the relevant EA design conceptions and models. Any conception can either be backed by a conceptualization or be based on an agreed conceptualization reflected in a an explicit information model.

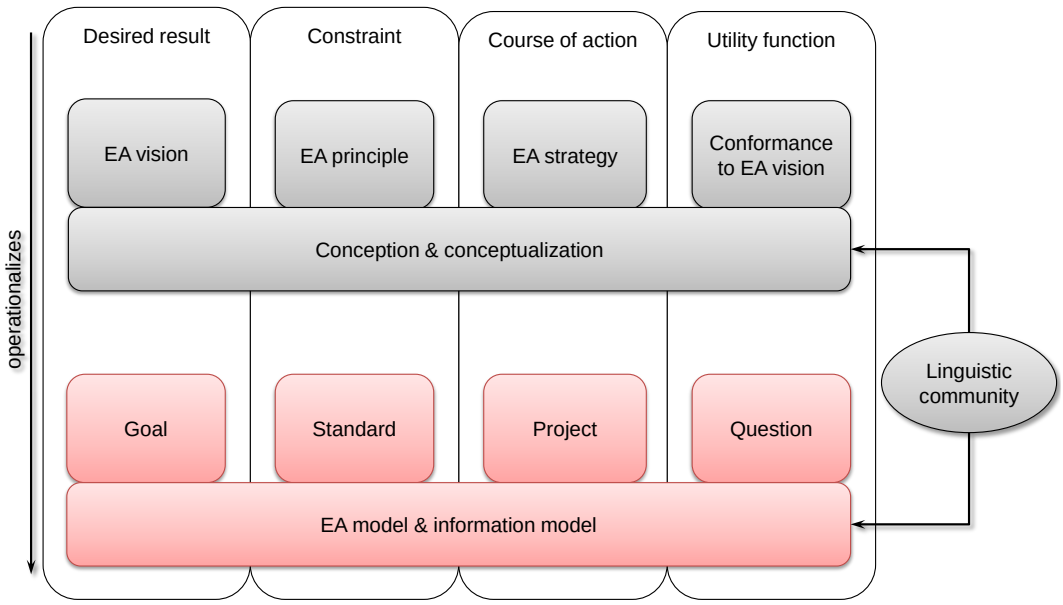


Figure 3.5: Conceptual framework of EA design conceptions (cf. Buckl et al. [Bu10b])

A final relevant dimension of EA modeling is motivated by Dietz in [Di06a, pages 8–12] as well as by Winter and Fischer in [WF06]. They distinguish between a *blackbox*-perspective and a *whitebox*-perspective on an EA or parts thereof. The blackbox-perspective exhibits a function-oriented viewpoint on the subject and provides a reduction to what Dietz calls "the essence" (cf. Dietz [Di06a, pages 8–12]) of the enterprise. Conversely, the whitebox-perspective reveals details of the implementation of the enterprise and details how specific functions are actually provided. An EA modeling language aiming to provide comprehensive support for EA management has to accommodate for all the aforementioned aspects of or pertaining to an EA.

3.2 Framework for analyzing EA modeling approaches

Our analysis of the state-of-the-art in EA modeling builds on an analysis framework, which is based on the fundamentals described above and which mirrors the diversity of the management subject. This is necessary to understand the approaches with respect to their EA modeling languages. EA frameworks, e.g. the one of Winter and Fischer [WF06], provide structuring principles for the subject EA. In the following we use the framework of Matthes et al. [Ma08] as it incorporates our design conceptions [Bu10b]. Further, the framework supplies a distinction between a blackbox-perspective and a whitebox-perspective. Matthes et al. distinguish between the architectural layers BUSINESS & ORGANIZATION, APPLICATION & INFORMATION, and INFRASTRUCTURE & DATA. Further, Matthes et al. apply cross-cutting perspectives as PROJECTS or GOALS. These cross-cutting aspects apply on the different architectural layers and are mirrored in the dimensions of distinction summarized in Table 3.2.

BLACK-BOX PERSPECTIVE	business & organization	application & information	infrastructure & data
WHITE-BOX PERSPECTIVE	business & organization	application & information	infrastructure & data
STRATEGIES & PROJECTS	business & organization	application & information	infrastructure & data
VISIONS & GOALS	business & organization	application & information	infrastructure & data
PRINCIPLES & STANDARDS	business & organization	application & information	infrastructure & data
QUESTIONS & KPIS	business & organization	application & information	infrastructure & data
CONFIGURE & ADAPT	initially	evolutionary	

Table 3.2: Language classification for an EA modeling technique

In the following, we detail each of the dimensions of distinction and the corresponding characteristics.

Analyzing blackbox coverage

An organization is designed to support the corporate objective, i.e., to deliver business capabilities to its customers. This yields a function-oriented perspective on the enterprise and its architecture, which is reflected in the notion of the blackbox perspective. The functions provided by the different architectural layers are modeled from an outsider perspective as blackboxes:

- BUSINESS & ORGANIZATION layer: the blackbox perspective on this layer describes a functional view on the business processes and organizational unit. They are thereby understood via concepts as BUSINESS CONTRACTS and BUSINESS CAPABILITIES as well as ORGANIZATIONAL CAPABILITIES.
- APPLICATION & INFORMATION layer: the blackbox perspective on this layer describes the application landscape from a functional point of view in terms of the provided BUSINESS SERVICES as well as their associated SERVICE-LEVEL AGREEMENTS and OPERATING-LEVEL AGREEMENTS.
- INFRASTRUCTURE & DATA layer: the blackbox perspective on this layer describes a functional view on TECHNICAL SERVICE PROVISION, e.g. network transmission or secure data storage, i.e., via concepts usually prevalent in a virtualization perspective on hardware.

Analyzing whitebox coverage

The whitebox perspective details on the actual architecture elements, which implement the supported functionalities and provides the decomposition of the enterprise on the different layers in a narrower sense:

- BUSINESS & ORGANIZATION layer: the whitebox perspective on this layer describes the BUSINESS PROCESSES and BUSINESS FUNCTIONS, but also the ORGANIZATIONAL UNITS that shape the support for business capability provision.
- APPLICATION & INFORMATION layer: the whitebox perspective on this layer describes the BUSINESS APPLICATIONS and INTERFACES, but can also account for the applications' COMPONENTS needed to provide business support.
- INFRASTRUCTURE & DATA layer: the whitebox perspective on this layer is concerned with the TECHNICAL DEVICES of an IT infrastructure, e.g. computing hardware or network devices.

Analyzing strategies & projects coverage

PROJECTS are the *means* of organizational change which in line with Murer et al. [MWF08] entails a broad understanding of the term **project**. It is not confined to activities with a short-term nature, but also to ongoing activities, as performed for example in maintenance contexts. On a higher level of abstraction, STRATEGIES describe the transformation and sketch how the parts of the enterprise are intended to change:

- BUSINESS & ORGANIZATION layer: strategies and projects on this layer target restructuring of the organization and the business processes, but also of the enterprise's product portfolio or market positioning.
- APPLICATION & INFORMATION layer: projects on this layer target changes to and maintenance of the application landscape, most particularly software development as well as customization projects. Strategies delineate the overall direction of IT transformation, e.g. heading for outsourcing of business applications.
- INFRASTRUCTURE & DATA layer: projects on this layer target infrastructure changes as hardware replacements, virtualization projects, or data migrations. Infrastructure-related strategies define infrastructural road maps, as e.g. vendor-strategies.

Analyzing visions & goals coverage

The GOALS describe the intended outcomes of EA transformations in a concrete and measurable way. On a more abstract level VISIONS frame the space of intended outcomes, i.e., delineate the intended state of the EA. Goals and visions apply on the different layers of the architecture, describing different kinds of intended states as follows:

- BUSINESS & ORGANIZATION layer: goals on this layer state intended qualities of the business processes, e.g. turn-over, and capabilities as well organizational qualities, e.g. number of employees. In contrast vision describe perspective qualities, e.g. downsized organization.
- APPLICATION & INFORMATION layer: visions on this layer, for instance, state intended upper bounds for the complexity of the application landscape, whereas goals define an intended availability for business applications and the thereby provided business support and services.
- INFRASTRUCTURE & DATA layer: visions on this layer can be prescriptions on infrastructure complexity, whereas goals can target network latency as well as hardware and infrastructure service availability.

Analyzing principles & standards coverage

PRINCIPLES are design constraints that are imposed onto the projects. They limit the design space for the EA and provide 'guiding rails' for the EA transformation. Standards describe the 'do's' and 'don'ts' on the level of affected architecture elements on:

- BUSINESS & ORGANIZATION layer: principles on this layer delineate the standard business processes that the organization seeks to apply in providing standard or commodity business capabilities.
- APPLICATION & INFORMATION layer: principles on this layer make prescriptions on application architectures to be used as well as interface standards to be employed.
- INFRASTRUCTURE & DATA layer: principles on this layer restrict the infrastructure technologies that may be used in the organization and impose constraints on the employed hardware.

Analyzing questions & metrics coverage

In [BMS10b, BMS10i] we apply the *goal-question-metric* of Basili et al. [BCR94] to measure goal achievement in EA management. Thereto, we introduce the concept of QUESTION, which links the relevant METRICS for goal attainment to the architectural elements [LS07, LS08] on the different architectural layers:

- BUSINESS & ORGANIZATION layer: questions on this layer target metrics for throughput, availability, and latency of business processes and thereby provided capabilities.
- APPLICATION & INFORMATION layer: questions on this layer target metrics for the structural complexity of the application landscape as well as for throughput, availability, and latency of the provided service.
- INFRASTRUCTURE & DATA layer: questions on this layer target metrics for infrastructure complexity as well as for hardware and services availability or utilization.

Analyzing configuration & adaptation coverage

The final dimension CONFIGURE & ADAPT is not derived from Simon's perspective, but relates to research question 1. In line with the argumentation of the IEEE standard any EA model is pertinent to the differing concerns as raised by the according stakeholders. These stakeholders can differ from organization to organization and can also change over time. Therefore, the EA modeling language has to be configured and adapted to the their corresponding concerns and EA management-related problems. We distinguish different degrees of configuration and adaptation support as follows:

- approaches providing no (explicit) mechanisms for configuration. Such approaches often do not regard EA modeling languages as organization-specific.
- approaches providing mechanisms to CONFIGURE the modeling language. These mechanisms usually allow to devise an initial organization-specific language, whose scope and reach is confined to the concerns of the using stakeholders, linked to the specific management goals, and mapped to the organization's terminology.
- approaches providing mechanisms of mature and ADAPT a once defined modeling language. These mechanisms describe how after the initial configuration, the scope and reach can be adapted, e.g. by adding further concerns or supporting additional management goals.

3.3 Revisiting EA modeling in prominent EA management approaches

The prominent approaches to EA management, as proposed in literature originate from widely differing backgrounds and target different aspects of the enterprise and its architecture. In the following, 14 approaches are analyzed in more detail using our analysis framework. Each of these approaches is preceded by a short fact sheet as shown in Table 3.3.

EA management or EA management-related approach
Name of approach:
Issuing organization:
Tool support:
Period of activity:
Publications:
Inner organization:

Table 3.3: Fact sheet for EA management approaches

The fact sheet provides general information about an approach, as the NAME, the ISSUING ORGANIZATION, the available TOOL SUPPORT, the group's PERIOD OF ACTIVITY, and the corresponding list of analyzed PUBLICATIONS. The INNER ORGANIZATION describes how the approach is structured:

- as one comprehensive MONOLITH without apparent inner structurization,
- with an EXPLICIT ORGANIZATION, in which the components establish explicit links to each other, or
- with an IMPLICIT ORGANIZATION, in which the components are grounded in a unified and linking terminology.

3.3.1 The Zachman Framework

EA management or	**EA management-related approach**
Name of approach:	Zachman Framework
Issuing organization:	Zachman Institute
Tool support:	none
Period of activity:	since 1987
Publications:	[Za87], [SZ92]
Inner organization:	monolith

In [Za87] Zachman developed a *framework for information systems architecture*, whose scope has ever since broadened to cover the architecture of the entire enterprise, becoming the perhaps most well-known framework for EA—the Zachman Framework. In its most recent version[7] the framework presents five modeling layers and six dimensions mirrored by corresponding interrogative pronouns, leading to the core question about the EA:

> Who does what in which way (how), when and where? Why does he do it?

The modeling layers are *scope*, *business*, *logical systems*, *technical systems*, and *detailed representations*, whereby the latter are according to Zachman not in the scope of EA management. On these different layers, the central questions of *what*, *how*, *where*, *who*, *when*, and *why*

[7] An overview on the framework is available online at `http://www.zifa.com/framework.pdf`. The recent version was accessed on 04-05-2011.

apply. Figure 3.6[8] outlines the structure of the Zachman Framework. The core question is answered on each of the layers with increasing level of detail, i.e., the conceptualizations used in answering the question become increasingly fine-grained.

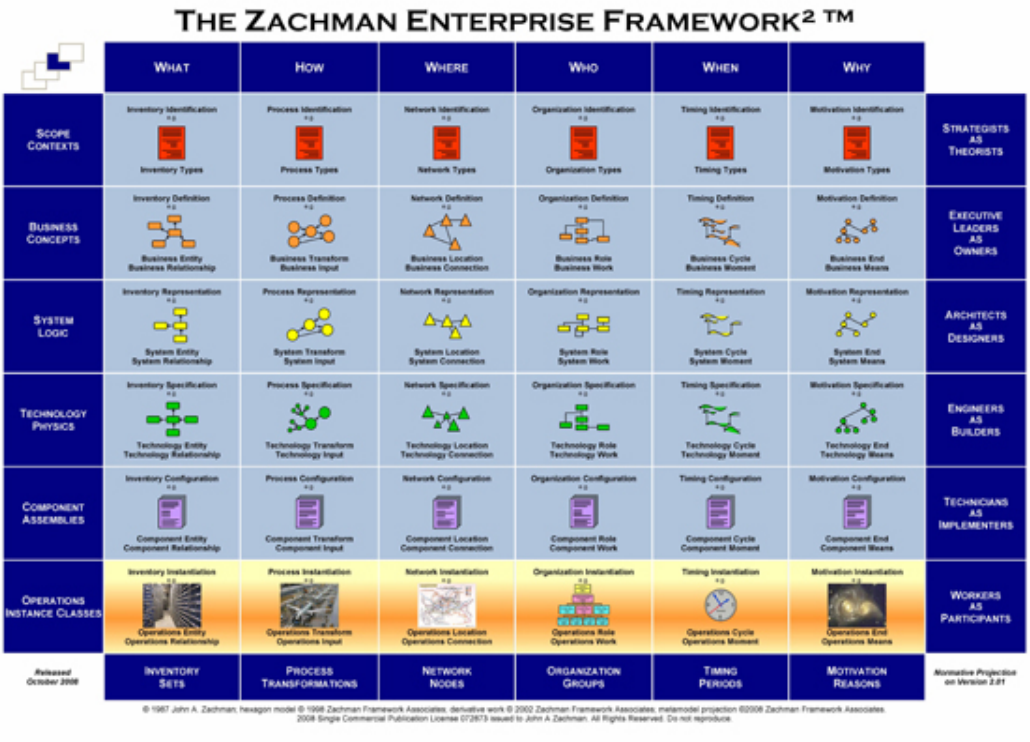


Figure 3.6: Two dimensional schema behind the Zachman framework

Central conceptualization of the Zachman framework with respect to the language used for describing architectures is the *thing-relationship-thing* paradigm. This conceptualization is adapted to the specific perspective taken in respect to the selected dimension to "input-output-input" in the *how*-perspective, "entity-relationship-entity" in the *what*-perspective, and "node-line-node" in the *where*-perspective (cf. Zachman [Za87, page 283]). In the initial work, the application of the paradigm on the remaining dimensions is only briefly sketched in an appendix providing concretizations as "organization-reporting-organization" for the *who*-perspective, "event-cycle-event" for the *when*-perspective, and "objective-precedent-objective" for the *why*-perspective (see [Za87, page 292]). Zachman further emphasizes that each of these perspectives is unique, i.e., that the corresponding descriptions are different even though they may pertain to the same object and therefore are inextricably related to one another. Another type of relationship between different perspectives exists regarding the modeling layer (scope) of the description. These relationships are further detailed by Sowa and Zachman in [SZ92, pages 592,603–605], explaining that a higher-layer model should not be derivable from the information contained in the according lower-layer models. Sowa and Zachman further discuss the recursive application of the framework in an enterprise context, delineating that beyond an enterprise-wide level, the framework also applies for the level of individual *products* (information systems). These abstract prescriptions are complemented with a crit-

[8] See `http://zachmanframeworkassociates.com/index.php/`, last accessed 04-05-2011.

ical distinction that emphasizes the design nature of the Zachman Framework as reflected by two *versions* that a framework user should create, namely an *as-is* and a *to-be* version of the architecture. Complementing the abstract paradigm "thing-relationship-thing", Sowa and Zachman introduce a graphical notation in [SZ92, page 607], the so-called "conceptual graph" that in conjunction with quantors from predicate calculus is used to describe concrete situations from the paradigm's perspective. In this vein, the conceptual graphs complement the *detailed cell metamodel* given in the same work [SZ92, pages 594–595], which concretizes the paradigm on business, information systems, and technology layer taking the how, what, and where perspective simultaneously. For each layer this leads to an integrated white-box meta-model introducing concepts as *business entity*, *business process*, and *business location* as well as their corresponding counterparts on the other layers. In line with the uniqueness requirement regarding the models on the different layers, the integrated meta-models as represented in their types and relationships are structurally equivalent, whereas no one-to-one mapping between the corresponding instantiations can be expected to exist. According to the explanations complementing the meta-model, all concepts contained therein may be subject to transformations during architecting, although concrete language facilities supporting the description of transformational activities are not supplied by the framework. This leads to a limited coverage of the dimensions (cf. Section 3.2) by the Zachman Framework as shown in Table 3.4, whereas many of the not fulfilled requirements are discussed there in some detail but not complemented with supporting language elements.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.4: Language classification for the Zachman framework

3.3.2 Architecture of Integrated Information Systems (ARIS)

The ARIS approach is a framework for holistic modeling of business information systems, targeting the development of such systems from a process-based perspective (cf. Scheer [Sc01, Sc02]). This in particular is mirrored by the overall method of the ARIS approach being very close to a 'classic' software development consisting of the sequence *requirements elicitation*, *design specification*, and *implementation description*. The waterfall-like method is nevertheless not executed once, but applied on the different views[9] that pertain to a business information system.

[9] In line with the terminology used in this report, the *views* would correctly be alluded to as **viewpoints**.

EA management or EA management-related approach	
Name of approach:	Architecture of Integrated Information Systems (ARIS)
Issuing organization:	University of Saarbrücken, IDS Scheer AG
Tool support:	ARIS Toolset
Period of activity:	since 1992
Publications:	[KNS92], [Ki99], [Sc01], [Sc02]
Inner organization:	monolith

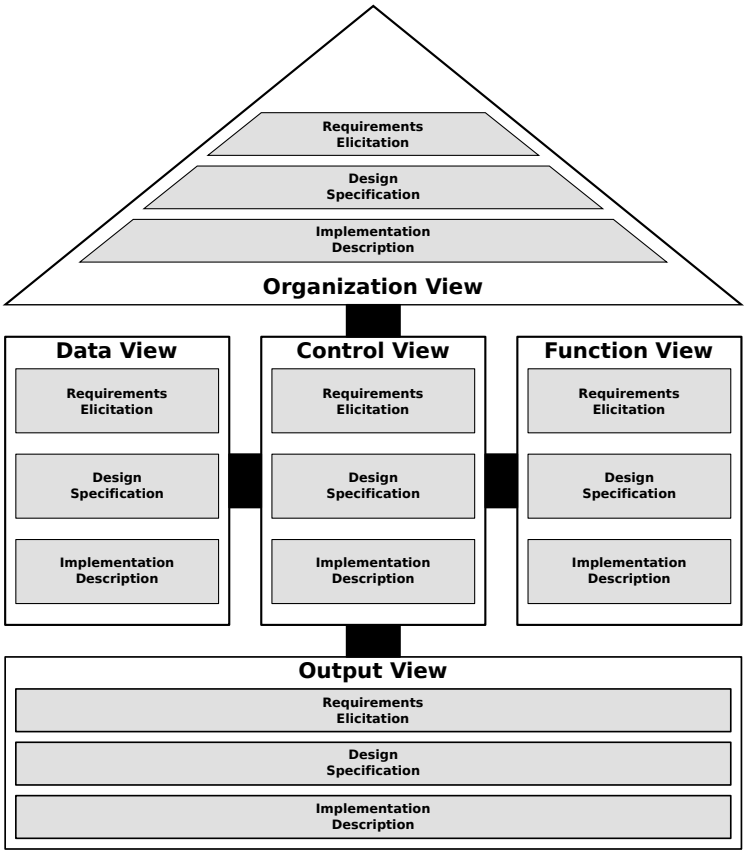


Figure 3.7: ARIS house

The so-called "ARIS house" (see Figure 3.7) introduces these views as follows:

- *organization view* describing the structure of the organization together with the lines of authority and the communication channels in the organization,
- *data view* describing the business data objects created, manipulated, and exchanged between the business functions
- *function view* describing the business functions that are to be executed by the organization as part of its value proposition
- *output view* describing the values, goods, and services delivered by the organization in executing its business functions

- *control view* interlinking the other views from a process-oriented point-of-view describing the business processes that are executed by organizations, work on business data objects, support the business functions, and are involved in delivering the goods and services

With the ARIS approach being centered around the question of modeling business information system especially from a process-oriented point-of-view, the meta-language of ARIS introduces manifold process-related concepts. From a strategic perspective, business processes are further distinguished into support processes and core processes, for which the model further supplies techniques to link to corresponding reference process models. The business functions of the organization are linked to the organization's goals and are distinguished with respect to the level of IT support provided. In a similar sense, business data objects are refined with respect to their IT involvement and the services provided by the organization are distinguished along their information-relatedness. Linking together these different perspectives on the organization, the concepts in a control view decompose the business processes into function and events, of which the former are executed by organizational units and create business relevant output. Figure 3.8 gives an overview on the meta-model of the ARIS approach against the background of the viewpoints provided by the ARIS house.

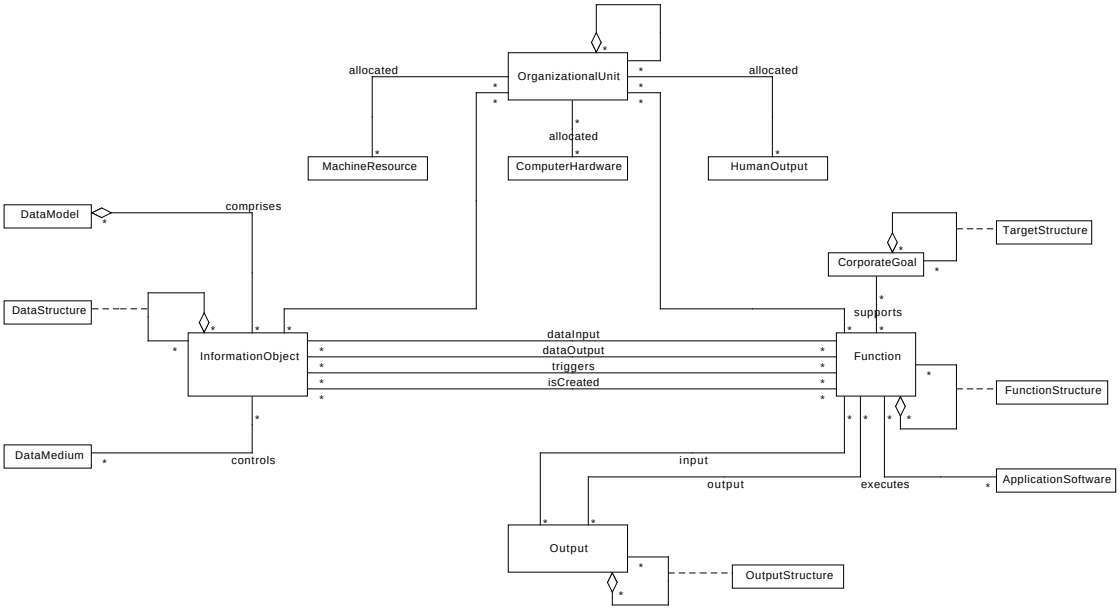


Figure 3.8: Overview of the meta-model of the ARIS approach

The event-function dichotomy of a control view is reflected in a corresponding modeling method, namely the one of the *event-driven process chain* introduced by Keller et al. in [KNS92]. An event-driven process chain details the structure of events and functions with additional *operators* that may be used to denote splits, joins, and decisions in the process execution. For each concept introduced in the ARIS meta-model, the ARIS approach further supplies a unique symbolic representation, making up the well-known and colorful appearance of the event-driven process chains and their extended versions. In the light of the aforementioned modeling capabilities, we classify the language aspects of the ARIS approach as shown in Table 3.5.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.5: Language classification for ARIS approach

3.3.3 The Generalised Enterprise Reference Architecture and Methodology (GERAM)

EA management or EA management-related approach	
Name of approach:	The Generalised Enterprise Reference Architecture and Methodology (GERAM)
Issuing organization:	IFIP-IFAC Task Force on Architectures for Enterprise Integration
Tool support:	none
Period of activity:	since 1994
Publications:	[BN94], [BN96], [IF99], [In99], [BNS03], [IF03], [No03], [In06]
Inner organization:	monolith

In the 1970s and the 1980s several EA-related frameworks have been developed (cf. CIMOSA in [CI04], the Zachman framework as discussed in Section 3.3.1, DoDAF in [De09a, De09b], or ARIS in Section 3.3.2). In response to the emerging number of frameworks in this area, the *International Federation of Information Processing* (IFIP) and the *International Federation of Automatic Control* (IFAC) established the *International Task Force on Enterprise Integration* aiming at the development of a reference framework that supports comparison and evaluation of existing approaches (cf. Bernus and Nemes in [BNS03, page 13]). As a result of the investigation, the Task Force developed the *Generalised Enterprise Reference Architecture and Methodology* (GERAM), which in 2000 became part of the international standard ISO 15740:2000 [In99]. The intention of GERAM is to provide a framework to compare, evaluate, and combine existing methodologies and modeling techniques.

GERAM consists of nine components as illustrated in Figure 3.9. In respect to the EA modeling language, only the following three components are of interest, although even these do not impose particular languages but define the criteria, which must be satisfied by an EA management approach (cf. [IF03, page 25]):

- *Generalised Enterprise Reference Architecture* (GERA): GERA describes the basic concepts to be used in enterprise engineering and integration projects. According to GERAM these concepts can be categorized as *human-oriented concepts*, e.g. capabil-

ities, skills, know-how, and roles of humans in the enterprise organization (responsibilities, authorities, etc.) and operation (the qualities of humans as resource elements), *process-oriented concepts*, e.g. functionality, behavior, entity life-cycles, and activities, and *technology-oriented concepts* describing the supporting technology involved in enterprise transformation and operation.

- *Enterperise Modeling Languages* (EMLs): EMLs define the generic modeling constructs for enterprise modeling. In particular, the EMLs provide constructs to describe and model human roles, operational processes, supporting information, and technologies.
- *Generic Enterprise Modeling Concepts* (GEMCs): GEMCs define and formalize the generic concepts of enterprise modeling. The following ways of formalization exists (in increasing order of formality): natural language explanations (*glossaries*), meta models describing the elements and their relationships (*information models*), and theories defining the meaning, i.e., semantics of enterprise modeling languages (*ontologies*).

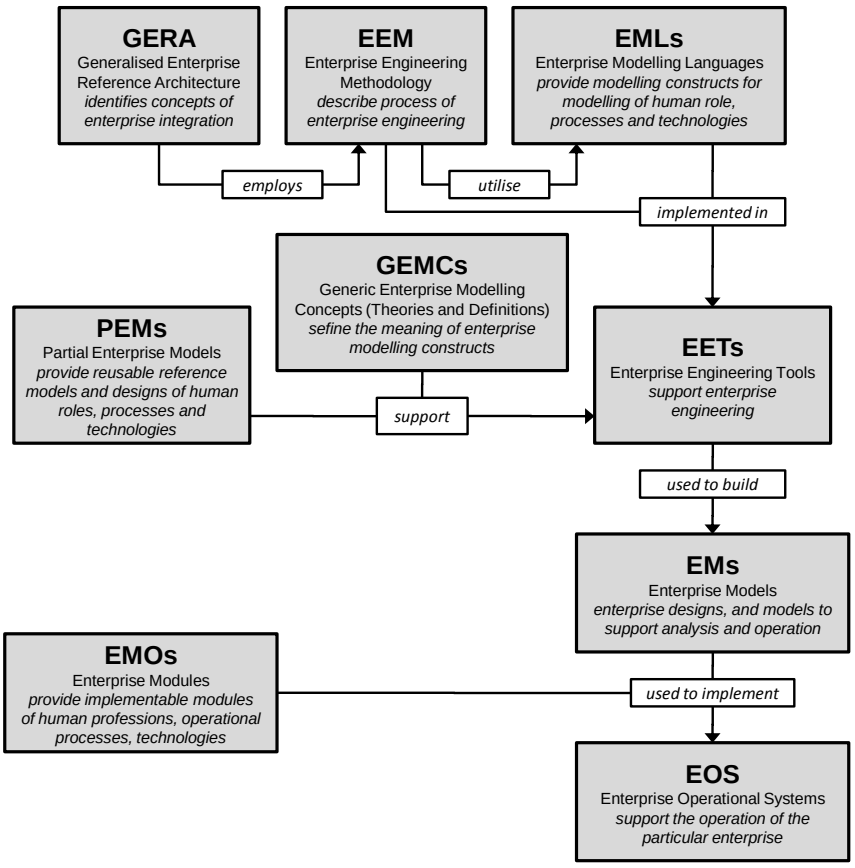


Figure 3.9: The components of the GERAM framework [IF99, page 5]

Reflecting the method-language-dichotomy as introduced by our analysis framework, GERAM distinguishes between methodologies for enterprise engineering (EEMs) and modeling languages (EMLSs) used by the methodologies. The EMLs and the GERA, which defines the foundations, are discussed subsequently alongside the analysis framework provided in Section 3.2.

GERA provides three dimensions for defining the scope and content of enterprise modeling as shown in Figure 3.10 (cf. [IF03, pages 42–44][10]), namely

- *life-cycle dimension* providing means for modeling entities according to the life-cycle activities,
- *genericity dimension* supporting the controlled particularization, i.e., instantiation, from generic and partial to particular, and
- *view dimension* enabling visualization of specific views of the enterprise entities.

Besides these dimensions, GERA opts to define the pragmatic purpose for each view and thus for the concepts to be considered by the EA management endeavor. Possible pragmatic purposes, e.g. support of design choices, simulation of processes to identify characteristics as cost or duration are given in [IF03, page 45].

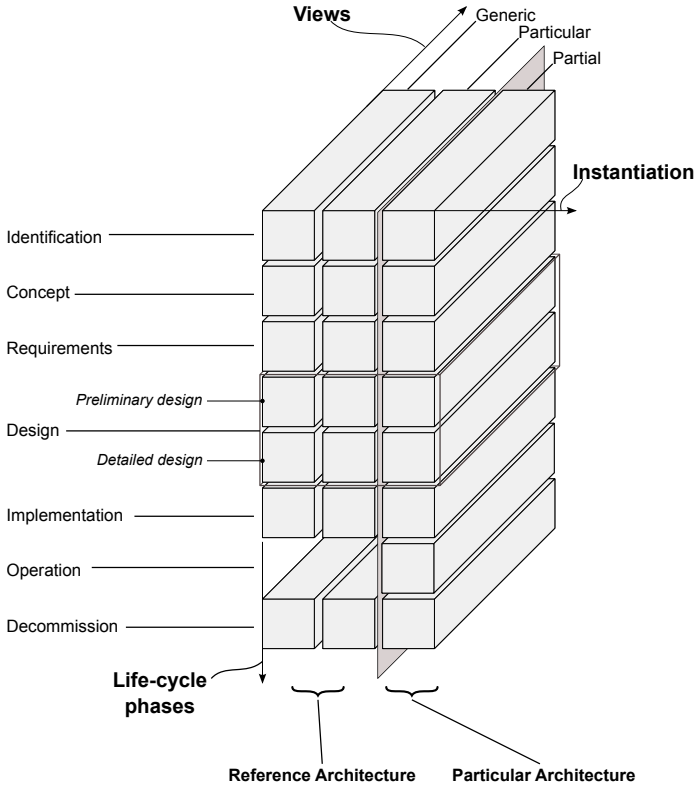


Figure 3.10: The components of the GERA modeling framework [IF99, page 18]

GERA defines a life-cycle for each constituting concept of the enterprise, which consists of the phases *identification*, *concept*, *requirements*, *(preliminary and detailed) design*, *implementation*, *operation*, and *decommission*. While most of the aforementioned phases are self-explanatory, the *concept* phase deserves a more in depth analysis with respect to our analysis framework. The phase is concerned with the definition of the entity's mission, vision, strategies, objectives, etc. [IF03, pages 32–34]. Thereby, cross-cutting aspects as strategies, projects, visions, and

[10] The GERA Modelling Framework represents the basis for the International Standard ISO 19439:2006. Framework for Enterprise Modelling [In06].

goals are linked to any concept considered during enterprise transformation. In line with the objective of GERAM to define requirements for EA (management) frameworks, no description how this relation should be conceptualized is given. Similarly, the concept of *life history* is discussed as main aspect of EA management approaches by GERA and the link to different kind of projects, e.g. engineering, redesign, or improvement projects, is explored and related to the phases of the EA concepts.

In addition, to the generalized propositions for a language for EA descriptions as discussed above, the EMLs define two requirements to enable integration of special purpose modeling languages (cf. [IF03, page 54]). First, every area as represented in the modeling framework must be covered for every enterprise entity type, and second, any model developed must be able to be integrated with models of other subject areas, if the information content of the model requires integration. The need to integrate different languages results from the distinct 'expressive powers' related to the intended purpose, e.g. description vs. analysis, of the languages. Thereby, a link between the methodologies used and the supporting languages can be established.

Keeping in mind the aim of generalization and therefore abstaining from presenting an explicit language for EA modeling, the requirements elicited by GERAM result in the evaluation given in Table 3.6.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.6: Language classification for GERAM

3.3.4 Semantic Object Model Approach (SOM)

In [FS95] Ferstl and Sinz diagnose a fundamental change in the way business information systems are understood via models. Whereas up to this point IS modeling centered around structural aspects of the systems, more recent approaches in those days started to identify IS with the set of interlinked business processes that the systems support. Especially with this perspective, the focus of IS modeling is broadened to not only incorporate the single system but also its enterprise environment, called the *context of the EA* by Ferstl and Sinz in [FS95]. Reflecting this understanding of the enterprise, the *Semantic Object Model* (SOM) approach introduces two key abstractions: the *business transactions* reflecting the exchange of services

EA management or EA management-related approach	
Name of approach:	Semantic Object Model (SOM) Approach
Issuing organization:	University of Bamberg
Tool support:	SOM Modeling Environment (in development—see `http://www.openmodels.at/web/som`, last-cited 04-05-2011)
Period of activity:	since 1994
Publications:	[Fe94], [FS95], [FS97]
Inner organization:	monolith

between *business objects* that conversely provide or consume such services (cf [Fe94]). Key principle of the approach is the *decomposition* of both objects and transactions into smaller parts thereof, getting from an abstract perspective on an enterprise to more concrete descriptions. This is further mirrored in the approach's framework for the EA as shown in Figure 3.11. The three layers of the EA cover the strategic goals as well as the value proposition of the enterprise, the supporting business processes, and the necessary organizational, technical, and physical implementation thereof.

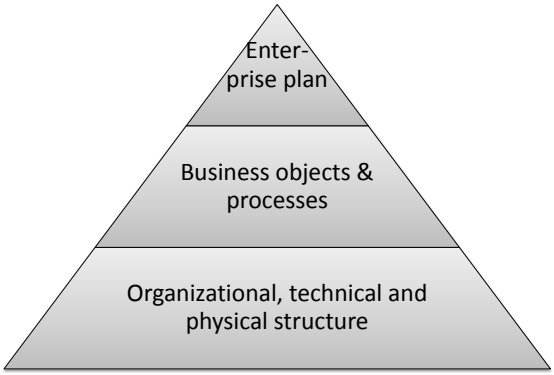


Figure 3.11: Enterprise architecture framework of SOM [FS95, page 8]

For the different modeling levels below the enterprise plan, Ferstl and Sinz [FS95, page 16] call for formal modeling techniques and languages based on the principle of decomposition. Using a BNF-like syntax, they describe decomposition rules which are to be applied in the V-model method. An exemplary rule reads as follows

$$O ::= \{O_1, O_2, [T(O_1, O_2)]\}$$

and describes that a modeler may decompose an object into a set of two sub-objects that are optionally linked with a transaction. Similar rules for decomposing transactions also exist.

Central to the SOM approach is the dichotomy of the object system (structure) and the process system (behavior) that pertains through all layers of the EA framework shown in Figure 3.11. Ferstl and Sinz reify this core distinction in the integrated meta-model of the approach as described in [FS95, page 16].

This model (cf. Figure 3.12) concretizes the two systems via corresponding concepts. Processes are decomposed into tasks, which in turn may be triggered by external events or form a sequence of tasks linked via internal events. For the object system, Ferstl and Sinz introduce

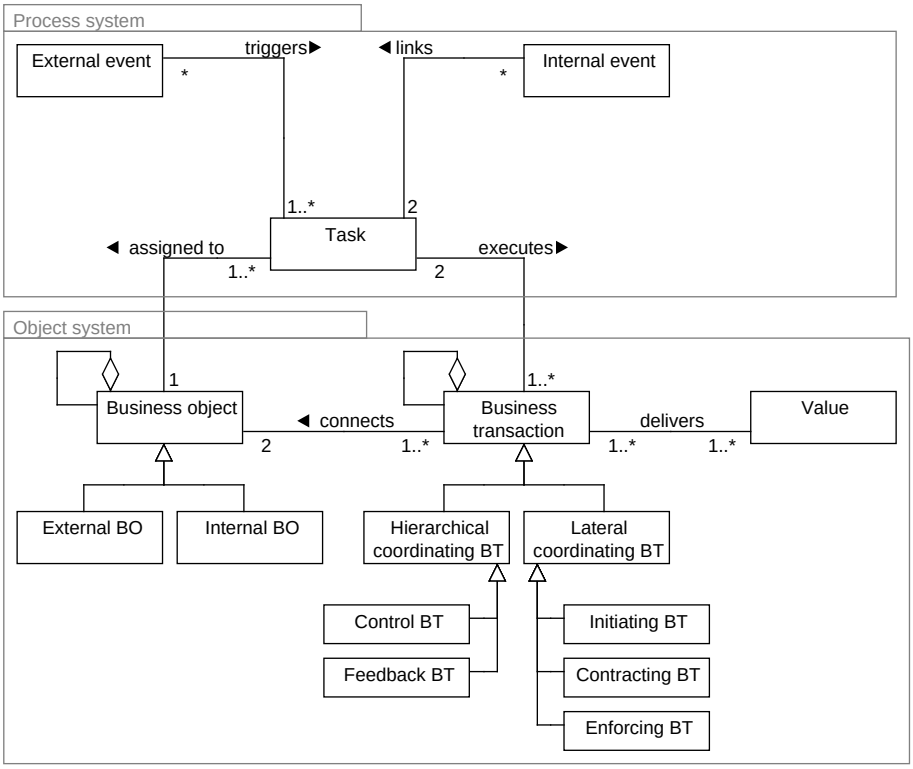


Figure 3.12: Meta-model of SOM [FS95, page 16]

a distinction between external and internal participating business objects. Concerning the kinds of transactions in which such objects are involved, two principles outlined in [Fe94] are incorporated into the model:

- *Negotiation principle*: This principle describes that every transaction may be seen as composed of initiating transactions (request), contracting transactions (acknowledge) and enforcing transactions (deliver). These kinds of transactions are used to describe interactions between actors that have no hierarchic control relationship in between (*lateral coordination*).
- *Hierarchic coordination principle*: This principle describes that hierarchic relationships can form the basis for coordination in a sense that one participating partner (super) controls the other partner (sub), which in turn provides feedback on the transaction.

Building on the principles, the meta-model and a set of transformation rules, as exemplified above, the SOM approach has a strong focus on the business perspective on both the enterprise as well as its information systems. Not directly reflected in the meta-model is the linkage between the tasks and the strategic goals as well as the value proposition of the enterprise. On the downturn, the conceptual object model that may well be identified with a 'classic' object-oriented model for IS is not directly linked to the meta-model but discussions on potential linkages are undertaken by Ferstl and Sinz in [FS95, FS97]. In terms of the analysis framework for the language aspect as described in Section 3.2, the SOM approach classifies as depicted in Table 3.7.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.7: Language classification for the SOM approach

3.3.5 Multi-perspective Enterprise Modeling (MEMO)

EA management or EA management-related approach	
Name of approach:	Multi-perspective Enterprise Modeling (MEMO)
Issuing organization:	University of Duisburg-Essen
Tool support:	MemoCenter NG
Period of activity:	since 1994
Publications:	[Fr94], [Fr98a], [Fr98b], [Fr99], [Ju07], [Fr08], [He08], [Ki08], [Fr09]
Inner organization:	explicit organization

In [Fr94] Frank motivates the need for models "of the whole enterprise or of parts of it" as means to address IS-related challenges as well as to leverage IS-enabled innovation, such as business process re-engineering. These models are required to span the different levels of the organization, ranging from a strategic perspective over an organization perspective to an IS perspective. The different perspectives should not be treated independently in order to

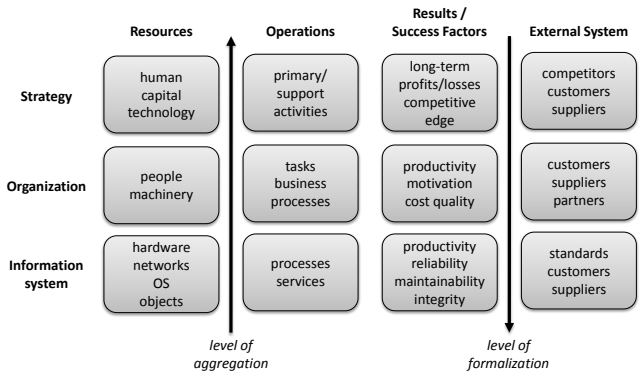


Figure 3.13: Viewpoints and foci of MEMO

reduce the risk of friction. Therefore, Frank devises the notion of the *multi-purpose enterprise modeling*[11] bringing together different viewpoints and foci on the enterprise (cf. Figure 3.13).

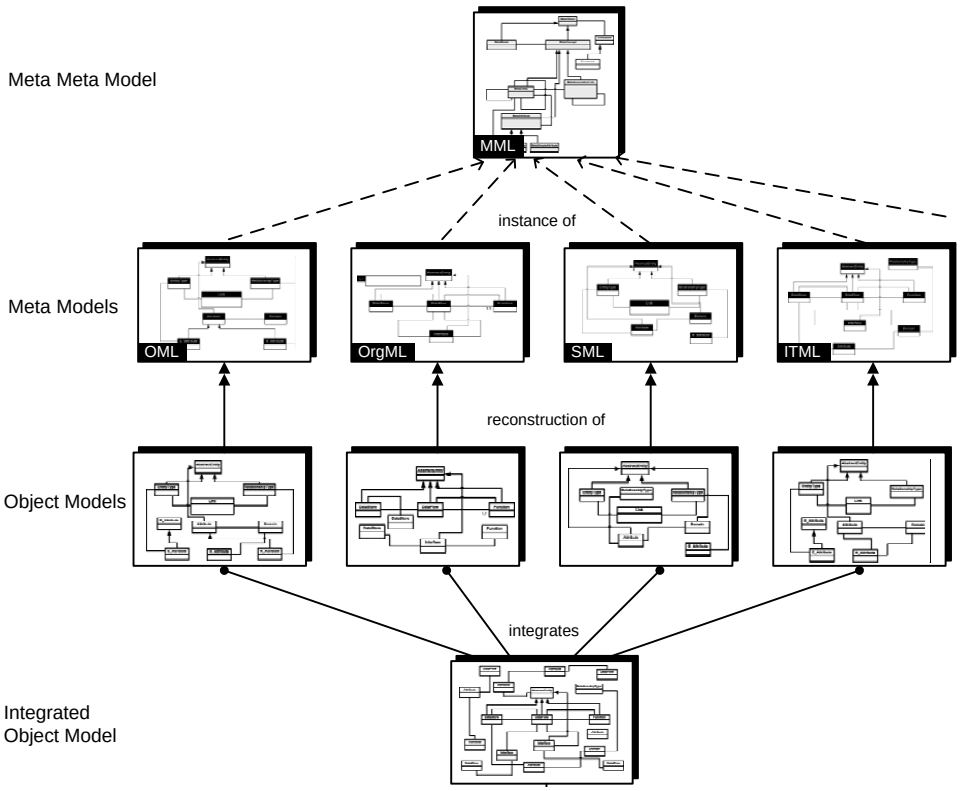


Figure 3.14: Make-up of MEMO's language stack [Fr09]

By design the *Multi-perspective Enterprise Modeling* (MEMO) is able to mirror the plurality of perspectives on the enterprise by supplying different languages. Each language brings along its unique set of concepts together with a dedicated notation (cf. Frank [Fr99]). The *Object Modeling Language* (OML), for example, [Fr98b] supplies typical object-oriented concepts, as classes, associations, and attributes, which may in turn be used to describe the object model underlying an information system. The *Resource Modeling Language* (ResML) of Jung [Ju07] supplies concepts for describing physical resources, as *computing devices*, intangible resources, as *patents* and *software*, as well as human resources together with their hard and soft skills. Each of these concepts is further detailed via corresponding attributes covering certain characteristics, such as *performance*. Switching from the resource perspective on hardware devices, the *IT Modeling Language* (ITML) of Kirchner [Ki08] provides concepts that detail the resources and allow describing IT-specific relationships among them. In particular, specific hardware devices are concretized, i.e., *scanners* or *printers* are distinguished via specialized classes bearing specific attributes. In a similar sense, the general concept software is further detailed via concepts e.g. to model *layered IT architectures* as well as *communication formats* for interfaces. The *Score Modeling Language* (ScoreML) of Frank et al. [Fr08] and Heise et al. [He08] reflects a cross-cutting perspective on the enterprise, dedicated to modeling arbitrary

[11] The approach was in subsequent publications renamed to *multi-perspective enterprise modeling*.

indicators that may be assigned to reference objects in the enterprise model. The mechanism of the reference object thereby concretizes a recurring principle of the MEMO approach. Different languages (*Organization Modeling Language* (OrgML) for organizational structures and processes as well as the *Strategy Modeling Language* (SML) for strategic modeling) committed to a specific perspective on the enterprise are grounded in a single meta-language and are interlinked via shared concepts. Figure 3.14 displays the basic make-up of the language stack of MEMO.

The *MEMO meta-language* (MML) has evolved over time since its initial publication in [Fr98a], but has retained its object-oriented nature to its most recent version, presented by Frank in [Fr09]. In all versions the meta-language supplies specific mechanisms to indicate that a language concept is a *border object*, i.e., is located at the intersection of two language perspectives. Summarizing above findings about the MEMO approach, we classify it according to our framework as shown in Table 3.8.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.8: Language classification for MEMO

3.3.6 The Open Group Architecture Framework (TOGAF)

EA management or EA management-related approach	
Name of approach:	TOGAF
Issuing organization:	The Open Group
Tool support:	TOGAF 9 Method Plugin for the Eclipse Process Framework Composer tool– see `http://www.opengroup.org/architecture/togaf/epf_intro.html` (last accessed 04-05-2011)
Period of activity:	since 1995 (TOGAF version 1.0)
Publications:	[Jo09], [Th09a], [Th09b]
Inner organization:	explicit organization

The Open Group is a vendor and technology-neutral consortium with the objective to foster information flow via open standards for enterprises. In 1995, The Open Group published the first version of TOGAF which was based on the *Technical Architecture Framework for*

Information Management (TAFIM) published by the Department of Defense. The current version 9.0 of TOGAF has been released in October 2009 [Th09a]. TOGAF is based on the terminology introduced in the ISO Standard 42010 [In07] and provides a method and supporting models and techniques for developing an EA management function. As a widely-used and known framework, the major players in the market of EA management tools have incorporated TOGAF in their tools (cf. the analysis of sebis in [Er06a, se05] and Matthes et al. in [Ma08]).

In line with other EA management approaches, TOGAF proposes to structure the EA in different architecture domains representing subsets of the overall EA [Th09a, page 10]. Thus, TOGAF distinguishes between

- *business architecture* concerned with strategic, governmental, organizational, and process-related aspects,
- *data architecture* describing the structure of an organization's data assets and data management resources,
- *application architecture* considers the application systems, their interactions, and their relationships to the business processes, and
- *technology architecture* describing the logical software and hardware capabilities required to support the deployment of business, data, and application services.

Answering the question which elements should be considered in an EA management endeavor, TOGAF presents the *core content metamodel*, which is illustrated in Figure 3.15. Therein, the core entities and relationships that make up an EA are described (cf. The Open Group in [Th09a, pages 376–409]).

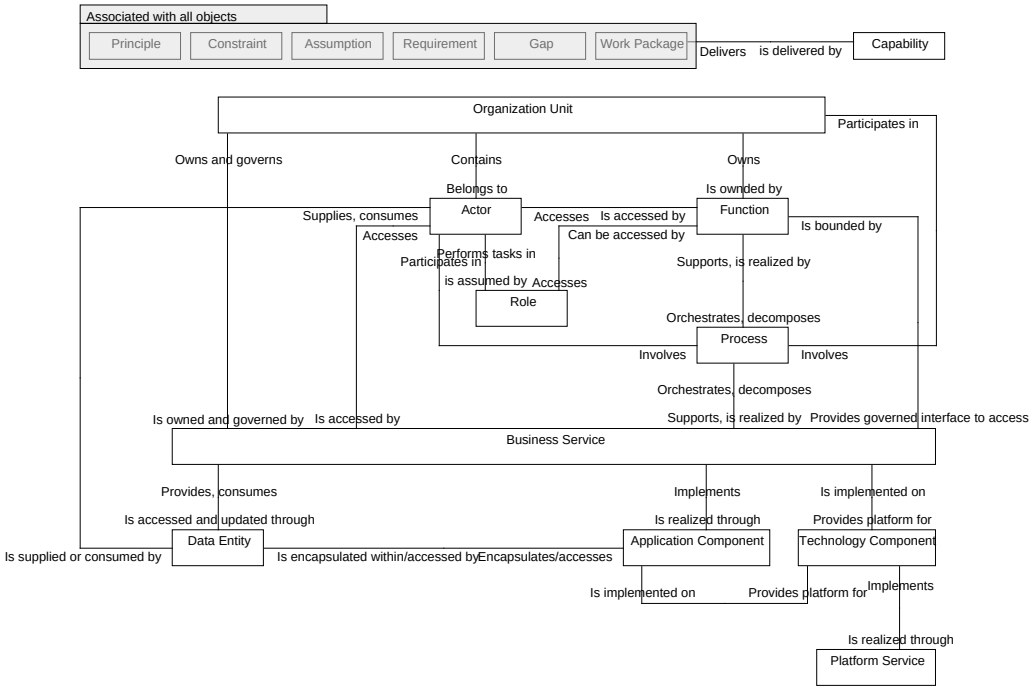


Figure 3.15: The core content metamodel of TOGAF [Th09a, page 376]

The core content metamodel provides entities for all architectural layers from business architecture, e.g. organizational unit or process, via data and application architecture, e.g. application component or data entity, to technology architecture, e.g. technology component or platform service. Besides the entities, which can be grouped to one of the architectural layers, TOGAF also introduces crosscutting entities associated with all objects among others principle, requirement, and work package. Thereby, the kind of relationship, e.g. affects, introduces, retires, is not discussed. TOGAF further provides six metamodel extensions (cf. The Open Group in [Th09a, pages 380–392]), namely

- *governance extension* to support operational governance by introducing concepts as goal, objective, measure, and contract,
- *services extension* to enable description and management of IS services in addition to business services,
- *process modeling extension* to allow detailed modeling of process flows by adding product, event, and controls,
- *data extension* to enable more sophisticated modeling and management of data by introducing entities like physical or logical data component,
- *infrastructure consolidation extension* to support consolidation endeavors by introducing physical and logical application components as well as the location entity, and
- *motivation extension* to enable measurement of business performance by introducing concepts as driver, goal, and objective.

For each of the above extensions, TOGAF describes the situation in which the respective extension should be used and the benefits it bears. Table 3.9 summarizes the key characteristics of TOGAF in general and the content meta model in special classified against the background of the language framework from Section 3.2.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.9: Language classification for TOGAF

3.3.7 The EA management approach of TU Lisbon

The research group of José Tribolet at TU Lisbon has a long history in what they call in their publications "information system architecture" (ISA) (cf. [VST03, pages 78–79]). In the recurring definition information system architecture is regarded as architecture on an intermediary level between enterprise architecture and software architecture laying a focus on "the representation of the IS components structure, its relationships, principles, and directives, with the main purpose of supporting business". With this broad definition and in the light of more recent publications of Vasconcelos et al. [VST07] and of Aveiro et al. [AST10b], we reconcile the group's research as contribution to the field of EA management. In these publications more emphasis is laid on method aspects related to EA, whereas the focus of the work initially was on modeling ISAs. For supporting concise modeling, Vasconcelos et al. propose in [VST03, page 79] the so called "CEO framework" that introduces a high-level meta-model for describing ISAs, subsequently refined in [CST09] to an embracing framework targeting EAs in five views[12]: an *organizational view*, a *business view*, an *information view*, a *system's application view*, and a *system's technological view*.

EA management or EA management-related approach	
Name of approach:	(Approach of TU Lisbon)
Issuing organization:	TU Lisbon
Tool support:	none
Period of activity:	since 2003
Publications:	[Va01], [VST03], [Va04], [VST05], [Ca07], [MaZT07], [VST07], [AMT08], [VST08], [CST09], [AST10a], [AST10b], [Av10], [CST10], [MZT10], [Za10]
Inner organization:	implicit organization

The CEO framework introduced by Vasconcelos et al. in [VST03, page 79] is realized in a meta-model profile for the UML [Ob10b] covering three *sub architectures* (cf. Vasconcelos et al. [Va01]), namely the *informational architecture* containing business relevant data types, the *application architecture* describing supportive applications, and the *technological architecture* representing the technologies used for application implementation. This trifecta of IS subarchitectures is embedded into the context of the EA (cf. Figure 3.16) by establishing links to business goals, business processes, and resources in an orthogonal manner. The corresponding basic meta-model (cf. Figure 3.17) establishes the abstract notion of the *Block* to represent ISA concepts of any sub architecture. Further concretizing this abstract representation of ISA concepts, Vasconcelos et al. denote in [VST03, pages 79–81] different IS and IT concepts as *applications* or *platforms* but also specialize blocks to concepts providing a more functional perspective on the IS sub architecture as e.g. *business service* or *IS service*.

The CEO framework does not impose restrictions on the specializations to be used, although the different techniques and metrics described in related publications target only a subset of the provided types (cf. [Va04], [VST05] and [VST07]). In this sense, one might argue that the approach of the CEO framework retains a notion of configurability but does not make prescriptions on when to use which of the specializations. Building on the concept of the Block and its specializations an exemplary case in [VST03, pages 81–83] outlines how

[12] According to the terminology adopted in our work, the term *viewpoint* would be more appropriate here.

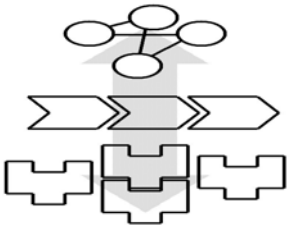


Figure 3.16: Goal/process/system framework according to Vasconcelos et al. [Va01, page 72]

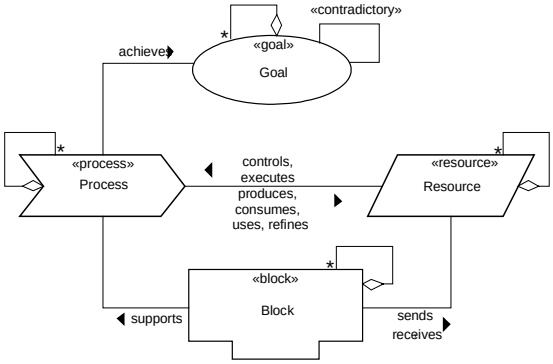


Figure 3.17: Meta-model of CEO framework according to Vasconcelos et al. [VST03, page 79]

current states of the architecture as well as target states thereof can be represented. The embracing perspective on structural aspects of the EA ranging from business to IT as well on functional aspects of the intermediary aspects of the IS is present in the more recent publications of Zacarias et al. [Za10] as well as Caetano et al. [CST10]. In [Za10] the goal and resource perspective on business processes (cf. Figure 3.17) is furthered with a concrete understanding of input and output factors as well as associated agents taking a particular role in the execution. Based on such functional view on business processes, Caetano et al. [CST10] describe mechanisms for decomposing processes into sub-processes that are independently embedded into their organizational, motivational, and operational contexts. In [Va04, VST05] Vasconcelos et al. discuss the concept of architecture analysis. The analyses target different aspects of the ISA with [Va04] putting emphasis on metrics for evaluating "IS/business alignment", whereas [VST05] discusses the applicability of 'classical' software architecture metrics, as "lack of cohesion" on ISA level. In [VST07] Vasconcelos et al. raise a multitude of relevant quality attributes, reified in corresponding questions linking to a subset of the 16 concrete metrics presented therein. For each metrics actual computation rules based on an according EA modeling language are provided. A comprehensive overview on the relationship between different metrics and according "ISA qualities" (questions) is further provided by Vasconcelos et al. in [VST08], emphasizing that these metrics and questions cover all sub architectures, with functional analyses of the business perspective being discussed by Zacarias et al. in [Za10]. Further, the questions target generic quality attributes of an architecture and are hence not linked to architecture-related goals as modeled according to the CEO framework's meta-model (cf. [VST03, page 79]). In the recent publication [AST10b] Aveiro et al. delineate two information models, namely the "viability model space" targeting the management function concerned with EAs and the "GOD model space". Latter information model targets the "Generation, Operationalization, and Discontinuation of objects", i.e., of architectural elements. While the viability perspective is specifically concerned with adaptations to the management methods, the GOD perspective is capable of describing arbitrary ENGINEER TRANSACTIONS that change the architecture or parts thereof. In this sense, the notion of the project is introduced into the canon of the modeling language in a cross-cutting manner targeting architectural elements from business to infrastructure level. In [AST10a, page 235] the notion is further developed and extended towards a lifecycle modeling for arbitrary archi-

tectural elements as well as for modeling the actors responsible for performing "Engineering Processes", i.e., changes to the architectural elements. A minor drawback nevertheless exists with this mechanism. In describing the GOD model space Aveiro et al. [AST10b, page 155] do not provide indications on how this information model is to be linked to the meta-model of the CEO framework. This may be ascribed to the focus of the work but leaves a willing user of the approach puzzled in a twofold way, not only as a description of the linkage is missing, but also as different underlying meta-languages (UML [Ob10b] and ORM [Ha05]) are used. Table 3.10 shows the classification of the approach of TU Lisbon according to the requirements delineated in Section 3.2.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.10: Language classification for the approach of TU Lisbon

3.3.8 The Systemic Enterprise Architecture Methodology (SEAM)

EA management or EA management-related approach	
Name of approach:	Systemic Enterprise Architecture Methodology (SEAM)
Issuing organization:	École Polytechnique Fédérale de Lausanne (EPFL)
Tool support:	SeamCAD [LW06]
Period of activity:	since 2003
Publications:	[We03], [RBW03], [LW04b], [LW05], [BW06], [LW06], [RW06], [RW07], [We07a], [We07b], [We08], [Re09]
Inner organization:	explicit organization

The *Systemic Enterprise Architecture Methodology* (SEAM) roots in the work [We03] of Wegmann, where he elaborates on the multi-disciplinary nature (see also Rychkova et al. [RBW03], Lê and Wegmann [LW04b], etc.) of EA projects and the resulting need to support these projects with methods and models. Here thereby outlines the central method-model-dichotomy, also alluded to as *method-notation*-dichotomy by Rychkova et al. in [RBW03], of SEAM. Diagnosing a lack of support in respect to the method part of EA project support, SEAM seeks to complement existing approaches with additional methodical guidance. An example for such complementation is given by Wegmann et al. in [We08], where the

Zachman Framework (cf. Section 3.3.1) is augmented with a *systemic conceptualization* based on SEAM. The systemic nature of SEAM-based conceptualizations makes up another predominant characteristics of the approach in the different publications. Wegmann formulates in [We03, pages 486–488] the underlying *systemic paradigm* based on the SEAM ontology, which itself builds on the prefabrics of RM-ODP [In96], and the *constructivist* epistemologic perspective. In line with the constructivism principle, SEAM assumes that knowledge about a system is relative to the observer, meaning that no observer-independent descriptions of reality exist. By this fact Wegmann motivates in [We03, page 487] a hierarchical understanding of any system considering different *levels of reality* owned by dedicated stakeholders. This understanding pertains to the work on SEAM, especially through the foundational language descriptions by Lê and Wegmann [LW04b, LW05], and is further mirrored in the methodology's tool support (SeamCAD) presented by Lê and Wegmann in [LW06].

A key characteristics of the modeling language associated with SEAM is outlined by Wegmann in [We03, page 488] as part of the ethics of the systemic paradigm. He states that each concrete EA project needs to develop an actual enterprise model. In this sense the language prescriptions of SEAM are formulated on an abstract level allowing the project team to reify them and thereby to configure the language as needed. Central aspect of configurability in respect to the language is the intrinsically hierarchic understanding of the enterprise system in the method, see e.g. Rychkova et al. [RBW03] or Lê and Wegmann [LW04b]. In the latter article, foundational theories for the language are outlined, namely the *Living System Theory* of Miller [Mi95] and the conceptualization of RM-ODP [In96]. Based on these theories a set of three basic concepts, namely *computational objects*, *information objects*, and *actions* of EAs is devised. In subsequent work of Lê and Wegmann [LW05] these concepts are complemented with a textual description of their semantics as well as a formalization based on the Alloy 2.0[13] language. At this point in time a different formalization perspective on the SEAM language had already been outlined by Rychkova et al. in [RBW03], being concretized in the subsequent work [RW06] of Rychkova and Wegmann. This work describes an executable *Abstract State Machine* (ASM) interpretation of the language concepts. The complete information model underlying the SEAM language (cf. [LW05, page 185]) puts special emphasis on the hierarchical nature of the according descriptions by annotating the layer-spanning relationships via specific UML stereotypes. This reflects the aforementioned dimension of configuration, in which a using EA project team has to find an appropriate level on its own. While only implicitly alluded to in the information model specification, the same language is used to describe as-is and to-be states of the underlying system building on two different kinds of abstractions (cf. Wegmann et al. [We05, WRL05]): an *organizational* and a *functional* one. Where the functional abstraction is used to specify what the system under consideration does or is intended to do, the organizational perspective details how this functionality is provided internally. The linkage between these perspectives further allows to conduct gap analysis between actual and intended function abstracting from implementation specific details. The ASM-based technique for performing such analyses, as described by Rychkova and Wegmann in [RW06], introduces further language concepts, namely relationships between *actions* and *properties*, of which the latter reflect specific qualities of *computational objects*. The three types of relationships are concretized by Rychkova and Wegmann in [RW07] making use of the *Business Process Modeling Notation* (BPMN) [Ob10a] to model action-action-relationships as well as of a pre- and post-condition based technique for action-property-relationships. In

[13] For additional information about Alloy see `http://alloy.mit.edu/`, last accessed 04-05-2011.

this context, post-conditions are used to describe property changes mediated by the according action. Finally, two types of property-property-relationships are introduced: *part-of* and *used*, reflecting hierarchical (inter-layer) and lateral (intra-layer) dependencies between properties, respectively. In the most recent publication [Re09] of Regev et al. SEAM models are revisited against the background of four generic questions *utility*, *warranty*, *value*, and *risk*. Each of these questions is stated on an abstract and general level, for which a using project must find a specific operationalization into concrete metrics. Table 3.11 summarizes the key characteristics of the modeling language employed by SEAM classified against the background of the framework from Section 3.2.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.11: Language classification for the Systemic Enterprise Architecture Methodology

3.3.9 The ArchiMate language

EA management or EA management-related approach	
Name of approach:	ArchiMate language
Issuing organization:	Telematica Institute / Novay
Tool support:	ArchiMate Workbench [La05, La09a]
Period of activity:	since 2003
Publications:	[Jo03], [Jo04b], [La04], [La05], [Jo06], [Ar07], [La09a], [Jo10], [QEJ10], and `http://www.opengroup.org/archimate/doc/ts_archimate/`, last accessed 04-05-2011
Inner organization:	monolith

The ArchiMate modeling language for EAs has a long development history starting with the early work [Jo03] of Jonkers et al., who therein outline the key requirements and principles of what would later become a language for coherent enterprise architecture descriptions. In particular, they introduce a notion of flexibility in respect to the model, plurality in respect to visualizations as well as viewpoints, and integrability with respect to existing modeling documentations. Building on this basic understanding, Jonkers et al. describe the three core aspects of an enterprise that any suitable modeling language should account for, namely *structure*, *information*, and *behavior*. For each of these aspects as well as for the three relevant

layer, namely *business*, *application*, and *technology*, the ArchiMate modeling language provides appropriate concepts and visualizations. Figure 3.18 summarizes the *architecture framework* behind the ArchiMate language as defined by the aforementioned aspects and layers.

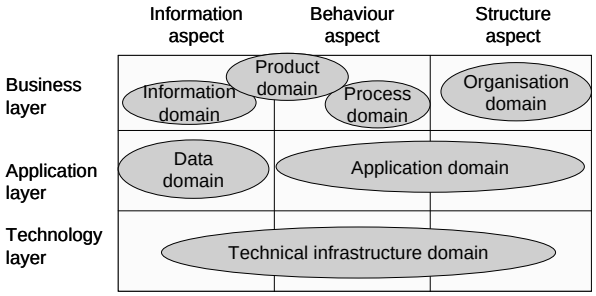


Figure 3.18: The ArchiMate architecture framework according to Jonkers et al. [Jo03]

In [Jo04b] Jonkers et al. call ArchiMate an *umbrella language* that is used to integrate different architecture-related design models from different languages. Central to this understanding is the flexibility of ArchiMate in respect to its level of specificity, while a remark puts that the ArchiMate language resides on a higher level of abstraction than the design languages. According to Jonkers et al. this ascribes to the fact that "a single language covering all domains [...] would probably result in an unworkable behemoth" [Jo04b, page 258]. Jonkers et al. discuss in [Jo03] key requirements for the ArchiMate language, of which flexibility in respect to the viewpoints as well as to the meta-model are deemed of highest interest. With respect to the former requirement, Jonkers et al. discuss mechanisms to decouple visualizations from underlying EA models, making it possible to specify stakeholder-specific viewpoints. The latter requirement is addressed with a core meta-model consisting only of six concepts as shown in Figure 3.19.

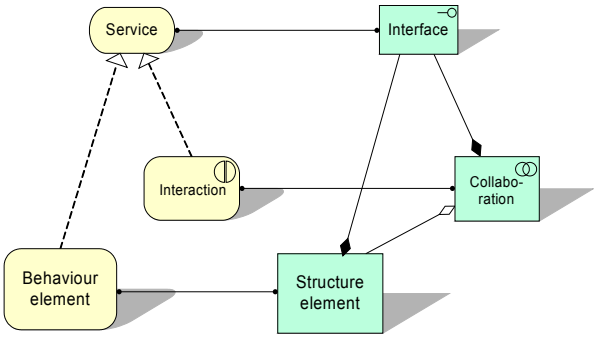


Figure 3.19: The core concepts of the ArchiMate meta-model according to Arbab et al. [Ar07]

As stated by Arbab et al. in [Ar07] these concepts reflect three central dichotomies of an EA, namely a) *internal-external* which resembles a white-box/black-box differentiation, b) *individual-collective*, and c) *behavior-structure*. In [La05, pages 90–105] and later in [La09a, pages 91–106], Lankhorst gives a more detailed exposition of the meta-model, therein describing that the six concepts are specialized on the three architectural layers of business, application, and technology. Complementing what would otherwise turn out to be a language for describing layered architectures, Lankhorst further introduces a set of so-called "structural relations" that may be used to link structural elements on arbitrary layers. In addition, *behav-*

ioral relations are introduced as means to link behavior on different architectural levels. These two set of relations may be considered a core source of ArchiMate's flexibility, as they allow to interrelate the layers even if no strict layering can be achieved. More precisely, an interaction element from the business layer can be directly linked to the technical collaboration that it stems from. On the opposite, this flexibility calls for responsible modeling. With respect to analytical methods applied on the model, Lankhorst describes in [La09a, pages 206–208] a technique for annotating models with the necessary quantitative information, mirrored by corresponding attributes that augment the core meta-model. Based on these augmentations, he devises general rules and mathematical models that may be used to compute or derive quantitative information on one architectural layer from corresponding information on another layer. In particular, two sets of rules are described, namely one for *top-down workload calculation* and one for *bottom-up performance calculation*. When it comes to the customization of the modeling language, both editions of Lankhorst's book [La05, La09a] provide *guidelines for modeling*, describing a preliminary phase *before to start* and a phase for deciding *what to capture* in the model. Complementing these content-related discussions, also guidelines on the visual appearance of the language are given by Lankhorst in [La09a, pages 144–150]. There, topics of layout, utilization of color, and establishing a unique 'symbolic terminology' are accounted for. In the recent whitepaper [QEJ10, pages 14–15] Quartel et al. introduce additional language concepts that can be used to model goals and principles affecting the EA. For the former concept, a decomposition into "assessment", e.g. concretizing metrics, is incorporated into the language. The principles remain on an abstract level and cannot be concretized using standards or constraints. Another recent whitepaper [Jo10, page 9] specifically targets transformation aspects of EA management as reflected in the concepts of the "project" and the "project result", respectively. Using these means, the language can express planned states for the EA, although specific aspects of time are not incorporated. In particular, the language prescriptions remain limited with respect to the lifecycle of EA elements. Summarizingly, it can be said that the ArchiMate language presents itself as a comprehensive EA modeling language that is useful for describing current (and future) states of an EA. This leads to a classification of the ArchiMate language as shown in Table 3.12.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.12: Language classification for ArchiMate language

3.3.10 The EA management approach of KTH Stockholm

EA management or EA management-related approach	
Name of approach:	(Approach of KTH Stockholm)
Issuing organization:	KTH Stockholm
Tool support:	EA Tool [Ek09]
Period of activity:	since 2004
Publications:	[Ek04], [Jo04a], [GLS06], [JNL06], [JE07], [Jo07], [La07], [La08a], [LJ08], [Nä08], [Bu09g], [Ek09], [KUJ09], [RNE09]
Inner organization:	monolith

The EA management approach developed at KTH Stockholm aims at providing decision support for IT management in enterprises, among others for the CIO, as key responsible for strategic IT-related decisions. Ekstedt et al. outline in [Ek04] this focus for the topic of EA, further relating it to the disciplines of software engineering and IS engineering, on whose methods EA management is required to build. With the focus on support for decision making, it is not surprising that the approach's contributions center around techniques and models for analyzing EAs with respect to specific qualities. Thereby, the approach of KTH Stockholm seeks to complement existing approaches for modeling EAs, such as the pattern-based approach of Techische Universität München (cf. [Bu09g]) or the ArchiMate modeling language [Nä08]. Complementing the work on evaluating architecture qualities, Ekstedt et al. outline in [Ek09] an analysis tool.

The KTH approach specifically accounts for the cost/utility ratio of EA-related information and does hence not aim to provide a comprehensive information model. Ekstedt et al. provide in [Ek04] a core information model, linking business-level concepts as *business processes* and *organizational units* to IS-related concepts, e.g. *software components*. The model is nevertheless complemented with guidelines for utility-based adaptation, specifically highlighting that every using organization has to select the concepts with highest utility compared to the costs. In [JNL06, Jo07] Johnson et al. introduce a key concept of their approach, the so-called *extended influence diagram*. Building on these diagrams, Johnson et al. devise a technique that can be used to describe quality characteristics mirroring goals and relate these abstract characteristics to measurable, i.e., operationalized, characteristics of the architecture. In particular, the influence diagrams can be used to designate which architectural characteristics may be influenced by management activities, i.e., are *controllable*. Exemplifying, how the technique may be used to operationalize the architectural characteristic of *maintainability*, Lagerström provides in [La07] an exemplary extended influence diagram and derives a corresponding information model for maintainability analysis. This model is strongly system-centric, i.e., describes application systems in the context of the maintenance processes, the operating platform, and the available documentation. In [LJ08] Lagerström and Johnson further detail this model, in particular the maintenance processes, into *change activities* on system and component level. In more recent publications, the notion of the influence diagram is replaced with the more formalistic understanding of the *probabilistic relational model* (cf. Getoor et al. [Ge07]). This modeling technique based on conventional relational models allows go beyond the influence diagrams by making explicit uncertainty with respect to the contained information. The publications of Raderius et al. [RNE09] and of Sommestad et al. [SEJ08] show the applicability of a technique on the architectural characteristics of *availability* and *security*, respectively. Both publications provide specific information models reflecting the

characteristics constituting the relationship modeling. Närman et al. take in [Nä08] a more general perspective on different architectural qualities using Bayesian networks to complement the relationship models, which are themselves based on the ArchiMate architecture modeling language (cf. Section 3.3.9). In [JE07] Johnson and Ekstedt supply a set of information models corresponding to different architectural viewpoints, e.g. an *organization viewpoint*, a *business process viewpoint* and *application usage viewpoint*, of which the latter introduces a service perspective on applications. A further information model is concerned with goals, more precisely with their interdependencies. Lagerström et al. describe in [La08a] how Bayesian belief networks can be used to discover correspondences between information models originating from different sources. In particular, the networks can be used to find out, which classes originating from different information models may be identified with each others. The network-based technique can be employed in step three of the guidelines provided by Källgren et al. in [KUJ09], where the information models underlying the stakeholder-specific viewpoints are integrated into a comprehensive information model. In the light of above properties, the approach of KTH Stockholm is classified as shown in Table 3.13.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.13: Language classification for approach of KTH Stockholm

3.3.11 The EA^3 Cube™

EA management or EA management-related approach	
Name of approach:	EA^3 Cube^{TM}
Issuing organization:	Scott A. Bernard, Syracuse University
Tool support:	none
Period of activity:	since 2005
Publications:	[Be05], [Do08], [BG09], [Do09a], [Do09c], [Do09b]
Inner organization:	monolith

In [Be05] Bernard presents his experience gained through his work in practice and academia in the area of EA management. Identifying the need for a textbook for students, Bernard wrote the book "An Introduction to Enterprise Architecture Management" in which he presented

the EA^3 Cube^{TM} approach. According to Bernard, a framework for EA management follows the dichotomy of language (*what*) and method (*how*) [Be05, page 75] and consists of six basic aspects:

1. an EA governance process, which links the EA management function to other enterprise-level management functions,
2. a repeatable methodology describing the management function,
3. a framework representing the core elements and layers, i.e., the scope, of the initiative,
4. an integrated set of artifacts, i.e., architectural descriptions,
5. documentation tools with a repository to support architectural descriptions, and
6. associated best practices, which guide the implementation of the management function [BG09, page 220].

These constituents are further detailed subsequently along the EA^3 Cube^{TM}.

In [Be05, pages 38–40] Bernard introduces a framework for EA descriptions—the EA cube, which consists of three dimensions (see Figure 3.20). The first dimension is concerned with the different architectural *levels* ranging from high-level strategic goals and initiatives to technical network and infrastructure aspects on the bottom. The *segments* dimension divides the overall EA in different parts covering one ore more lines of business, i.e., distinct areas of activity in the organization. From a holistic perspective, these lines of business should cover all architectural layers. Complementingly, the third dimension *artifacts* refers to the components that make up the organization. Thereby, vertical components that serve one line of business but may affect more than one architectural level and horizontal, i.e., crosscutting components, which serve several lines of business, are distinguished.

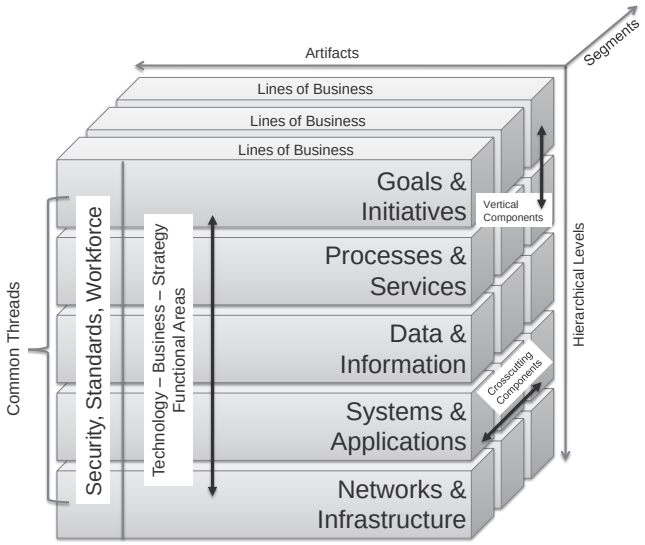


Figure 3.20: The EA^3 cube description framework [Be05, page 40]

The above introduced concept of cross-cutting components can not be put on a level with the idea of cross-cutting aspects as introduced in Section 3.2. As the cross-cutting components as introduced by Bernard in [Be05, page 40] relate to instances, while the cross-cutting aspects

introduced in Section 3.2 related to the class level. Goals of an EA management endeavor, which represent a cross-cutting aspect in terms of the analysis framework, are referred to by Bernard in [Be05, pages 64–69] and their relation to supporting components of the EA is detailed [Be05, page 181]. Similarly, strategies, goals, and measures are discussed to quantify the EA management endeavor [Be05, pages 72–74] but only from a methodological perspective.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.14: Language classification for EA^3 Cube Framework

3.3.12 The EA management approach of Niemann

EA management or EA management-related approach	
Name of approach:	(Niemann)
Issuing organization:	act! consulting (consultant)
Tool support:	none
Period of activity:	since 2006
Publications:	[Ni06]
Inner organization:	monolith

Klaus Niemann is managing director of act! consulting, a consultancy specialized in the topic of EA management. With more than 20 years of experience in this field, his book "From Enterprise Architecture to IT Governance" [Ni06] summarizes the key lessons and guidelines for EA management from Niemann's perspective. Central to the EA management approach is a method called "EA cycle" consisting of phases for documentation, analysis, planning, and acting. Within this method cycle, the language-related prescriptions concentrate on the documentation phase. In this phase the scope and reach of the EA management function and the EA description is defined, further populating the corresponding model with relevant information [Ni06, page 41]. Niemann proposes to structure the EA model into three main levels called "business architecture", "application architecture", and "systems architecture", respectively [Ni06, page 77]. Each sub-architecture is further complemented with a detailed description of the concepts and relationships contained as well as with information on cross-architecture relationships. Futher, functional and non-functional requirements are alluded to as cross-cutting aspects targeting different sub-architectures as shown in Figure 3.21.

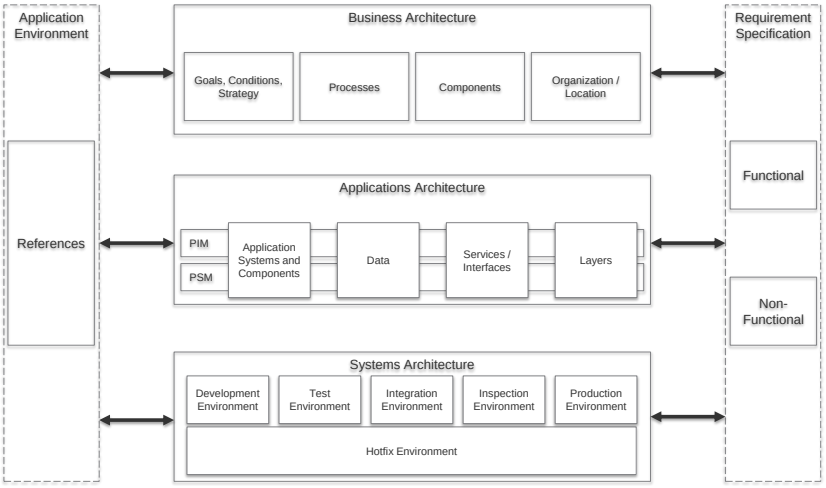


Figure 3.21: The EA framework of Niemann [Ni06]

Understanding EA management as highly interlinked with existing management processes in the enterprise, projects are considered in the EA planning phase [Ni06, page 161]. The corresponding description nevertheless remains on a textual level. In particular, Niemann does neither provide prescriptions on how to incorporate projects into the EA description nor on how to reflect their effects. In a similar sense the topic of principles, called "reference models" by Niemann [Ni06, page 97], is only textually alluded to, omitting concrete modeling concepts for reflecting principles or relating them to architectural elements. For the analysis phase of the EA cycle, quantitative as well as qualitative analysis techniques are provided and detailed [Ni06, pages 126–152]. While being detailed with respect to the steps of the corresponding techniques and with respect to possible metrics concretizing EA-relevant questions on *homogeneity* or *complexity*, manifestations of the questions in the EA modeling language are not provided. In summary, the approach of Niemann can be characterized with respect to language aspects as shown in Table 3.15.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.15: Language classification for the approach of Niemann

3.3.13 The EA management approach of the University of St. Gallen

EA management or EA management-related approach	
Name of approach:	(Approach of the university of St. Gallen)
Issuing organization:	University of St. Gallen
Tool support:	ADOben [Ai09c, Ai09b]
Period of activity:	since 2007 (2003)
Publications:	[ÖW03], [WF06], [AS07], [Br07], [KW07], [SS07], [Ös07], [Ai08b], [ARW08a], [Fi08], [HW08], [WS08], [Ai09c], [Ai09b], [AW09], [Ku09], [KW09], [RA09], [AG10]
Inner organization:	explicit organization

At the university of St. Gallen there is a long history in the field of *business engineering* building on the groundworks of Österle and Winter in [ÖW03]. Understanding business engineering as an holistic approach for designing organizations in the information-age, hence closely related to EA management, one can claim that the university of St. Gallen is into the topic of EA management since 2003. Nevertheless, at least since 2007 more and more of the work becomes specifically devoted to EA management topics. In 2007 Winter and Fischer [WF06] introduced the layered framework for the EA as shown in Figure 3.22.

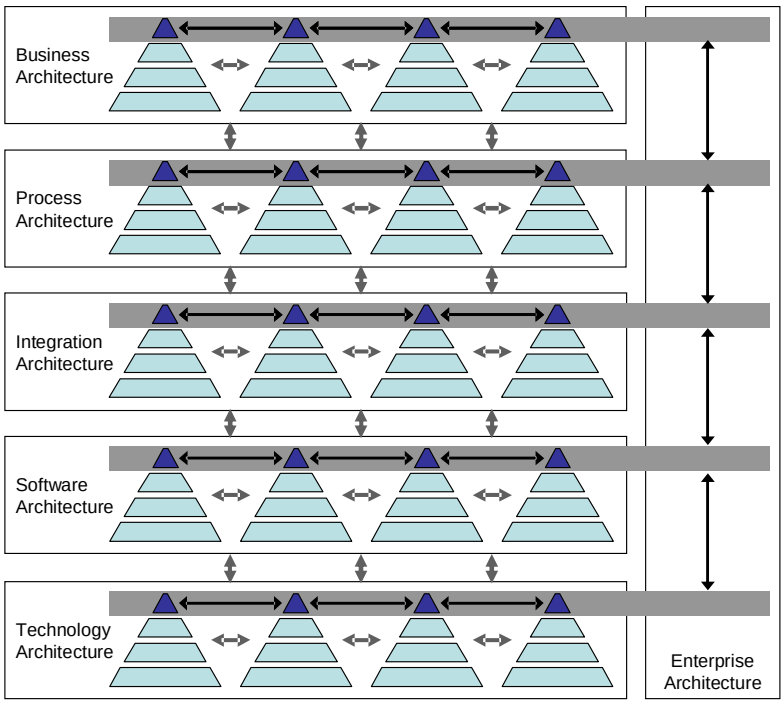


Figure 3.22: Essential layers of an EA [WF06]

Backbone of the EA modeling language employed in the approach of the University of St. Gallen is the *core business meta-model* described by Österle et al. in [Ös07]. This meta-model, shown in Figure 3.23, introduces the basic concepts that reside on the different layers of an EA.

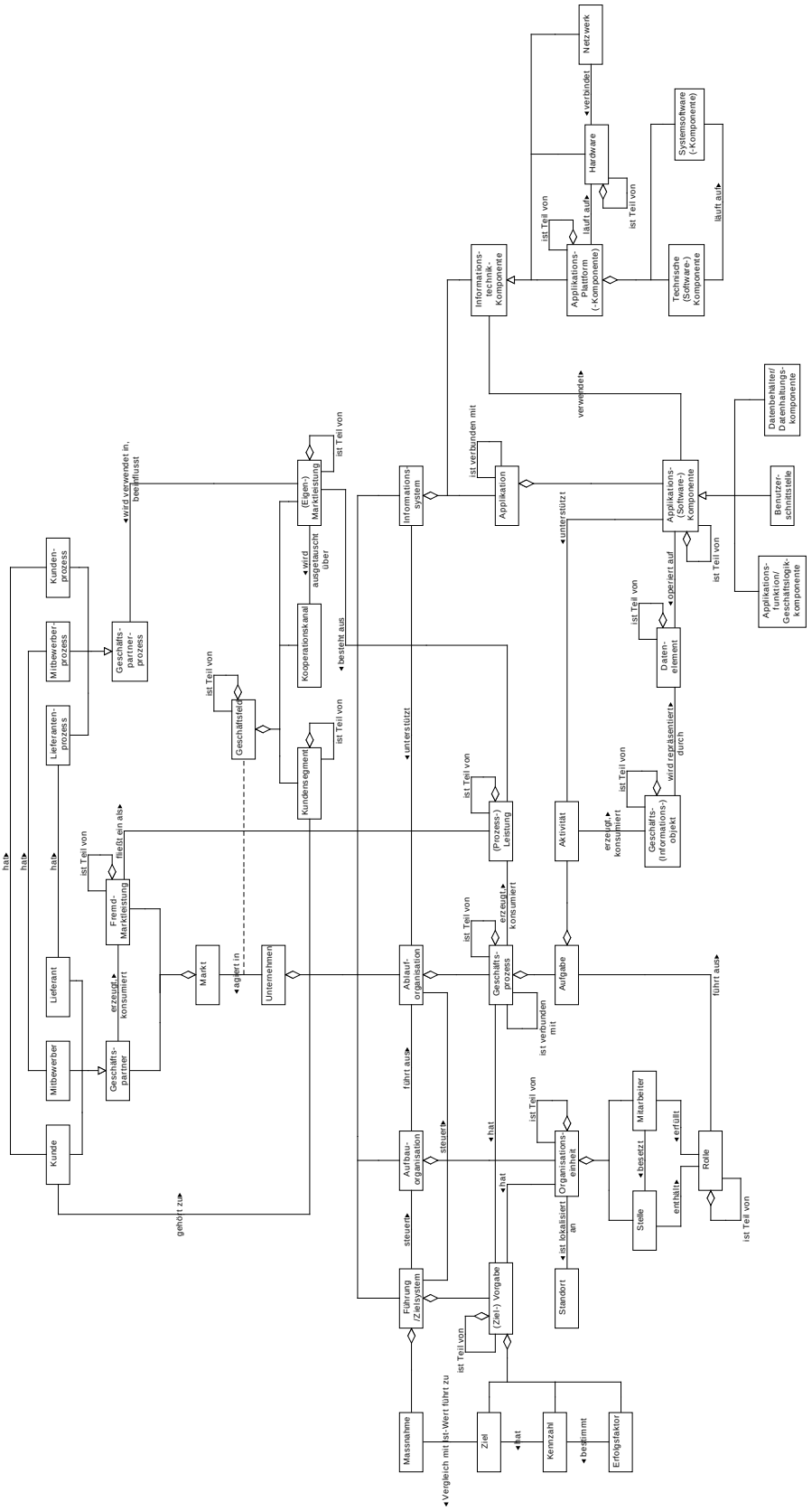


Figure 3.23: Core business meta-model of Österle et al. [Ös07]

The concepts contained in the core business meta-model are described on an abstract level, focusing on relationships and abstaining from giving details on the attributes of the concepts. Further, Österle et al. call for an intuitive understanding of the concepts giving no explicit glossary of terms. In [Br07] Braun complements the core business meta-model[14] with more explicit sub-models for *strategy modeling*, *organization modeling*, and *IS modeling*. The corresponding sub-models introduce the relevant model concepts together with a textual definition of their semantics and additionally provide a notation, i.e., define the symbols used to represent a specific concept. The different sub-models are further complemented with techniques and viewpoints that are to be applied as part of performing EA management. Braun puts special emphasis on the relationships between the sub-models [Br07, pages 171–178] and delineates how other modeling language, as the UML, may link to specific sub-models [Br07, pages 178–181]. Notwithstanding, the different sub-models ought to be used as a whole in order to achieve comprehensive business modeling. In a brief side-note, Braun discusses how service-oriented concepts may be incorporated into the meta-model, an idea taken up by Aier and Winter in [AW09]. There, they advocate for introducing an additional architectural layer, called *alignment architecture* to decouple *supply* and *demand* layers in an architecture. Exemplifying this along a decoupling in the IS support for business processes, Aier and Winter introduce the concept of the *enterprise service* on an intermediary level. These services are identified using the domain clustering technique as described by Aier and Schelp in [AS07].

Aside from aforementioned discussions on the additional abstractions in architecture modeling, Kurpjuweit and Winter discuss in [KW07, pages 6–10] the need for organization-specific meta-models. This is explained with the high maintenance effort connected to describing parts of the overall organization that are not needed for EA management. As the understanding of what is needed for EA management may differ from organization to organization, Kurpjuweit and Winter describe a "systematic approach to meta-model engineering" that enables an organization to develop a specific meta-model. The approach consists of five steps as follows:

1. identify relevant concerns, i.e., areas-of-interest,
2. elicit stakeholder requirements and derive situated metrics,
3. select viewpoints and create a viewpoint relationship overview,
4. select or design meta-model fragments, and
5. integrate meta-model fragments.

Whereas the general approach may be regarded as adaptation of classic domain modeling techniques, especially Step 4 deserves special attention. The selection of meta-model fragments points towards a collection of such fragments that was at that time yet to be compiled. Two years later, Kurpjuweit made available such a collection in [Ku09]. In the thesis he describes different partitions of the core business meta-model that may be regarded meta-model fragments in their own rights. Each fragment is further complemented with at least one viewpoint that may be applied to visualize the specific information contents conveyed in the fragment, more precisely in instantiations thereof. For particular examples of viewpoints in application cases Kurpjuweit [Ku09, pages 224–228] further details which additional information may be attached to a viewpoint in order to embed it into the situated context of an EA management method. Based on the common terminology of the core business meta-model,

[14] According to the terminology from Section 3.2 the meta-model can be identified with the information model.

Kurpjuweit describes techniques for integrating different meta-model fragments and further details the system approach for meta-model engineering as introduced in [KW07]. Summarizing the characteristics of the language prescriptions in the EA management approach of the University of St. Gallen, we classify it as shown in Table 3.16.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.16: Language classification for the approach of the University of St. Gallen

3.3.14 Strategic IT management approach of Hanschke

EA management or EA management-related approach	
Name of approach:	Strategic IT management
Issuing organization:	iteratec GmbH (consultancy)
Tool support:	iteraplan
Period of activity:	since 2010
Publications:	[Ha10]
Inner organization:	explicit organization

In [Ha10] Hanschke presents what she calls *strategic IT management*, a collection of best-practices further labeled as a "practical toolkit"[15] for EA management. In the preface of the work Hanschke calls upon the need to have such workable toolkit, as literature on approaches for strategic IT management is abundant, but "remote from real-life practice and do not permit ad-hoc use" [Ha10, page 65]. This reflects the gap that the work of Hanschke seeks to close by describing practice-proven prescriptions and guidelines for the context of EA management. It may be ascribed to this self-image of the approach that the majority of statements contained in [Ha10] are of pragmatic nature, not detailing the intricacies of doing research on a sound and coherent terminological basis. Moreover, the presented approach is complemented with an open source EA management tool, *iteraplan*[16], in which the prescriptions and guidelines, especially concerning the language perspective, are implemented. Central to the approach is

[15] Publisher's description of the book available at `http://www.iteratec.com/download/StrategicITManagement.pdf`, last accessed 04-05-2011.
[16] See `http://www.iteraplan.de/`, last-accessed 04-05-2011.

a framework deconstructing the EA into a set of interrelated sub-architectures as shown in Figure 3.24. For each of these architectures, the strategic IT management toolkit provides best-practices and prescriptions, although the focus of [Ha10] lays on the application architecture and the corresponding management activity of *IT landscape management*.

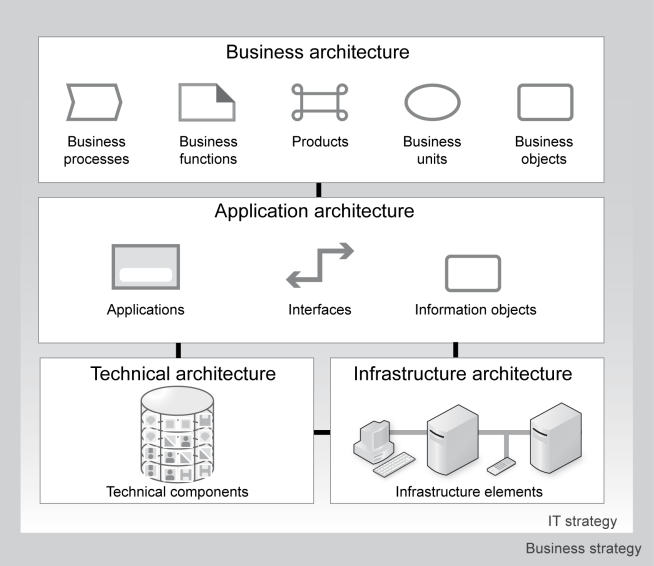


Figure 3.24: EA framework "Best-practice enterprise architecture" of Hanschke [Ha10, page 66]

With its emphasis on providing an executable toolkit, the strategic IT management approach of Hanschke [Ha10] adds profound information on language related aspects. More precisely, all four landscapes are to some degree detailed with information models that outline the relevant concepts on the business, the application, and the infrastructure side as well as concepts for standardizing applications and infrastructure elements. On the business level these concepts target both an external perspective, describing the organization's offerings to its customers e.g. *products*, as well as an internal perspective of service delivery in business processes and via business functions. On the application and the infrastructure layer structural aspects are alluded to, describing the business applications and their interconnections as well as the linkages to hardware and network devices, always with a technical perspective. Projects are essential elements in the information model and may be linked to EA concepts on business, application and technical level. In this sense, the language brings along mechanisms for describing how projects influence the corresponding landscapes on an abstract level, whereas the actual transformations performed by the projects can only be inferred on the application level using application lifecycle information. With the *technology landscape* being a central constituent of the approach's understanding of the EA, different concepts belonging to this landscape may be used to describe the technological standardization of the applications as well as their interfaces and interconnections. This allows to describe concretizations of architectural principles which are in turn devised as part of *technical standardization* (cf. [Ha10, pages 223–242]). In the context of standardization, again aspects of lifecycle modeling are alluded to, making it possible to specify, whether a technology *should be used in the future*, is *in a phase out*, or has already been *retired*.

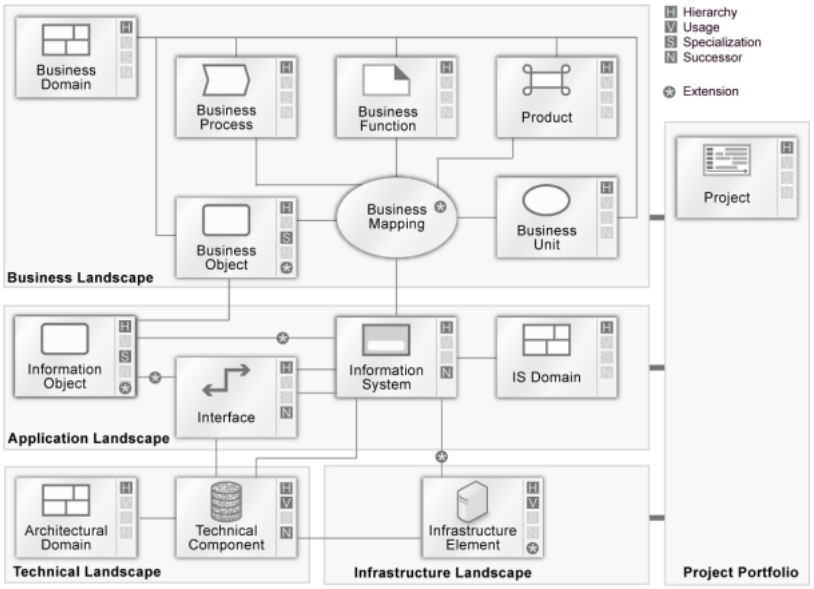


Figure 3.25: EA information model

Special language mechanisms are applied on different concepts to account for a volatility in respect to the taken level-of-detail. Exemplifying this, the approach supplies a language primitive *hierarchization* that denotes that an instance of the corresponding concept may be repeatedly decomposed into sub-instances. Other such primitives are *specialization* to denote that instances of the concept represent types in the EA domain, which can be specialized, and *successor* to describe that an instance of the concept can be superseded by another instance of the same type. The information model of the approach as shown in Figure 3.25[17] indicates hiearchization by the *H*-sign in the top-right corner of the corresponding concept.

The analysis mechanisms of the approach add further concepts for describing typical analysis questions in terms of so-called "analysis pattern" [Ha10, pages 151–157] that may be applied on any architectural level. These patterns do not only describe steps to be taken in analyzing landscapes, but also anchor the corresponding analyses in the affected architectural concepts. Therefore, textual descriptions of the analysis procedures are given and architectural attributes are referenced textually. These prescriptions could in the majority of cases be converted to algorithmic analysis techniques reified in EA-related questions, although the approach only sketches such a procedure. Additionally, the approach describes how these analyses relate to the concept of the "IT goal", which is briefly alluded to by Hanschke [Ha10, pages 23–26]. At this point nevertheless no conceptual linkage between goals and the affected elements of the application landscape is established. In the context of IT landscape documentation methods, the approach of Hanschke [Ha10, pages 206–215] supplies what is called "guidelines for personalization" there. In these guidelines, the need to adapt the modeling language to the specific information demands of the stakeholders (called *beneficiaries* there) is alluded to, and concrete prescriptions on how to analyze the concern-stakeholder-relationships are provided. In this sense, mandatory and *nice-to-have* modeling concepts are distinguished

[17] See `https://www.iteraplan.de/wiki/display/iteraplan/iteratec+Best-Practice+Enterprise+Architecture`, last-accessed 04-05-2011.

and an understanding of the needed data quality is discussed. Against the background of the embracing information model provided by the approach, no dedicated mechanisms for evolving an existing model are considered. Reflecting these characteristics against the background of the language analysis framework discussed in Section 3.2, we can classify the approach as depicted in Table 3.17.

Black-box perspective	business & organization	application & information	infrastructure & data
White-box perspective	business & organization	application & information	infrastructure & data
Strategies & projects	business & organization	application & information	infrastructure & data
Visions & goals	business & organization	application & information	infrastructure & data
Principles & standards	business & organization	application & information	infrastructure & data
Questions & KPIs	business & organization	application & information	infrastructure & data
Configure & adapt	initially	evolutionary	

Table 3.17: Language classification for Strategic IT management

3.4 Summary and conclusion

The fundamental concepts of modeling presented in Section 3.1 establish a profound and sound basis on which we establish an analysis framework for our review of the state-of-the-art in EA modeling. The design perspective of Simon [Si96] is central to this framework as presented in Section 3.2. In Section 3.3 we applied our analysis framework on EA management approaches and their EA modeling languages. Based on the gained insights and our findings in [BS11], we draw the subsequent conclusions:

The **blackbox-perspective** on business and application aspects is well covered by many approaches. This may be a consequence of the importance of service-orientation on both application and process level. A blackbox-perspective on infrastructure is taken by only a few approaches, although the recent virtualization 'rally' may call for a more strict distinction between infrastructure components and their services.

The **whitebox-perspective** is employed by a majority of approaches, almost equally targeting business, application, and infrastructure aspects. This can be explained with the importance of documentation in most EA management approaches [BS11]. The equal coverage on the three layers further objects the often raised argument of EA management being IT-centric.

Both **strategies & projects** are covered by less than half of the approaches. This means that only a minority of the approaches covers the concepts needed to plan EA transformations.

Both **visions & goals** are made explicit by slightly less than half of the approaches, at least when it comes to business or application level goals. This is surprising, as there is a strong agreement on the need for target states for the architecture, whereas means for rationalizing the target state are scarce.

Both **principles & standards** are only covered by a few approaches with special focus on standards on both application and infrastructure layer. This fact mirrors the strong IS focus of today's EA management approaches as well as the fact that as of today no concise understanding of the role of principles in EA management has developed (cf. [St09]).

Both **questions & KPIs** are addressed by half of the approaches, nearly equally concerning metrics on application and infrastructure level with business level metrics being not only slightly behind. This reflects a situation as discussed by Frank et al. in [Fr08] stating that the architecture-related metrics are decoupled from business relevant ones.

The dimension **configure & adapt** is only covered by about half of the approaches. This is especially surprising as the need for an organization-specific information model has repeatedly been discussed in literature, e.g. in [Bu07b, Ai08a, KW09].

This summary shows that EA modeling is more than one step away from a configurable and adaptable 'lingua franca'. Table 3.18, which builds on our analyses as well as on the work of Winter et al. [Wi10], shows that the Archimate language (cf. Section 3.3.9) provides advanced prescriptions that cover the different aspects necessary for supporting the design activity of EA management well. The approach also supports the initial configuration. Mechanisms to ensure consistency over the configuration are nevertheless not incorporated in the configuration method, but are left to the users of the language. In addition does the Archimate language not strictly separate the different aspects of EA modeling, but interlinks aspects as projects and standards, such that users cannot chose to employ only one of these aspects. Cross-cutting aspects of EA modeling are only supported by a minority of the approaches and even, if support is provided, it remains limited to dedicated architecture layers. This means that instead of providing support for modeling arbitrary standards, the approaches mostly confine themselves to specific standardization concepts for the different architectural layers or even particular architectural concepts on these layers. This nevertheless is not a consequence of the specificity of the standardization concepts per layer, but seems to be an accidental restriction. Our work in [BMS10b, BMS10i] discusses this accidentiality along the content metamodel of TOGAF and outlines how **dispersive** types are helpful to describe cross-cutting aspects in a more concise manner. In Section 6.2 the role of dispersive typing is analyzed in more detail, and dispersive type are applied to reflect cross-cutting aspects. These considerations take place in the context of a comprehensive method for designing EA modeling languages, whose constituents are described in Chapter 6.

Black-box perspective	business & organization (2, 3, 6, 7, 8, 11, 12, 13, 16, 21)	application & information (3, 8, 11, 12, 13, 14, 20)	infrastructure & data (3, 13)
White-box perspective	business & organization (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21)	application & information (1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21)	infrastructure & data (1, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 17, 18, 19, 20, 21)
Strategies & projects	business & organization (5, 8, 11, 12, 13, 20, 21)	application & information (5, 8, 11, 12, 13, 20, 21)	infrastructure & data (5, 8, 11, 13, 21)
Visions & goals	business & organization (2, 5, 6, 7, 9, 11, 13, 19, 20)	application & information (5, 8, 9, 11, 13, 19, 20, 21)	infrastructure & data (5, 9, 13, 19)
Principles & standards	business & organization (8, 9, 13, 18)	application & information (8, 9, 13, 18, 19, 21)	infrastructure & data (8, 9, 13, 17, 18, 19)
Questions & KPIs	business & organization (7, 9, 13, 14, 21)	application & information (7, 8, 9, 11, 13, 14, 19, 21)	infrastructure & data (7, 9, 11, 13, 14, 19, 21)
Configure & adapt	initially (3, 4, 6, 8, 14, 17, 20, 21)	evolutionary (6, 9, 20)	

[1] The Zachman Framework
[2] Architecture of Integrated Information Systems (ARIS)
[3] The Integrated Architecture Framework (IAF) [Wo10]
[4] Enterprise Architecture Planning (EAP) [SH93]
[5] The Generalised Enterprise Reference Architecture and Methodology (GERAM)
[6] Semantic Object Model Approach
[7] Multi-perspective Enterprise Modeling (MEMO)
[8] The Open Group Architecture Framework (TOGAF)
[9] Extended Enterprise Architecture (E2A) [SK04], [Sc06a], [Sc06b], [Sc06c], [Sc06d], [Sc08a]
[10] EA management approach of MIT [Ro03], [WR04], [RB06], [RWR06]
[11] EA management approach of TU Lisbon
[12] Systemic Enterprise Architecture Methodology (SEAM)
[13] Archimate
[14] EA management approach of KTH Stockholm
[15] Finnish Enterprise Architecture Research (FEAR) [HP04], [PH05], [Pu06], [IL08], [LHS08], [VS08], [PNL07], [SHL09], [VSL09]
[16] Methodology for (re)design and (re)engineering organizations (DEMO) [Di05], [Di06a], [LD08], [ED09]
[17] The EA^3 Cube^{TM} [Be05], [Do08], [BG09], [Do09a], [Do09c], [Do09b]
[18] Dynamic Architecture for modelling and development (DYA) [St05], [Wa05], [BS06], [SBB07a], [SBB07b], [Lu09], [SB09], [Br10] [St10a]
[19] EA management approach of Niemann
[20] EA management approach of the University of St. Gallen
[21] Strategic IT management of Hanschke

Table 3.18: Summary of language classifications

We have different ways of solving problems.
» Yes. You ignore them.
How could I possible ignore them? I see the problems. I feel them and I'm trying to make something good out of it.

Dialog of Una and Æon Flux – Æon Flux

CHAPTER 4

Towards a Building Block-Based Method for Designing EA Modeling Languages

The analyses of the state-of-the-art in describing EAs undertaken in Section 3.3 have shown that current EA modeling approaches differ in respect to their focus. In particular, there usually is at least one approach covering a relevant aspect of an EA. We have also shown that no single approach offers comprehensive support for all aspects, although some fairly developed approaches are able to cover large parts of relevant EA aspects. These approaches usually comprise comprehensive EA modeling languages that are intended to be used as a whole. A using organization has to customize the language such languages without guidance on how to perform these customization. This lack of support ascribes to the circumstance that the approach often does not encompass explicit mechanisms for customization, as the analyses revealed. The organization can hence not fully benefit from the prescriptions made, but has to develop own adaptation mechanisms.

In this chapter we develop the *building blocks for EA management solutions* (BEAMS) to close the aforementioned gap. We start with a requirements elicitation in Section 4.1, where we establish quality criteria for EA modeling language. Therefore, we use the general language quality framework of Krogstie [Kr02] that provides six categories for language qualities, so called *appropriatenesses*. In response to the discussed gap, BEAMS proposes a method for developing organization-specific EA modeling languages. We continue our requirements elicitation on the method level from the perspective its prospective users and delineate their expectations in Section 4.2 and apply the design guidelines of Hevner et al. [He04]. This sources help to understand requirements that pertain to the development method. They are further of interest in answering our research question 3 raised in Section 1.1. We lay the groundwork for the method by discussing relevant fundamental concepts from literature in Section 4.3.

4.1 Quality criteria of EA modeling languages

Many of today's EA management approaches have, as shown in Section 3.3, a specific perspective on the goals that can be attained and on the problems that can be addressed by applying the approach. This perspective reflects the approach's understanding of EA and EA management. Each approach satisfies *by design* a specific set of EA management-related user quality criteria, while there is no general agreement on such criteria among the different approaches. This mirrors the diverse expectations of the users regarding EA management. According to the findings of Section 3.3, any EA modeling language must be

1. appropriate for the *modeled domain* (the EA),
2. usable for both *model creators* and *model users* (the EA stakeholders), as well as
3. accessible for tool-supported EA model creation.

Especially the coverage of the different structural as well as cross-cutting language aspects shows what users expect from an EA modeling language. In the following, we use different sources for eliciting the quality criteria. One relevant source is GERAM, which has been analyzed in Section 3.3.3. According to its focus on a generic reference for EA management, it supplies a set of general principles, which are delineated by Bernus and Nemes in [BN94] and by Bernus et al. in [BNS03, pages 14–15]. The following principles lay the basis for defining the quality criteria[1]:

GE1 "best treatment of the enterprise scope": the EA management function must cover all activities in designing, operating, and maintaining the enterprise and its architecture also with a modeling language,

GE2 "provision of a consistent modeling environment": the EA management function must offer an expandable set of related modeling views based on a common information model, going from high level enterprise concepts to executable code,

GE3 "existence of a detailed methodology": the EA management function must provide a technically correct and consistent methodology both usable and understandable to the target communities, supporting specialization to particular contexts,

GE4 "adoption of good engineering practice": the EA management function must utilize reusable building blocks to encapsulate complex details of EA management in order to foster sustainability of the management activities, and

GE5 "provision of a unifying perspective": the EA management function cover the enterprise from strategies and products to processes and development activities, as already addressed by related management activities.

The principle **GE4** is special in its nature and does not pertain to the EA management function in operation, but to the development of such function. In this respect, the principle does not contribute to the elicitation of the quality criteria for EA modeling languages, but is incorporated in the best-practice-based development method. Above principles are further detailed by the ISO Standard 15704:2000 [In99] as well as by Noran in [No03]. In revisiting the principles as part of our elicitation of quality criteria, we use the additional details.

[1] In line with the terminology of GERAM, we talk of modeling instead of describing EAs in **GE1-GE5**.

In addition, we derive quality criteria from a survey on the state-of-the-art in EA management tools conducted by us [Ma08] at the chair sebis at Technische Universität München in 2008. As reference for analyzing the EA management tools, our survey establishes two sets of *scenarios*, namely "scenarios for analyzing specific functionality" and "scenarios for analyzing EA management support" [Ma08, page 39]. Former scenarios describe functional quality criteria for EA management tools, whereas latter scenarios delineate typical activities of EA management. These activities are described on a level of detail where not only the core concerns are outlined in an abstract fashion, but also concrete management-relevant questions are given. These questions are further complemented with visualizations that exemplify possible answering techniques. The eighteen scenarios reflect the EA management quality criteria as experienced in the chair's EA management projects during a six years period, but further leverage the knowledge of 30 industry partners participating in the survey, of which 25 sent at least one leading EA representative to on-site workshops. These workshops are part of a larger method that we [Ma08] applied in developing the scenarios. This five step method ensures that the scenarios reflect a broad variety of expectations related to EA management in practice:

1. The survey team derives the initial descriptions of the scenarios based on project experiences and EA management-relevant literature.
2. Representatives from the industry partners and the survey team conduct on-site workshops to discuss the initial scenario descriptions.
3. The survey team revises the scenario descriptions based on the workshops' feedback. This step includes re-scoping of scenarios and introducing new scenarios.
4. Representatives from the industry partners conduct remote reviews of both the scenarios and provide feedback.
5. The survey team incorporates the feedback into the final scenario descriptions.

Following above method we elicited eighteen scenarios, see Table 4.1.

Specific functionality		EA management support	
SF1	Importing, editing, and validating model data	**ES1**	Landscape management
SF2	Creating visualizations of the EA[1]	**ES2**	Demand management
SF3	Interacting with and editing of EA visualizations[1]	**ES3**	Project portfolio management
SF4	Annotating visualizations with certain aspects	**ES4**	Synchronization management
SF5	Supporting leightweight access	**ES5**	Strategies and goals management
SF6	Editing model data using an external editor	**ES6**	Business object management
SF7	Adapting the information model	**ES7**	SOA transformation
SF8	Handling large scale EAs[1]	**ES8**	IT architecture management
SF9	Supporting multiple users and collaborative work	**ES9**	Infrastructure management

[1] In [Ma08] the scenarios' titles refer to "application landscapes", although entire EAs are considered.

Table 4.1: EA management tool scenarios of Matthes et al. [Ma08]

The scenarios **SF1** "importing, editing, and validating model data", **SF6** "editing data using an external editor", and **SF8** "handling large scale EAs" are only of minor interest for our subsequent elicitation of quality criteria. This owes to the fact that these scenarios aim at tool-specific aspects, which are not specific to the EA modeling language.

From 2006 to 2008, we conducted a survey [Bu09a] to analyze the state-of-the-art of the EA management practice. This survey, whose results are based on 22 interviews with enterprise architects and on an online questionnaire filled by 31 enterprise architects, raises ten key topics of EA management practice. We apply a rigorous method for extracting the findings from the interviews, building on interview transcription and iterative text comprehension taking into account interviewee's feedback. The identified topics provide another valuable source to understand, what EA management is meant to be:

PT1 "searching for a definition of EA management" shows the diversity of definitions of the subject and its terms

PT2 "the E in EA management" discusses the holistic nature of EA management

PT3 "governance and EA management" relates EA management to supporting and existing governance activities

PT4 "EA frameworks offer only minor assistance" delineates the fallibilities of current EA frameworks

PT5 "the information gathering process" inquires into the difficulties of gathering the relevant information

PT6 "visualizations as a communication basis" shows how visualizations are employed in EA management

PT7 "on the importance of communication" discusses the communicative nature of EA management and the enterprise architects

PT8 "tools for EA management" delineates the importance of tool support and its current shortcomings

PT9 "the role of SOA" relates EA management to the topic of SOA

PT10 "metrics and KPIs" discusses the usability of quantitative measures to support EA management

Practitioners' topic **PT4** "EA frameworks offer only minor assistance" does not directly support a specific quality criterion of the language. This on the one hand can be considered a consequence of the nature of the topic, being a mere statement. On the other hand, the statement itself can be regarded the rationale behind the investigations in this thesis, leading to a different approach towards EA management and EA modeling languages.

In the following, we derive a set of requirements for EA modeling languages. To structure and guide the derivation, we use the quality framework for modeling languages presented by Krogstie in [Kr02]. This framework introduces several *qualities* of a model specification:

Physical quality is concerned with internalizability and externalizability of domain knowledge and corresponds to the *representation* quality of Stachowiak [St73]. Both qualities demand that the model covers aspects of the modeled domain in an unambiguous way.

Syntactic quality describes that all model statements are correct with respect to the syntactical rules of the modeling language.

Semantic quality is concerned with semantical completeness. This means that on the one hand the meanings of all parts of the model are described and that on the other hand no conflicting meanings are provided.

Pragmatic quality aims at mechanisms, e.g. indexes or glossaries, that improve understandability of the modeling constructs.

Tool quality is concerned with tools to create and maintain the model as well as with the question, how good a model creation and utilization can be supported with adequate tools.

Perceived semantic quality describes that the model users should be able to interpret the model and to understand the meaning of the model as a whole.

Social quality is concerned with socialization of the model, i.e., the adoption of the model in the intended user community as a means of description and communication.

In line with Krogstie [Kr02] a model results from the application of a modeling language to a subject from the specific domain. Via syntactic quality the modeling language influences the other qualities, that can be achieved. From this fact Krogstie derives the notion of *appropriateness*, by which he relates the modeling language to the recipients of the corresponding qualities. Figure 4.1 depicts Krogstie's quality framework, which provides a basis for understanding quality criteria for an organization-specific EA modeling language.

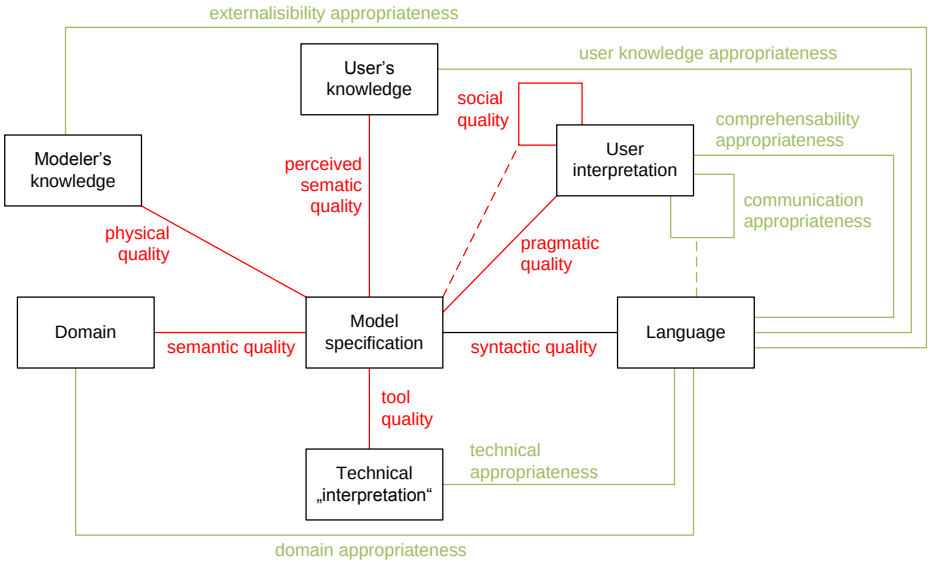


Figure 4.1: Quality framework for modeling languages adapted from Krogstie [Kr02]

We must distinguish two types of human actors relating to a model and thereby also to the corresponding language. Firstly, there are the *modelers* or model creators, who use the language to externalize their knowledge about the domain. Secondly, there are the *users* of the models, that want to understand the information made available in the model. As several

of the appropriatenesses discussed in the following are concerned with the human actors, it should further be noted that their understanding of the models is influenced by their specific knowledge on the domain, i.e., the *base level* in terms of Tanaka and Taylor [TT91].

4.1.1 Communication appropriateness

Communication appropriateness aims at the interpersonal qualities related to a modeling language and to the *communicative function* of the language as discussed by Becker et al. [Be03, pages 8–10]. We understand communication appropriateness as the fact that the modeling language promotes user agreement on the contents of the model, based on an already established understanding of the modeled domain. For the context of EA modeling languages communication appropriateness also accounts for 'political' implications of the EA descriptions. Such implications can e.g. arise, when information about a planned restructuring of the organization is promulgated early.

The state-of-the-art analysis reveals that many of the analyzed approaches acknowledge that EA management is a collaborative endeavor. The corresponding key quality of 'good' EA management is to communicate the 'right information to the right people at the right time'. From this tryptichon, especially the latter two properties are relevant in respect to communication appropriateness. A quotation given in [Bu09a, page 21] highlights the implications of right and timely communication: "It is important and difficult to communicate the results of architectural planning [to application owners] who loose the business applications, they are responsible for". The scenario "supporting lightweight access" presented by us in [Ma08, pages 48–49] details what communication appropriateness means, raising the requirement to restrict user access on information or limit it to read-only access. Noran describes in [No03, pages 122–123] a related quality criterion for *enterprise modeling languages*, demanding that any language explicitly targets an intended audience. The scenario "supporting multiple users and collaborative work" [Ma08, pages 51–53] details on the information supplier role. This entails assigning responsibilities for information supply regarding specific areas-of-interest in the EA. In [Bu09h] we discuss the difficulties that arise, when responsibilities for information maintenance remain undecided. In response, we present a bottom-up approach to address the difficulties. The fact that EA descriptions get outdated fast, as the analyses of Aier et al. [Ai09a] show, calls for clearly defined maintenance responsibilities.

Quality criterion Cm1: Rights and responsibilities assignment
An EA modeling language must supply means for assigning access rights to parts of the EA description as well as to define maintenance responsibilities for specific areas-of-interest in the EA.

The creation and maintenance of EA models are collaborative tasks in which a large number of users residing in a highly distributed environment contributes information. This results in a continuous stream of changes and updates that are applied to the EA description. In his discussions on the *Generalised Enterprise Reference Architecture* (GERA), Noran [No03, pages 81–82] puts special emphasis on the *life history* pertaining to all architecture elements. The life history of an element thereby describes the entirety of change steps that the element has undergone. The approaches analyzed in Section 3.3 do highlight the importance of complying with regulations, as e.g. with the *Sarbanes-Oxley Act* (SOX) [1002], which demands

traceability especially with respect to decisions. Against the background that EA management provides holistic decision support targeting the enterprise on many different levels, the need for compliance entails the need to trace changes especially regarding planned states of the EA. This understanding is further backed in the scenario "supporting multiple users and collaborative work" [Ma08, pages 51–53], describing the importance to trace changes to the EA descriptions. As part of *historization* also the scenario "landscape management" [Ma08, pages 55–58] discusses the need to know, which changes have been applied to architecture elements. We also discuss this aspect in [Bu09h], where we highlight versioning of EA elements as a key benefit of a wiki-based approach to EA documentation. Also Aier et al. discuss in [Ai09a] the importance to retain change traces in order to understand, how 'old' parts of the EA description are. This information is used to identify maintenance demands and helps to keep the EA descriptions timely. We discuss a similar topic in [Bu09a, pages 15–18] for the aspect of "information gathering", where meta-information about the last-modification and the last-modifier is considered an important aspect of EA descriptions.

Quality criterion Cm2: Traceability of changes
An EA modeling language must supply means for documenting change traces for the elements in the EA description to document who has edited what and when.

4.1.2 Comprehensibility appropriateness

Comprehensibility appropriateness relates the modeling language with the *social actors* that interpret models created in the language. Krogstie describes in [Kr02] the essence of comprehensibility as the ability of the language's intended audience to understand all statements possible in the language. In this vein, comprehensibility relates to the notion *model pragmatics* of Stachowiak (cf. Section 3.1.1).

The analysis of the state-of-the-art reveals the importance of perspectives in EA modeling. All analyzed approaches do not convey only a single way of looking at the EA, but delineate a spectrum of possible **viewpoints** in the sense of the IEEE Recommended Practice 1471:2000 (cf. Section 3.1.3). These viewpoint differ from approach to approach, which means that there exist approaches, which cover disjoint viewpoints on the EA. As a viewpoint is a stakeholder-specific conceptualization, the plurality of viewpoints reflects the multi-disciplinarity of EA management as well as of its stakeholders with respect to their background (cf. Lucke et al. [LKL10]). We discuss in [Bu09a, pages 18–19] the importance of different visualizations grounded in a "common information basis". The multi-perspectivity in EA management is a central topic explicitly accounted for by Noran in [No03, page 95] and reverberating through the EA management scenarios presented by us in [Ma08]. All scenarios therein target a single enterprise with a consistent architecture, which is approached and visualized from different perspectives. The particular perspectives reflect the viewpoints of the stakeholders as involved in the specific scenario. Such viewpoint omits certain architecture elements, which are not necessary to understand a specific context. We also discuss multi-perspectivity as an important property of EA descriptions in [Bu09h], highlighting that stakeholder-specific viewpoints are required in the context of comprehensive EA descriptions.

Quality criterion Cp1: Multi-perspectivity support
An EA modeling language must support different stakeholder-specific perspectives on the overall enterprise.

The discipline of EA management is still a maturing discipline, which is among others reflected in the terminology used to describe the management subject EA. This terminology can partially build on the existing work in related disciplines, e.g. software engineering or business process modeling. It further has, as shown in the state-of-the-art analysis, to establish additional terms to cover aspects originating on the EA level. The analysis of Schönherr [Sc08b] shows the terminological plurality in the field along a simple, but fundamental example. He points out that there is no commonly accepted definition of the term EA itself. A similar discussion on the practitioners' perspective on EA can be found in [Bu09a, pages 10–11], showing that practitioners experience the need for a consistent terminology in the field. Revisiting our analysis in [BS11] we diagnose that the situation is less critical, when it comes to EA concepts. Many of the approaches discuss the importance of having a *glossary* of terms, thereby promoting a common understanding, on which the users can call, when they utilize the EA modeling language. The glossary supplies the semantics for the concepts of the EA modeling language by providing standardized *predicators* (cf. Kamlah and Lorenzen [KL67, pages 26–29]) for the language's underlying conceptualization. In [No03, page 176] Noran explicitly accounts for the construct of the glossary, demanding that any EA modeling language targeting the enterprise supplies at least textual descriptions for the language concepts.

Quality criterion Cp2: Glossary support
An EA modeling language must supply backing glossary of terms textually defining the relevant concepts.

4.1.3 Domain appropriateness

Domain appropriateness relates the modeling language with the domain to be represented in the sense of the *representation characteristic* outlined by Stachowiak (cf. Section 3.1.1). In line with Krogstie [Kr02], we understand domain appropriateness as the fact that in the represented domain no statements can be made that cannot be expressed in the language. For our elicitation of quality criteria, domain appropriateness builds on our general understanding of the EA domain described in the analysis framework in Section 3.2.

The analysis of the state-of-the-art shows that EA management is an activity operating at the junction of business and IT topics. The analyzed approaches understand EA management as the logical continuation of what Henderson and Venkatraman call *strategic alignment* in [HV93]. Therein, the authors advocate for considering the transformation of the organization in an integrated manner both from a business and an IT perspective. The alignment nature of EA management, linking business and IT, is also reflected in the scenarios of our tool survey [Ma08]. These scenarios range from "business object management" [Ma08, page 68] concerned with the business-relevant object types and their processing along the process chains to "infrastructure management" [Ma08, pages 75–76], in which the technical infrastructure needed to support the enterprise operations is considered. In [Bu09a, page 10] we discuss the "holistic" nature of EA management, covering both business and IT aspects. This embracing nature of the management subject pertains to the modeling languages that accordingly have to cover all the different aspects. A discussion undertaken by Noran in [No03, page 95] partially mitigates this criterion. From the GERA perspective he demands that any language for modeling enterprises is complete with respect to the corresponding modeling framework, i.e., its corresponding conceptual basis.

Quality criterion Do1: Business and IT coverage
An EA modeling language must supply concepts for describing both business and IT elements of an enterprise, as well as the interconnecting entities.

Today's enterprises find themselves confronted with a volatile regulatory and economic environment (cf. Wagter et al. [Wa05]), to whose changes the organization as a whole has to react. Henderson and Venkatraman describe in [HV93] the need to transform both the organizational structures and processes as well as the underlying IT support. These enterprise transformation processes are directed towards one or more enterprise goals. In [Ma08, pages 66–68] we discuss "stragies and goals management" as a typical scenario of EA management aiming at establishing and managing these goals, which provide transformation guidance. In addition to the transformations, the enterprise undergoes continuous change resulting from ongoing maintenance activities. We exemplify one of these activities in the scenario "IT architecture management" in [Ma08, pages 72–74] with a migration of a database management system. Such activities not necessarily entail EA-level changes, but can interfere with ongoing transformation processes. We discuss these interdependencies in the scenario "synchronization management" [Ma08, pages 61–66]. EA management is a coordinating management function as we discuss in [BMS09c] and in [Bu09a, pages 13–14]. Therefore, the EA modeling language must provide means for modeling what Murer et al. call in [MWF08] the *managed evolution* of the organization. This in particular means that the modeling language must support the distinction between the current state, called "as-is state" by Noran in [No03, pages 185–186], and planned states of the EA, which delineate the EA roadmap [Bu09f]. These states are discussed by Matthes et al. in [Ma08, pages 55–58] as part of the scenario "landscape management" as transformation snapshots in the sense of Noran [No03, pages 68–69]. Noran further discusses the *transformation processes* consisting of the activities performed to evolve the enterprise. We also discuss this topic in [Ma08, pages 58–60] with the scenarios "demand management" and "project portfolio management", which link the projects to the architectural changes.

Quality criterion Do2: Transformation coverage
An EA modeling language must supply concepts for describing the evolution and transformation of the EA, relating the changes to the underlying demands and projects, and accounting for the temporal frame of architectural elements.

The design space for the EA has become increasingly diverse through different developments. New technologies, new paradigms, but also novel business models open new options for shaping the organization in response to existing or changing demands. This plurality of options nevertheless does not come without costs, but leads to an increased heterogeneity in the organization, i.e., a situation, where similar functionalities, services, or capabilities are provided in highly diverse manners. According to Murer et al. [MWF08] a management function that controls the organization's evolution must counter this heterogeneity with guidelines that limit the design space. The analysis of the state-of-the-art shows that *principles* and *standards* for evolving and developing the EA are discussed by one of two EA management approaches. Aitken describes in [Ai10] a related contribution to the field and presents an EA design pattern specifically targeting the definition of design principles. The establishment and management of guidelines for the EA is also discussed in the scenario "IT architecture management" presented by us in [Ma08, pages 72–74]. Dedicated EA concepts are used in the scenario to reflect prescriptions on how to constitute a business application in an enterprise.

This information is used in a twofold way, firstly for assessing the homogeneity of the current state of the EA, and secondly for constraining the admissible architecture evolutions against the background of their changes in respect to architecture standardization. Noran introduces in [No03, pages 185–186] "model-based control" and demands that any language for modeling enterprises must be usable to control via guidelines and prescriptions. On a more concrete level the topic of SOA, as we discuss in [Bu09a, pages 23–26], is an instantiation of specific guidelines that are pertinent to the field of EA management.

Quality criterion Do3: Guidelines coverage
An EA modeling language must supply concepts for designating certain architectural structures as intended or desired or as unwanted.

4.1.4 Externalizibility appropriateness

Krogstie explains in [Kr02] externalizibility appropriateness as a relationship between the language and the model creator's knowledge. Appropriateness means that the language is suitable to express all statements in the *relevant* knowledge of the model creator. In order to derive quality criteria from this demand, we have to understand the meaning *relevance*. In the context of EA descriptions relevant knowledge is confined to knowledge that is needed to address the problems that an EA stakeholder has risen. In line with the discussions of Aier et al. [ARW08b] many different problems are considered in EA management. General requirement **G3** of GERAM [BN94, pages 14–15] states that the problem-specific perspectives have to integrate into a "technically correct whole", which is further easy to use for the model creators. For the subsequent quality criteria, we refine the general understanding of EA problems along two dimensions, namely 'where' and 'what'. EA problems are compositions of a specific area-of-interest (**concern**) and a particular **goal** to be achieved in this area.

The majority of the EA management approaches analyzed in Section 3.3 states that the EA forms a highly complex management subject. As Rhodes et al. delineate in [RRN09] organizations tend to underestimate the complexity of the corresponding EA management endeavors. These organizations do not focus on dedicated parts of the EA and often end up with "giant models" [Bu07b]. Such models are usually hard to use and form a well-known *anti-pattern* of EA management, as documented by us in [Bu09d]. If the EA modeling language fails to provide the concepts needed to cover the relevant parts of the EA, model creators tend to develop 'workarounds' that deprive the EA description of its usability (cf. Bauer [Ba10b] and Matthes et al. in [Ma08, pages 49–50], "adapting the information model"). In this sense, completeness with respect to the concern, as demanded by Noran in [No03, page 95], is a prerequisite for an adequate scoping of the EA model. Such scoping is according to Lucke et al. [LKL10, page 8] a critical issue of EA management. Only a minority of the EA management approaches analyzed in 3.3 explicitly accounts for the question of scoping the EA description in respect to the stakeholders' concerns. Information model adaptation is a key quality of a specific type of EA management tools, the so-called *metamodel driven* tools [Ma08, page 344].

Quality criterion Ex1: Concern coverage
An EA modeling language must provide mechanisms to explicitly account for the concerns of its stakeholders, keeping the EA descriptions simple but sufficient.

The analysis of the state-of-the-art in EA management shows that explicit goal setting is a topic for a majority of the prominent EA management approaches. Goals are EA quality criteria raised by the stakeholders of the management function and reflect a broad variety of possible objectives that the management endeavor can target. Our work [Ma08] on practice-related EA management scenarios shows the range of possible goals from reducing application landscape complexity to optimizing business object processing. The scenario "strategies and goals management" [Ma08, pages 66–68] accounts for this plurality and describes modeling techniques that can be used to make explicit, which part of the EA is affected by which goal. Similar findings can be derived from the analysis undertaken by Aier et al. in [ARW08b], which describes the variety of different goals addressed by means of EA management. In [Bu09a, pages 26–29] we outline the plurality of practice-relevant goals and highlight the practitioners' need for concrete measures, metrics and KPIs to reflect these goals [LS07, LS08]. In [BMS10b] and [BMS10i] we describe how specific concepts in EA information models can be used to concretize goals to **questions** in line with the *goal-question-metric* approach of Basili et al. [BCR94]. A question is reified by concept that can be added to an arbitrary architecture element. With questions, stakeholders can externalize their understanding of a goal in terms of architectural qualities. Explicit concretized goals are the basis for "model-based control" in the sense of Noran [No03, pages 185–186].

Quality criterion Ex2: Goal coverage
An EA modeling language must provide mechanisms to explicitly account for the goals of its users, and for externalizing the goals via questions.

4.1.5 Technical appropriateness

Technical appropriateness relates the language with *technical actors* (tools) that are used to work with the language. As Krogstie discusses in [Kr02] such appropriateness entails *formality*, *analyzability*, and *executability*. The latter property refers to the *operational semantics* of the language (see Chapter 5 for in-depth discussions on the different types of semantics) and is only of limited interest for modeling languages. As we discuss in [Bu09a, pages 21–23], analyses of the EA are a topic of high practical relevance. Analyzibility builds on a formal semantics of the language. The importance of tools providing a "consistent environment" for modeling enterprises is also delineated by criterion **(GE2)** of GERAM [BN94, pages 14–15].

According to the state-of-the-art, EA visualizations are highly relevant means of communication in all EA management approaches. Visualizations can be used to present the complex structures and dependencies making up the architecture of an enterprise to the stakeholders. As we show in [Bu09a], practitioners make heavy use of various diagrams and visualizations as a basis for communication. The creation of such visualizations is, as discussed by Wittenburg in [Wi07] as well as by us in [Bu07a], a time-consuming and error-prone task, especially when executed manually. Lucke et al. discuss in [LKL10] that tool support for creating visualizations is beneficial for successful EA management. Our survey [Ma08, pages 41–48] formulates three scenarios targeting the visualization of EAs. The first scenario "creating visualizations of the application landscape" describes the general variability of the types of visualizations that a tool should support. Scenario "annotating visualizations with certain aspects" adds graphi-

cal annotations, e.g. 'color-coding', to the basic visualizations to indicate certain architectural properties. The third scenario "interacting with and editing of visualizations of the application landscape" discusses how visualizations can be changed to facilitate communication.

Quality criterion Te1: Visualization generation
An EA modeling language must be designed in a way that graphical models describing EAs can be generated out of non-graphical descriptions.

Analyses play an important role as supportive techniques for design activities. Different EA management approaches in the state-of-the-art analysis discuss ways of to analyze the EA or parts thereof. As Lankes describes in [La08b] analyses are used mainly for two different reasons. Firstly, they realize the *check*-phase of a typical management cycle (cf. Deming [De82] or Shewart [Sh86]), i.e., measure the achievement of goals. Secondly, analyses are applied to identify potentials for improving the EA and for discovering areas that could profit from a transformation. Our findings in [BMS09a] show the broad variety of types of analyses applied to EA descriptions, ranging from *pattern-based* qualitative analyses to *metrics-based* quantitative analyses. Especially analyses of the latter type are according to [Bu09a] considered a relevant, but yet still developing topic in the field of EA management. In the scenarios "interacting with and editing of visualizations of the application landscape" and "annotating visualization with certain aspects" described by us in [Ma08, pages 44–48], we discuss the topic of analyses from a visualization-centric perspective. Former scenario introduces the idea of the graphical *impact analysis*, an ad-hoc analysis method, whereas latter scenario discusses how both quantitative and qualitative analysis results can be displayed. Following our argumentation in [BMS10i] as well as the one of Noran in [No03, pages 185–186], analyses are an important aspect in EA management, which has to be adequately reflected in the EA description.

Quality criterion Te2: Analysis coverage
An EA modeling language must supply concepts for anchoring qualitative and quantitative analyses on architectural elements, supporting automated analyses.

4.1.6 User knowledge appropriateness

User knowledge appropriateness roots in the relationship between the language and the knowledge of its users. In line with Krogstie [Kr02] this means that all language statements made by different users can be intercalated in the knowledge of any other user. This forms an abstract quality requiring that statements in the language remain *interpretable* against the evolving knowledge backgrounds of the users. GERAM further raises in [BN94, pages 14–15] the quality criterion that interpretation must be easy to perform.

User acceptance is crucial for the adoption of EA models as means for EA management. Wittenburg discusses this in [Wi07] from a visualization-centric point of view. He outlines that in order to promote the use EA models, these models should use familiar notations, i.e., symbols that are well-known to the "target audience" (cf. Noran in [No03, pages 122–123]). The analysis of the state-of-the-art show that the topic of notations is only discussed by a minority of the EA management approaches. Some of these introduce dedicated symbols for

specific EA-relevant concepts. Other approaches allude to the need to align the graphical notation of EA models with the notations of standards, as the UML. The adequate selection of the standard to adopt can according to us [Bu09a, pages 18–19] be considered relevant for user acceptance. The scenario "adapting the information model" presented in [Ma08, pages 49–50] raises a particular quality criterion pointing in the same direction. According to this scenario an EA management tool must support to assign a specific standard symbolic representation to an EA concept. This is a means to familiarize the user with the underlying EA modeling language. The question of familiarization pertains to the "layering principle" of Wittenburg [Wi07, pages 83–84]. This principle describes that EA visualizations should build on a slowly-changing *base-map* that facilitates quick recognition of architectural elements. More volatile information is added on *layers*. Different layers can be used to display the same information using different ways of representation. The layering principle allows to supply users with only the specific information, which fits their specific knowledge background.

Quality criterion Us1: Representational fit
An EA modeling language must supply means to adapt its notation to fit the knowledge of its intended users, especially adopting familiar and quickly recognizable symbolic representations.

The analysis of the state-of-the-art shows that especially more comprehensive EA management approaches discuss the evolvability of the EA management function itself. In particular, the approaches state that the relevant goals and concerns of the management function change in the course of managing the EA. For the aspect of the goals, this fact is discussed in the scenario "SOA transformation" in [Ma08, pages 69–72]. This scenario describes an organization, which has already implemented EA management to among others pursue standardization. This organization decides to adopt the paradigm of service orientation. This raises a new goal for their EA management function. In [Bu10d] we discuss three evolutions of an EA concern targeting enterprise-wide information access. As the organization's EA management evolves, the corresponding concerns become increasingly fine-grained and the EA modeling language has to adapt. The scenario "adapting the information model" in [Ma08, pages 49–50] discusses such language adaptations. A similar consideration can be found in [No03, pages 185–186], where Noran describes that "modularity and extensibility with respect to the meta-model" is a key property of any EA modeling language. Our discussions in [BMS10f] focus on the evolutionary nature of EA descriptions and discuss the need to prepare the descriptions as well as their underlying languages to inevitable adaptations. As Aier et al. state in [Ai08a, page 559] evolutions must not be "disruptive", but have to be supported via language mechanisms that help to guide language users with respect to their developing understanding.

Quality criterion Us2: Sustainability
An EA modeling language must be designed in a way that it supports future adaptations with respect to the addressed EA management-relevant goals and covered concerns.

The *communicative function* (cf. Becker et al. [Be03]) of an EA modeling language has implications with respect to knowledge appropriateness. The EA is a highly diverse subject and targets concepts originating from a broad variety of domains and disciplines. Therefore, the

corresponding language has to bring together many different terms. The analysis of Schönherr [Sc08b] shows that this terminologically challenging background has lead to a diverse image of the EA and its constituents. According to our argumentation in [Bu09h] terminological differences are the consequence of the stakeholders' differing *base levels* (cf. Tanaka and Taylor in [TT91]) with respect to describing the EA. This means that stakeholders tend to use terms that are according to their knowledge neither very general nor very specific. They instinctively adopt an intermediary base level for communication. With the different backgrounds of the involved stakeholders, *boundary objects* (cf. Star and Griesemer [SG89]), i.e., objects relevant to two backgrounds or perspectives, are at risk to be denoted by synonyms. The application of a standardized terminology nevertheless does not necessarily have positive implications with respect to understandability. As Aier et al. discuss in [Ai08a, page 559], this can cause "terminological disruption", which means that the terminology is perceived as 'alien' by the stakeholders. We discuss in [Bu09a, pages 18–20] that an enterprise architect has to "translate" between different stakeholders. Noran [No03, pages 122–123] regards an accepted and clear terminology being "appropriate to the aspect modeled" a necessary prerequisite for successful EA management.

Quality criterion Us3: Terminological fit
An EA modeling language must supply means to adapt its terminology to fit the linguistic communities of its users, especially with respect to boundary objects.

4.2 Requirements for the development method for EA modeling languages

Different approaches to develop an EA modeling language can, as already discussed in Section 1.1, be taken, namely the *greenfield approach*, the *customization approach*, and the *integration approach*. With respect to these approach, the design method has to provide different kinds of prescriptions. A method supporting

greenfield development supplies a process model for eliciting the language's concepts together with a meta-modeling technique for documenting the concepts. Further, such method must provide techniques to ensure consistency and avoid terminological disruptions as well as ambiguities,

development by customization supplies an embracing modeling language built on an integrated terminology. Supplementary techniques help to determine which concepts can be omitted with respect to specific EA management goals and EA concerns, while ensuring overall consistency of the modeling language.

development by integration supplies an organized collection of modeling language fragments complemented with techniques for selecting the fragments suitable for certain EA management goals and EA concerns. Moreover, the method must supply techniques for consistently integrating the chosen fragments.

All three approaches are grounded in techniques for meta-modeling, i.e., for describing the EA modeling language itself. In the light of GERAM requirement GE4 [BNS03, pages 14–15], the integration and the customization approach can be considered a good basis for a method

for developing EA modeling languages. Both approaches can leverage practice-proven knowledge. These approaches, while being distinguishable in respect to the 'direction' from which the EA modeling language is approached, require closely related techniques. In particular both approaches need techniques for selecting the appropriate concepts and for keeping the developed EA modeling language consistent. The major difference therefore reduces to the question, if an embracing model is adapted by omission and restriction or if model fragments are combined and integrated. Therefore, the requirements for the development method can be elicited regardless of the approach taken. Following Section 4.2.1 presents the requirements links them to the "guidelines of modeling" delineated by Becker et al. in [BRS95] as well as Schütte and Rotthowe in [SR98]. In Section 4.2.2 we revisit the general design guidelines formulated by Hevner et al. in [He04].

4.2.1 Domain-specific method requirements

The domain of developing organization-specific EA modeling languages raises a set of requirements to the development method. As a first step, we designate the intended user group for such method. In line with the discussions in TOGAF [Th09a], Op't Landt et al. [La09c, pages 114–119], and with the statements given by the practitioners [Bu09a], we assume that **enterprise architects** are the user group skilled for using the method. In particular, enterprise architects are expected to be

skilled communicators, i.e., are able to gather requirements from a broad range of EA stakeholders and are capable of understanding as well as translating their terminologies,

skilled modelers, i.e., are able to generalize the requirements as well as the relevant architecture elements in the enterprise towards a conceptual level, and

able to structure the enterprise into architectural concerns and goals.

Above does not necessarily mean that we also expect the enterprise architects to be the stakeholders rising the concerns and goals. The architects act as agents on behalf of these stakeholders and develop EA modeling languages that address the stated EA management problems. The role of the agents can further be taken by (external) consultants. Against this background, we assume for the subsequent requirements that at least one agent, either an enterprise architect or an EA consultant, with the outlined skillset is available to use the method.

Requirement MU1: Being easy to use
The method as well as the corresponding techniques and models must be easy to use for architects, who have familiarized themselves with the underlying terminology and conceptualizations.

In the light of the inherent complexity of the management subject, we do not assume that the method and its enclosed language fragments are understandable for the users without prior familiarization. Structuring principles of the method, as the separation of goal and concern, are to be learned by the users such that they can leverage the provided prescriptions. After an initial understanding of these principles is acquired, the method as well as the language fragments must be easy to understand and to use. Therefore, especially a consistent and uniform

way of presenting the language fragments is necessary, fostering overall *clarity* of the presentation. Latter aligns to the corresponding principle of modeling [SR98, pages 248–249], which states that any useful model has to provide an adequate level of intuitive understandability.

Requirement MU2: Providing practice-proven prescriptions
The method must supply the users with practice-proven design prescriptions as well as problem-specific fragments for EA modeling languages, facilitating their re-use.

This requirement particularly means that the prescriptions made by the method as well as the provided language fragments must have proven themselves as solutions to problems of practical relevance. Thereby, a two-faceted view is taken: firstly, the solution fragments described must reflect ones not theoretically invented but abstracted for adoption from different practice cases. Secondly, the corresponding problem must have occurred repeatedly and must have shown to be of relevance for practitioners. This links to the fundamental principle of *construction relevance*, which demands that any specific model has to be "right" in respect to the thereby addressed goal and problem [SR98, pages 245–246].

Requirement MU3: Being systematically extensible
The method and its underlying organized library of language fragments must be adaptable and extensible in respect to new practice-proven prescriptions.

The ever changing environment, in which the discipline of EA management acts, is subject to influences from economic, regulatory, and technical factors (see e.g. [La05, Wa05, RWR06]). These factors make the possible scope of the EA modeling languages a 'moving target'. This not necessarily renders the method for developing EA modeling languages infeasible, but makes it subject to ongoing adaptation and extension. Therefore, the method must support adaptation of the supplied language fragments as well as the contribution of new fragments without disrupting the underlying structure of the method. The extension method must supply means to ensure usability of the overall development method and its underlying organized library. This in turn can be regarded as implication of the principle of *systematic design* [SR98, page 249], which demands that any 'good' modeling remains consistent over changes.

Requirement MU4: Being semantically correct
The method must be semantically correct, i.e., especially formulated using a consistent terminology, pertaining to its different language fragments.

Understandability of a method is a key prerequisite to usability and semantic as well as syntactical correctness are necessary to promote understandability. For a method that builds on practice-proven prescriptions from various origins, the question of a clear conceptualization of the domain as well as of utilizing a consistent terminology is crucial. The terms constituting this terminology in addition have to be embedded into a consistent overall structure, i.e., be unambiguous and not conflicting with respect to their statements about EAs. Further assuming that the method employs a well-defined syntactical structure for providing its prescriptions as well as the language fragments, this requirement directly aligns to the one of *language adequateness* as stated by Schütte and Rotthowe [SR98, pages 246–247].

Requirement MU5: Leveraging tool supported design
The method must be supported with a tool that guides the users in performing the method's steps and helps to access the language fragments.

The analysis of the state-of-the-art of EA modeling languages in Section 3.3 reveals, that there is a large number of different guidelines to address a wide range of different EA management-related problems. Therefore, the development method builds on a large number of language fragments. The users of the method should be supported by a tool that helps them to find the adequate fragments in a timely manner. Further, the tool should guide users through the steps of the method, documenting the configurations and integrations already performed, and support the remaining ones. Thereby, usability of the method is greatly enhanced by lowering the investment of the users. By providing a dedicated tool, we follow the principle of *economic efficiency* according to Schütte and Rotthowe [SR98, pages 247–248].

Requirement MU6: Integrating with other EA management approaches
The method must link the corresponding language fragments with the prescriptions made in EA management approaches, therefore providing a starting point for integration and comparison.

The language fragments should not be developed to be used solely by the method, but should also be designed to facilitate integration with other approaches, in particular the ones from which the fragments have been derived. For doing so, the fragments expressed in a consistent terminology must retain links to their original descriptions. The requirement reflects a key part of the principle of *comparability* as stated by Schütte and Rotthowe [SR98, page 249], which demands that any 'good' model supports its users in comparing the model to possible alternatives.

4.2.2 General design guidelines according to Hevner et al.

Hevner et al. derive seven key guidelines that reflect the fundamental nature of design science according to which "knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact" [He04, page 83]. For the context of the design problem addressed in this thesis, the artifact under consideration is no actual EA modeling language but with the method for developing such languages. This corresponds to **guideline 1** of Hevner et al. demanding that design science research produces a purposeful artifact in the form of a *construct*, a *model*, a *method*, an *instantiation*, or a combination of the aforementioned [MS95, pages 256–258]:

- The *constructs* constitute a conceptualization that is used to describe the context, problem, and solution domain, i.e., provide a suitable vocabulary for discussions on the design field. For our contribution, the constructs[2] provide a language for describing the organizational context and the EA management problem that should be addressed, as well as the corresponding EA modeling language.

[2]Constructs in this context correspond to the meta-concepts discussed in Section 2.2.2.

- A *model* gives a set of propositions about relationships between the constructs, i.e., provides a useful combination of constructs in respect to a defined notion of utility. In our contribution, the mdoel interlinks the design problems and context with the solutions, i.e., forms the core of the **organized library** of building blocks (see Section 2.2).
- A *method* delineates a set of steps or guidelines on how to perform a particular task, building on the constructs' language. In our contribution, the method prescribes who uses which constructs to document which aspects of the context and problem domain. Further, the method describes which techniques are used to derive an appropriate solution from the model.
- An *instantiation* realizes some or all of the aforementioned artifacts, i.e., demonstrates the feasibility and effectiveness of the artifacts. For our contribution, a nexus-based *information systems generator*, see Section 2.2, forms the instantiation. The generator supports users in accessing the model, in performing the steps of the method, and in applying the relevant techniques, leading to an organization-specific EA modeling language.

In **guideline 2** Hevner et al. [He04] state that the design artifact has to solve a relevant problem, i.e., has to provide an innovative contribution changing the phenomena that occur. For our work this means that the artifact helps to close the gap between the current state in developing EA modeling languages and an intended state according to the requirements of Sections 4.1 and 4.2.

Guideline 3 of Hevner et al. [He04] highlight the importance of evaluation as part of the research process. After applying the design artifact in the target environment, the researcher has to find out, which requirements are satisfied. If the researcher diagnoses that several requirements are not met, the research process can be iterated, thereby evolving and adapting the artifact and providing more elaborate conceptualizations, models, methods, and instantiations. For our work, evaluation has to target the artifact on a meta-level with the methods as outlined in Section 2.3. We have to evaluate the artifacts general applicability instead of a specific application.

The research contribution is subject to discussions in **guideline 4**, which states that the design artifact must provide a clear contribution in the corresponding research field. In [He04] Hevner et al. distinguish three different types of contributions:

- *design artifacts* solving a heretofore unsolved problem, i.e., ones extending the knowledge base,
- *foundations* introducing novel constructs, models, methods, or instantiations for an already solved problem, and
- *methodologies* defining new ways to evaluate design science research contributions.

Regardless of the type of contribution, the artifact has to be implementable against the background of the existing knowledge base. Our work contributes new foundations for the problem of developing organization-specific EA modeling languages.

Guideline 5 of Hevner et al. [He04] focuses on research rigor and requires the application of rigorous methods for constructing and evaluating the designed artifact. In particular, rigor relates to the effective use of the existing knowledge base, meaning that the researchers must be able to ground their findings in the theoretical foundations of the discipline. Further,

rigor depends on the ability of the researchers to skillfully select the appropriate techniques to justify and evaluate their artifact. For our work rigor has especially to be considered in respect to the organized library of building blocks.

In **guideline 6** the nature of design as a search process is discussed, emphasizing that 'good' design is not achieved in a single step, but employs iterative search procedures. Such procedures are often based on heuristic search strategies, in which alternative design solutions are created and subsequently tested against the elicited requirements. The decomposition of the actual design problem in sub-problems is a favorable strategy according to the guideline. Nevertheless, the achieved sub-solutions have to be analyzed with respect to unforeseen interdependencies. Our work is the result of a long-running research project in which preceding contributions, as the EA management pattern catalog [Ch10], have been presented.

The final **guideline 7** is concerned with the proliferation of the research outcomes. The strong practical focus of design science research makes it necessary that the findings, i.e., the design artifacts and their rationales, are communicated to both a technology-oriented as well as management-oriented audiences. Latter act as consumers of the design artifacts and need sufficient information to decide, if purchasing or making use of the artifact is feasible to address their specific problems. The technology-oriented audience on the opposite need information to replicate and instantiate the artifact, hence focusing more on information about artifact construction. Our research is made available via the corresponding information systems generator and is discussed in relevant publication outlets.

4.3 Contributing theories for the building block-based method

In the following we prepare our method for developing an EA modeling language and start with some definitions and considerations on the notion of the (conceptual) modeling language. Section 4.3.1 establishes our basic understanding of modeling languages and their constituents *syntax*, *semantics*, as well as *notation*, and delineates different ways to describe these constituents. Furthering the considerations from the requirements elicitation, we investigate the subject of relevant architectural properties in more detail in Section 4.3.2. Thereby, we use a formal framework proposed by Guarino and Welti in [GW00a] to reflect on the notion of *identity* in the EA context. In Section 4.3.4 we revisit relevant theories for multi-perspective modeling and the related issue of consistency. Section 4.3.5 discusses how notational aspects and syntactic aspects of a modeling language can be considered independent from each other along a framework established by us in [Er06b]. In the final Section 4.3.6 we present the notion of EA management patterns introduced by us in [Bu07b] and furthered by Ernst in [Er10]. In particular, we revisit how patterns are used to present practice-proven solutions for EA management and analyze potential drawbacks with respect to integration and redundancy.

4.3.1 Modeling languages

Several definitions of the term modeling language are given in literature not at least differing with respect to their commitment to an ontological and epistemological position. In line with Mylopoulos [My92] and Guizzardi [Gu05] which both take an *interpretivist* position, we define a modeling language as follows:

Definition: Modeling language
A modeling language is an explicit set of language primitives and rules that can be used to describe some aspects of the physical and social world for purposes of understanding and communication.

This definition reflects three central characteristics of a language, namely the *language primitives* and *rules* that make up the structure of the language, the *descriptive nature* with respect to the universe of discourse, and the *communicative quality* of the created descriptions. These characteristics are also mirrored in the three constituents of a modeling language as alluded to by Harel and Rumpe [HR00] as well as Kühn [Kü04, pages 30–37]:

Syntax [3] The syntax describes the primitives, of which the language is constituted, the structure of language expressions, and the correctness rules for language expressions. Kühn presents two techniques for defining a language's syntax:

> In the *graph-grammar* approach any model is interpreted as a graph consisting of primitive nodes and edges, i.e., the elements of the grammar's alphabet. Complementing, the grammar provides a set of rules that defines which *productions* can be applied to a valid graph to obtain another valid graph.
>
> In the *meta-model* approach a meta-model, i.e., a model residing on a linguistically higher level of abstraction, describes the modeling primitives as linguistic instances of concepts in another language, the *meta-language*.

Semantics The semantics describes the meaning of the primitives and the expressions in relationship to the represented objects from the universe of discourse. In line with Klar [Kl99], Kühn describes the following techniques to define semantics:

> In *text-based semantics definitions* the meaning of the language primitives is described using natural language.
>
> In *denotational semantics definitions* elementary mathematical objects, e.g. sets, are used to denote the meaning of the primitives.
>
> In *operational semantics definitions* language primitives are mapped to executable operations on an abstract virtual machine.
>
> In *axiomatic semantics definitions* logic axioms and inference rules are used to mirror language primitives.
>
> In *type-theoretic semantics definitions* language primitives are mapped to higher-order mathematical and type-theoretic objects, e.g. data types and algebras.

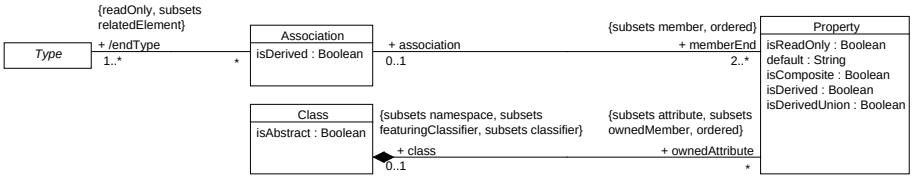
Notation [4] The notation describes how the language primitives and expression are visualized, i.e., represented in a textual or graphical manner. In respect to defining notations, Kühn distinguishes two techniques:

> *Static notation definitions* assign, usually by example, a static symbol to any syntactic primitive.
>
> *Dynamic notation definitions* assign a symbol to the syntactic primitives and add rules for translating states of the primitive onto visual variables of the symbol.

[3] The **syntax** is sometimes also alluded to as *abstract syntax*.
[4] The **notation** is sometimes also alluded to as *concrete syntax* in distinction to the abstract syntax.

Below, we exemplify the language constituents along the well-known *Unified Modeling Language* (UML) [Ob10d] and explain how syntax, semantics, and notation of selected elements of *UML class diagrams* look like. In the example, we find a situation common to many other modeling languages, especially in the EA context: Firstly, the semantics of the modeling concepts is provided in an informal and textual manner. Secondly, the graphical notation of the modeling language is given by example, which might cause ambiguities in the utilization of notational concepts. Finally, the description of language's syntax by using a meta-model is a source of inconvenience also, as the meta-model itself must be expressed in the primitives provided by the *meta-language*. While EA modeling languages often resolve this 'issue' by using multi-purpose modeling languages as the UML [Ob10d] or *Entity Relationship* (E/R) modeling [Ch76], the UML itself has to address this issue in a more sophisticated manner. UML introduces are core set of language primitives, the so-called *UML infrastructure*, whose syntax can be described by themselves, more precisely by instances of a modeling language, whose syntax is isomorph to the one of the UML infrastructure.

Example 4.1: UML language primitives. Central abstraction of UML class diagrams is the concept *class*, of which *properties* and *associations* should be described. The standard defines a class as "a type that has objects as its instances". In a similar fashion, the semantics of the property and the association are textually defined as "a declared state of one or more instances in terms of a named relationship to a value or values" and "a set of tuples whose values refer to typed instances", respectively. Defining the syntax of the language, the UML uses a class diagram as follows:



The notation used to represent the language's constructs is given in the UML specification in a table as the subsequent. The table contains graphical examples that illustrate graphical representations for the concepts:

Class	Graphical representation of a class named "Class"
Class attribute1:type1 attribute2:type2	Graphical representation of a class with two attributes named "attribute1" and "attribute2"
Class1 — associates ▶ (0..* / 0..1) — Class2	Graphical representation of an association "associates" connecting two classes

4.3.2 A formal ontology of properties

The IEEE Recommended Practice 1471:2000 in its adapted version of Wittenburg [Wi07] can be used to understand the environment in which EA descriptions are employed. Characteristics of this environment also pertain to the EA modeling languages. In the light of the quality criteria for the language elicited in Section 4.1 the standard requires a brief revisit. Externalizibility criteria **Ex1** and **Ex2** give rise to a more detailed perspective on the nature of viewpoints in EA descriptions. The criteria introduce a distinction between **goals** and **concerns**, which are combined to a stakeholder's specific EA management-related problem. This distinction is of interest for our method as it allows to re-use the operationalization of an abstract goal, as *increase homogeneity*, on different architectural concerns, e.g. business applications or hardware devices. Put in the perspective of conceptual modeling, above distinction yields two fundamentally different properties that an architecture element can have. Firstly, an element can represent a 'thing', e.g. a business application or a hardware device. Secondly, an element can represent some sort of property, e.g. being homogenizable. While being a 'thing' element entails to have a set of properties, e.g. a name, being a 'property' element adds a particular perspective on an associated thing element. Moreover, being homogenizable is not necessarily a permanent property of an element, at least, if the stakeholders' perspective is taken. It can be that business applications are subject to the goal of increased homogeneity for a certain period of time, after which the focus of the EA management activities shifts to homogenizing business processes. At this point, it is not longer necessary nor sensible to model that business applications are 'homogenizable', i.e., the corresponding property of the business applications ceases to exist.

With the framework of Guarino and Welti [GW00a] we put above considerations on a more sound basis. According to them, the term *property* is in practice often used confusing in respect to its taxonomic structure. Together, the properties form the **conceptualization** of the world [GW00b] and provide the basis for conceptual modeling. This according to Guarino and Welti inevitably leads to difficulties with respect to formally understanding the conceptual models as well as in applying the models to an actual modeling domain. Guarino and Welti devise in [GW00a] and in [GW00b] a formal toolkit that seeks to remedy the aforementioned issues. The toolkit introduces three principles, called *meta-properties*:

- *unity* is a relationship between an entity and its constituting parts,
- *identity* is an equivalence relation between entities, stating whether two entities are the same, and
- *rigidity* describes that a characteristic is essential, i.e., is constant for an entity.

Especially the later two meta-properties are of interest for our subsequent considerations.

Guarino and Welti ground their understanding of identity in the philosophical concept of the *identity condition* (IC). Given an arbitrary property, an identity condition is established via an *equivalence relationship*. Whenever this relationship holds between two properties, these properties are considered identical. This statement may at first sight seem to be an eloquent formulation of a tautology, but at a closer look supports a formal understanding of the principle of identity. The formalization of an IC on instances $\mathcal{I}$ reads as follows:

$$\forall x, y \in \mathcal{I}: \ \phi(x) \wedge \phi(y) \Rightarrow (\lambda(x, y) \Leftrightarrow x = y)$$

The relational predicate λ supplies the equivalence relationship. This understanding of ICs is nevertheless not fully satisfactory. Firstly, the formulation does not allow to distinguish between *supplying an IC* and *carrying an IC*. This distinction especially applies in respect to properties that do not hold permanently, but *inherit* from a permanent property. Secondly, the formulation does not directly account for the existence of instances and is hence not capable to express that one instance existing at one point in time is identical to another instance existing at a different point time. In [GW00a] Guarino and Welti respond to the first issue by distinguishing between *necessary* and *sufficient ICs*. They resolve the second issue by a distinction between two kinds of identity in respect to the points in time from $\mathcal{T}$. In formal terms, these kinds read as:

- *synchronic identity* with $\forall x, y \in \mathcal{I}, t \in \mathcal{T} : \ \phi_t(x) \wedge \phi_t(y) \Rightarrow (\Lambda_{t,t}(x, y) \Leftrightarrow x = y)$ and
- *diachronic identity* with $\forall x, y \in \mathcal{I}, t, t' \in \mathcal{T} : \ \phi_t(x) \wedge \phi_{t'}(y) \Rightarrow (\Lambda_{t,t'}(x, y) \Leftrightarrow x = y)$.

The parameterized predicate ϕ_t evaluates to `true`, if property ϕ holds for the (existing) element at the designated point in time. The equivalence relationship is replaced by a parameterized relational predicate Λ for two selected points in time.

Time further plays an important role for the meta-property of *rigidity*. Guarino and Welti [GW00a] state that a property is rigid, if it is *essential* to all corresponding elements. This means that any element, which has such a property at one point in time, also has the property at any point in time, while it exists. Using temporal logic, we can formalize the notion of rigidity as

$$\forall x \in \mathcal{I} : \ \phi(x) \Rightarrow \Box\phi(x).$$

ϕ denotes a predicate representing the property and $\Box$ stands for the *universal quantifier of time*, i.e., designates that the subsequent statements are bound to hold for all points in time. Negating the notion of rigidity leads to the meta-property of being *non-rigid* [GW00a] formalized as

$$\exists x \in \mathcal{I} : \ \phi(x) \wedge \neg\Box\phi(x).$$

This means that there exists at least one instance, for which the property holds at some particular point in time, but is not essential over the lifetime. A more interesting meta-property can be derived from rigidity, designating some properties as *anti-rigid* [GW00a]. An anti-rigid property is not essential to all of its corresponding instances, i.e., any instance which holds the property at some point in time, does not hold it at one or more other points in time, reading as

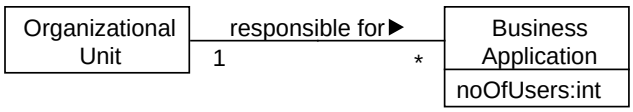
$$\forall x \in \mathcal{I} : \ \phi(x) \Rightarrow \neg\Box\phi(x).$$

Anti-rigid properties are frequent in the context of describing EAs, as life-cycle related properties as *being in planning* or *being in dissolution* are prototypic anti-rigid properties. Any particular business application is planned at some point of its lifetime and inevitably faces dissolution at the end of production. A method committing to develop sound and easy to use EA modeling language has to account for the anti-rigid nature of relevant domain-inherent properties. From this perspective, the method can beneficially build on the ontological framework of Guarino and Welti [GW00a].

4.3.3 Temporal and bi-temporal modeling

The modeling of temporal, i.e., time-dependent, facts is a prominent topic especially in the field of database research. Date [Da00, pages 730–731] understands a *temporal database* as a database "that contains historical information instead of, or as well as, current data". Date introduces the basics of temporal databases, starting with the concept of the *timestamped fact*. Any such fact is valid since a particular point in time and is valid until a (different) point in time. Both points can be indefinite, i.e., the corresponding facts may hold ever since or throughout. The points in time are determined by events that validate and invalidate the timestamped fact, respectively. This understanding is reflected by a **valid time** assigned to any fact in the database. Contrary, Date does not assume that the fact is immediately 'made known' to the database. This means that a potentially different point in time exists, at which the timestamped fact becomes part of the database, as well as some point in time can exist, where the fact is deleted from the database. The first point, when the fact is added, is called in line with Date the **transaction time**. Especially, when planning is concerned, transaction time and valid time are not necessarily identical, as also the following example explains.

Example 4.2: Transaction in EA modeling. An enterprise documents the business applications together with information on their number of users. In addition, information on the organizational unit responsible for any particular application at a point in time is stored. This entails an information model for their EA as shown below:



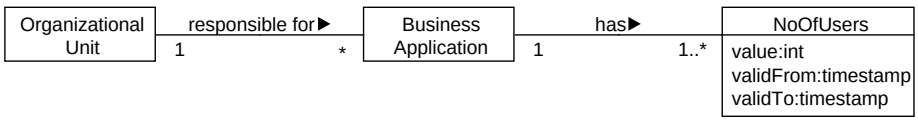
On January, 7th information on the business applications as-is is collected and entered into the database. Accordingly, transaction and valid time for the corresponding facts is January, 7th. A week later, the enterprise architects and business architects meet and decide on the application portfolio for the next planning period. Several applications are assigned to different organizational units or undergo changes in their user-base. These architectural changes are planned to be effective October, 1st (valid time), and are documented on January, 14th (transaction time).

For any relevant dimension of time (valid and transaction), the corresponding fact in the database has to be equipped with a timestamp, that stores the temporal information. If only one dimension of time is considered, the corresponding model is called a *temporal model*. A model equipped with both valid time and transaction time is accordingly called a **bi-temporal** model. Bi-temporality raises several implications in respect to constraints applying to the database. Uniqueness of a relationship end, for example, changes to uniqueness per point in

time, yielding a different set of constraints and evaluations. These issues become even more obvious, when object-oriented models are considered.

Anderson [An99] as well as Carlson et al. [CEF99] discuss, what temporal and bi-temporal modeling imply for object-oriented models. These authors present patterns for 'things that change with time'. Anderson putts particular emphasis on storing historic data. Central thereto is the pattern *edition* [An99, pages 267–270] that links a *memento* (cf. Gamma et al. [Ga94]) to a change transaction. Thereby, historic states of an object can be stored. In a similar vein, but with a slightly different intention, Carlson et al. present patterns targeting properties and relationships with limited periods of validity. In Example 4.3 we detail the difficulties associated with modeling a *temporal property* [CEF99, pages 241–247].

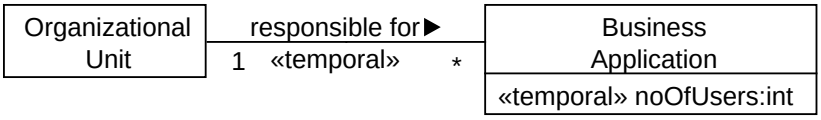
Example 4.3: Valid time in EA modeling. The enterprise implements the information model into the an object-oriented system. In order to reflect that a particular state of the user-base of a business application is—according to the plan—only valid for a certain period of time, the model is adapted as follows.



The information model has changed in two significant ways: firstly, the time-dependent property has been converted to a first-level class, equipped with timestamps. Secondly, a change with respect to the multiplicities has occurred. While formerly any business application had one particular value for the property NUMBER OF USERS, it now supplies at least one related instance of the property class. Without an additional constraint, such information model would allow that contradicting information on the user base at a particular point in time was supplied. But, as Carlson et al. argue in [CEF99], even if such constraint was provided, the information model becomes less clear. The authors propose a stereotype «TEMPORAL»to indicate that a particular relationship end is temporal, i.e., that the supplied multiplicity holds for any point in time.

The pattern *temporal association* of Carlson et al. [CEF99] can be used to add temporal validity to an association in an object-oriented model. Similar to the temporal property, the relationship (association) is reified in a class with properties indicating the period of validity. The make-up of the model is thereby changed in two ways: firstly an additional class is introduced and secondly the multiplicities are adapted appropriately. Again modeling clarity is lost, such that Carlson et al. advocate to use the stereotype introduced above. Example 4.4 shows the stereotypes applied.

Example 4.4: Valid time and transaction time in EA modeling. Finally, the enterprise implements the information model with all time-related features into an object-oriented system. The actual model to be implemented reads as follows.



4.3.4 Consistency in multi-perspective modeling

Quality criterion **Cp1 multi-perspectivity support** raises several implications with respect to the interplay of the different perspectives as well as their underlying modeling languages. A key implication relates to the question of *consistency* between the different perspectives. Andrade et al. discuss in [An04] a framework for understanding how different *viewpoints*, as they call the perspectives, relate. The underlying definition of viewpoint is slightly different from the one given in the IEEE Recommended Practice 1471:2000 [IE00, Wi07], as Andrade et al. state that a viewpoint contains partial information about an actual problem in the universe of discourse. According to the standard, this information would in turn be part of a *view* corresponding to the viewpoint. Based on the more embracing definition of the term, Andrade et al. introduce in [An04] the notion of the *discrepancy* between two viewpoints which can take two distinct forms:

- A *conflict* describes a situation, in which the problems to be addressed with one viewpoint interfere with another viewpoint's underlying problems.
- An *inconsistency* describes a situation, in which information established by one viewpoint is violated, i.e., contradicted, by information in the other viewpoint.

The two types of discrepancies necessarily entail two different resolution strategies, of which the strategy for addressing inconsistencies is of particular interest for our considerations. Andrade et al. propose in [An04, page 288] that in case an inconsistency is detected, firstly the contradictory pieces of information can be "refuted" against reality, and secondly a "transfer" to the correct information is initiated. Especially the first step of the process is to be distinguished from the strategy of "negotiation" that is applied to resolve conflicts, i.e., intentional discrepancies. Nevertheless, the refutation against reality does not rule out the involvement of knowledgeable stakeholders, which are necessary according to the epistemological position of *interpretivism*. The position of interpretivism on the downside aggravates to distinguish between the two types of discrepancies, but calls for a design of the corresponding modeling languages to prevent accidental inconsistencies.

A more detailed perspective on inconsistencies is taken by Nuseibeh et al. [NKF94, page 762]. They identify a viewpoint with a "distributable object encapsulating partial knowledge [...] about a systems and its domain" and decompose a viewpoint into five "slots":

- the *style* which describes the notation, i.e., representation schema, used by the viewpoint,
- the *work plan* that delineates the processes and activities in which the viewpoint is used,
- the *domain* which identifies the concern in the universe of discourse,
- the *specification* that supplies a partial description of the universe of discourse, and
- the *work record* which traces the development state and history of the viewpoint.

The work plan as well as the work record are only of limited interest in respect to consistency between different viewpoints, although the plan can cause conflicts. The style also does not critically interlink with the question of consistency either. Contrariwise, both the domain and the specification have to be considered, when relating viewpoints with each other in order to reason on consistency. Nuseibeh et al. introduce in [NKF94] four types of relationships between viewpoints, namely *independent*, *existentially dependent*, *partially overlapping*, and *totally overlapping*. Independent viewpoints cannot interfere with each other, while overlapping viewpoints share both parts of their specifications and their domains. When a viewpoint is existentially dependent from another one, this requires a domain-inherent relationship, according to which the existence of an instance in the specification of the first viewpoint entails the existence of one or more instances in the dependent viewpoint's specification. This actual nature of this dependency is not discussed by Nuseibeh et al. in [NKF94] but indications are provided that refinement-relationships and temporal relationships should be considered. Nuseibeh et al. introduce the concept of the *viewpoint template*, referring to types of viewpoints that are not actually instantiated. Such a viewpoint template defines style, work plan, and domain, of which only the latter accordingly is of interest for considerations on consistency. The relationships between viewpoints are in line considered as instantiations of corresponding relationships defined between viewpoint templates. These relationship definitions in turn commit to the relationships between the domains (*overlap*) or in respect to the work plan (*existential dependency*).

In [DQS08, page 737] Dijkman et al. establish "re-usable consistency rules" pertaining to "multi-viewpoint design" for architectures. They develop a conceptual framework, see Figure 4.2, building on the IEEE Recommended Practice 1471:2000 [IE00] and introduce additional elements thereto, namely:

> A *level of abstraction* describes a relative position in the design process that indicates which information is considered essential at this point in design.
>
> A *system property* designates a set of characteristics of the part of the system under consideration, being essential to the design.
>
> A *concept* is an abstraction of a design-relevant property of the system defined in a stakeholder's viewpoint on the design.
>
> A *concept instance* complements the concept with an actual value of the property, reflected in a particular view on the system.

Dijkman et al. [DQS08, pages 742–743] discuss formal consistency rules interlinking different viewpoints. Central thereto is an understanding of three essential and coarse-grained concerns, namely the *structural*, the *behavioral*, and the *information* concern. Dijkman et al. provide for each of these high-level concerns a dedicated set of abstract concepts as *entity*, *action*, or *type*, for which particular consistency rules are described using a formal mechanism. For any

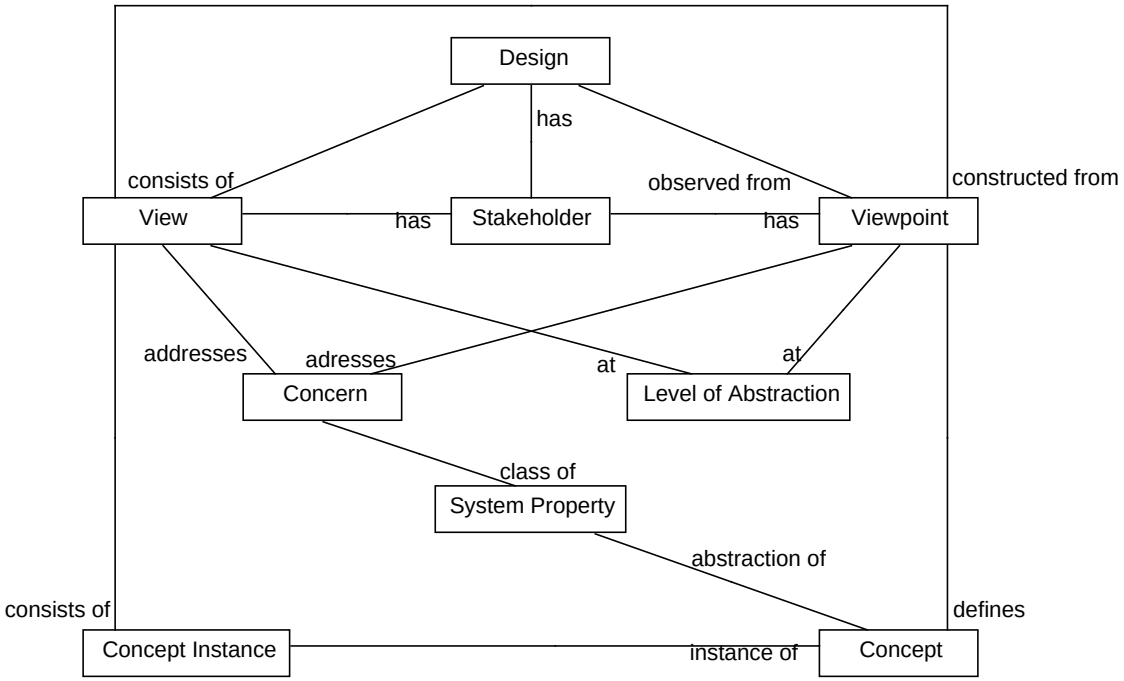


Figure 4.2: Framework for multi-viewpoint design (cf. [DQS08])

language not bound to these concerns, the approach's underlying considerations on consistency, as described by Dijkman in [Di06b], provide a good starting point. There, he distinguishes between four approaches to define consistency between viewpoints [Di06b, pages 46–51]:

1. An informal approach building on natural language consistency rules.
2. An approach building on a relational algebra over the concept instances, i.e., using declarative consistency invariants.
3. An approach defining a formal semantics for concepts in terms of a mathematical representation, which further yields consistency rules pertaining to concept instances.
4. An approach defining a semantic mapping for concepts to so called "basic concepts", on which corresponding consistency rules are established.

These approaches are not to be considered to be exclusive, but can according to Dijkman be combined. He specifically describes the benefits of combining approaches 3 and 4 as well as 2 and 4. In the latter case, well-established consistency mechanisms as the *Object Constraint Language* (OCL) [Ob10c] can be re-used to describe the consistency invariants on basic concepts instead on arbitrary concepts. This reduces the overall complexity and amount of consistency rules. In a similar sense, the formal semantics in approach 3 can be confined to target only basic concepts as of approach 4, thereby again reducing the complexity of formalization in respect to consistency. Regardless of the approach taken, the discussions of Dijkman provide further input to the question on how to formulate consistency rules.

On a general level, he distinguishes between two types of relationships between viewpoints, namely the *overlap* and the *refinement* relationship. One viewpoint refines another, if it covers all of the other's SYSTEM PROPERTIES and resides on a lower level of abstraction. Two viewpoints overlap, if they share at least one SYSTEM PROPERTY via the contained concepts. On actual views, Dijkman revisits the cases of overlap on the level of concept instances in [Di06b, page 161], distinguishing *equivalent*, *partly equivalent*, *complementary*, and *composite equivalent*. Figure 4.3 illustrates these different cases.

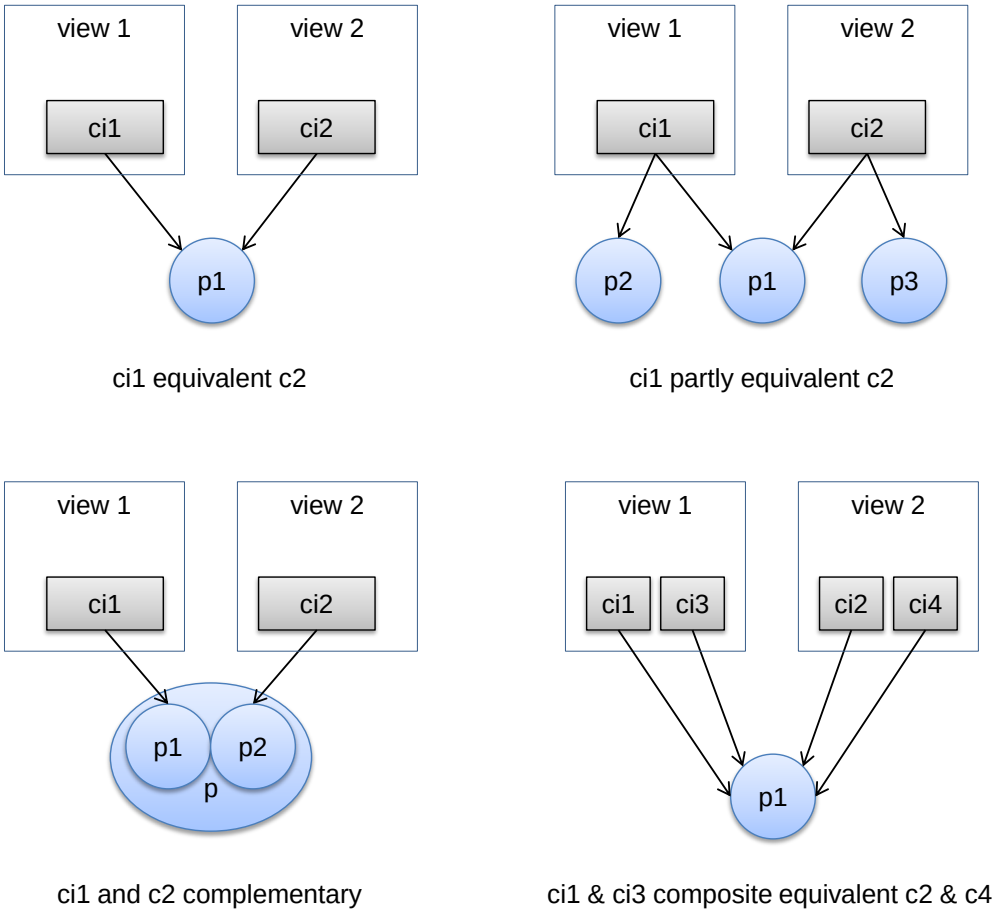


Figure 4.3: Cases of concept instance relationships according to Dijkman [Di06b, page 161]

Above discussions not only highlight the importance of consistency in multi-perspective modeling environments, but also establish a conceptual basis on which our method builds. In particular, the distinction between intensional discrepancies (*conflicts*) and accidental discrepancies (**inconsistencies**) is introduced by Andrade et al. [An04]. On the level of interrelating viewpoints, two different kinds of such relationships have to be distinguished: **overlaps** and *refinements* (existential dependencies). For the overlap relationship, the considerations of Dijkman [Di06b] provide further concretizations that relate to viewpoint-level properties.

4.3.5 Decoupling notational aspects of a modeling language

Different modeling languages, e.g. the UML, do not only provide a single notation for the language's concepts, but supply alternative visual representations for selected language primitives. This reflects a situation, where the syntax of the language remains fixed, while multiple different symbols are used for a single syntactical primitive to reflect the needs of different target audiences. As the analysis of the state-of-the-art in EA modeling languages revealed, the situation in the context of EA management is similar. Each modeling language brings along a unique notation and the using organizations are likely to either have adapted these notations according to their needs or have developed their own notations. Moreover, also in one organization, different stakeholders can use different notations for the same concepts.

Some research in the field of modeling languages and notations is motivated from the area of graphical modeling, more precisely from the tool perspective on this subject. This ascribes to the fact that building generic modeling toolkits for arbitrary languages necessarily calls for techniques that allow applying arbitrary symbolic or textual representation mechanisms to the syntactic primitives. Prominent proponents of such generic modeling toolkits are the *Generic Modeling Environment* (GME) [Le01] and the *Graphical Modeling Framework* (GMF)[5]. Both toolkits provide dedicated concepts for describing visualized instances of syntactic primitives, although these concepts are not solely of graphical nature but represent some sort of 'entanglement' between symbol and primitive instance. The actual situation in the graphical model resembles the one in the UML diagram interchange [Ob06b]. A graphical model in terms of the diagram interchange is a model of solely graphical instances, as rectangles, tightly coupled with a model of primitive instances. Such an entanglement in turn is native from a tool's perspective, but remains limited with respect to its merits both for understanding what a notation is and for decoupling notational aspects.

Domokos and Varro explore in [DV02] an alternative approach for graphically representing models. This approach builds on a graphical model instantiating a generic visualization meta-model, i.e., a meta-model that introduces primitives as rectangles, circles, and lines. The instances of graphical primitives are not directly coupled with the instances of the syntactic primitives that they should represent. In turn, a *model-to-model transformation* is applied to create symbols representing the underlying primitive instances. While this approach in general can be regarded powerful for describing notational aspects only loosely coupled to the syntax, its expressiveness is limited. This ascribes to the visualization meta-model that Domokos and Varro employ in [DV02], as it focuses on graph-like 'node-and-edge' diagrams and does not provide more sophisticated graphical primitives, i.e., for nesting symbols. Later primitives are nevertheless prevalent in the context of graphical models of the EA.

We discuss in [Er06b] a closely related approach for creating graphical representations of EA models according to a predefined notation. This approach builds on a model-to-model transformation linking syntactic primitives of the EA modeling language, contained in what we call the **information model**, to graphical primitives from the visualization meta-model, the so-called **visualization model**. Compared to the complementary model presented by Domokos and Varro [DV02], our visualization model is richer, as it provides a class of visual primitives called **visualization rules**. These rules are used to describe graphical relationships between symbols, e.g. to express that one symbol instance is nested into another one. As a

[5] Information on GMF can be found at `http://www.eclipse.org/gmf/`, last accessed 04-05-2011.

part of the approach, we show how the visualization rules relate to mathematical statements, which in turn can be used to compute a rule-conform layout of the visualization. Figure 4.4 shows the central mechanism of our approach, and describes the executable nature of the mechanism as discussed by us in [Bu07a]. Whereas Wittenburg has later shown in [Wi07], that the mechanism can be applied to generate relevant visualizations of the application landscape of an enterprise, a limitation of the approach applies. The visualizations are graphical representations generated from the syntactic primitives. In this sense, the model-to-model transformation provides only 'half of' a notation, which in turn should also allow adding, changing, and removing primitives in the described model. Our approach in [Er06b] describes a one-way transformation from syntactic to visual primitives, whereas a notation has to define both ways.

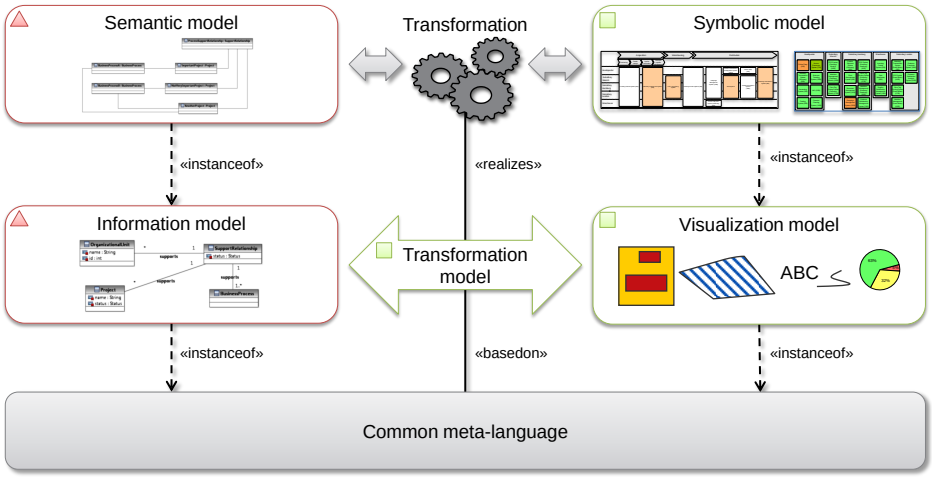


Figure 4.4: Model-transformation approach to visualization generation [Bu07a]

Despite the aforementioned limitations, our model-transformation approach [Er06b] forms a good basis to settle a technique for defining notations. This means that our technique in its core builds on a language for describing model-to-model transformations, i.e., **notation functions**.

$$\text{language primitives} \leftrightarrow \text{visual primitives} \qquad \text{(notation function)}$$

Above, we formulate a *bijective* function between the syntactic primitives and visual primitives of the EA modeling language. In case, the language's models are used read-only, the resulting **representation function** does not need to be bijective. This in turn means that no inverse of the representation function, i.e., no *interpretation function* exists.

$$\text{language primitives} \rightarrow \text{visual primitives} \qquad \text{(representation function)}$$

In Chapter 5 we explore the notion of **representation functions** and **notation functions** in more detail to understand how above technique integrates into the formal framework for describing architectures provided by the IEEE Recommended Practice 1471:2000 (cf. Section 3.1.3) and its successor, the ISO standard 42010 [In07].

4.3.6 EA management patterns

In [Bu07b] we discuss the difficulties, which an organization willing to establish an EA management function finds itself confronted with. In particular we elaborate how the absence of proven-practice prescriptions on a concrete and problem-specific level leads organizations to unwanted results. Especially the greenfield approach of developing EA modeling languages causes organizations fall for typical difficulties as "giant models" [Bu09d]. In [Bu07b] we respond to these difficulties by applying the notion of the *pattern* onto the field of EA management. A pattern is thereby understood in line with Alexander et al. [Al77, page x] as an observed and not invented solution for a recurring problem.

The EA management patterns outlined by us in [Bu07b, pages 154–155] are in line with above argument "small, re-usable units preferably based on established practices" and can be distinguished into three different types, namely:

Methodology patterns (M-pattern) [6] that define steps to be taken in order to address problems of a distinct type. These M-patterns further delineate the intended usage context and describe possible consequences from applying the steps. Finally, the patterns establish relationships to viewpoints that can be used in addressing a problem.

Viewpoint patterns (V-pattern) that describe how certain aspects, i.e., concerns, in the EA can be presented to the actors and stakeholders of the method. The V-patterns' underlying conceptualization of reality, i.e., its corresponding information model, is contained in a referenced pattern.

Information model patterns (I-pattern) that supply conceptual models for describing the EA or certain aspects thereof. The elements contained in the I-pattern are further complemented with textual definitions of their semantics. These textual definition together form the *glossary* that accompanies the patterns.

Patterns of all three types follow a standardized documentation schema derived from the *canonical form* for patterns according to Buschmann et al. [Bu96]. Ernst adopts this form for the context of EA management first in [Er08] and discusses the different components thereof in [Er10]. Accordingly, an EA management pattern always provides a fact sheet with an expressive name, a unique identifier as well as versioning information, and follows a typical structure consisting of eight components:

1. The brief *example* describes a practical application case of the corresponding pattern. The case can be anonymized or can contain real-world information.
2. The short description of the *context* delineates environmental factors that have to hold for the pattern being reasonably applicable.
3. The appellative description of the *problem* directly addresses the prospective users and delineates the type of problems onto which the pattern can be applied. The general problem statement is further detailed with *forces* that apply in the problem space and can influence the solution.

[6] More precisely, we should have talked about *method patterns* in [Bu07b].

4. The *solution* describes the steps to be taken (M-pattern), exemplifies the viewpoint to be used (V-pattern), or shows the information model (I-pattern). As part of the solution possible resolutions to the aforementioned forces are provided and corresponding alternatives are denoted.
5. The *implementation* delineates how the solution can be operationalized, e.g. which roles have to be established and which tool support has to be provided.
6. The *known uses* explicitly account for organizations that have successfully applied the described pattern. This section can be omitted for reasons of anonymity.
7. The *consequences* denote known side-effects that can arise from the implementation of the pattern.
8. Finally, related patterns are listed under *see also*, thereby establishing interconnections between patterns.

In respect to the interconnections between patterns, three different categories have to be distinguished. Firstly, interconnections can target patterns of a different type necessary or beneficial for implementing the pattern under consideration. Secondly, patterns of the same type can be targeted to denote interdependencies as being a prerequisite or being mutually exclusive. Finally, patterns of all types committing to the same terminology can be referenced. For the EA management patterns presented in the EA management pattern catalog, presented in [Bu08b] and [Ch10], only interconnections of the former two categories are made explicit. Thereby, especially the first type of relationships deserves special attention. Patterns of all three types are closely related to each other, as to say that any M-pattern involves at least one V-pattern to convey the relevant architecture information to the method's users. Further, a V-pattern links to an underlying I-pattern that supplies the concepts used in the viewpoint. The tightness of this relationship between V-pattern and I-pattern becomes obvious when regarding an actual viewpoint as a language for describing EAs. The notation of this language (cf. Section 4.3.1) is specified in the V-pattern, whereas both syntax and semantics for the language are supplied in the corresponding I-pattern.

In respect to the terminological relationships between patterns, the pattern catalog presented in [Bu08b] and [Ch10] takes a pragmatic perspective. A common glossary underlying the I-patterns and henceforth also the V-patterns ensures terminological consistency. To achieve this, the wording of selected patterns had to be adapted prior to integrating them into the pattern catalog. When it comes to the M-patterns, terminological clarity is only partially maintained. Especially in respect to denoting the goals and corresponding concerns, the M-pattern remain abstract, thus leaving their relationship to the catalog's underlying terminology unclear. While this is in line with Ernst [Er10] not to be considered a drawback of the catalog but an inherent characteristic of pattern-based approaches, users of the catalog have to resolve the according issues themselves. This in turn aggravates the utilization of the provided prescriptions.

Our pattern-based approach towards EA management presented in [Bu07b] provides a good starting point for a method for developing EA modeling languages. Especially, the relationships between the patterns as well as the consistent distinction between notational and syntactical aspects of EA modeling are valuable ideas.

4.4 Summary

In this chapter we elicited fourteen quality criteria for EA modeling languages and classified these quality criteria along the *appropriatenesses* of Krogstie's language quality framework [Kr02]. Due to their generality, the quality criteria apply to all organization-specific EA modeling languages and hence designate expected characteristics of the outcome of using the development method. Therefore, the quality criteria pertain to this thesis' contribution, as the development method should by design facilitate the creation of EA modeling languages fulfilling these 'instance-level' quality criteria. We additionally elicited requirements for the method and organized library themselves grounded in the guidelines of modeling, see Schütte and Rotthowe [SR98]. Finally, we approached the thesis' contribution from a design perspective, mapping the fundamental design principles of Hevner et al. [He04] to the application domain and devising statements to evaluate the quality of our contribution.

The quality criteria are derived from the criteria of GERAM [BNS03, pages 14–15], the scenarios of the EA management tool survey [Ma08], and the practitioners' statements in the state-of-the-art survey [Bu09a]. Topic **PT4** "EA frameworks offer only minor assistance" and scenarios **SF1** "Importing, editing, and validating model data", **SF6** "Editing model data using an external editor" as well as **SF8** "Handling large scale EAs" do, as discussed in Section 4.1, not provide input to particular quality criteria. Table 4.2 explains which quality criterion builds on which source.

	GERAM [BN94] and [BNS03, pages 14–15] GE1 – GE5	EA management tool survey [Ma08] SF1 – SF9 and ES1 – ES9	State-of-the-art survey [Bu09a] PT1 – PT10
Cm1		SF5, SF9	PT7
Cm2		SF9, ES1	PT5
Cp1	GE2	ES1 – ES9[1]	PT6
Cp2	GE3		
Do1	GE5	ES6, ES9[2]	PT1 – PT3
Do2	GE1	ES1 – ES5	
Do3		ES8	PT9
Ex1	GE3[3]	SF7	
Ex2	GE3[3]	ES5	PT10
Te1	GE2	SF2 – SF4	PT6, PT8[4]
Te2	GE2	SF3, SF4	PT8[4], PT10
Us1	GE3[3]	SF7[5]	PT6
Us2	GE3[3]	SF7, ES7	
Us3	GE3[3]		PT7

[1] Multi-perspectivity is reflected in the plurality of different EA management scenarios, not in a single scenario.
[2] The two scenarios serve as representatives for the diversity of the domain.
[3] Requirement GE3 connects to the question of consistent expressiveness and is hence relevant for all language requirements categorized to externalization appropriateness and user knowledge appropriateness.
[4] Practitioners' topic PT8 alludes to tool support in general and is hence relevant for all language requirements categorized to technical appropriateness.
[5] Standardized symbolic representations are only a minor topic in the scenario SF7.

Table 4.2: Quality criteria of EA modeling language and their sources

The thesis' contribution is a development method for EA modeling languages that per design fulfill the fourteen quality criteria. We employ different theories from bordering fields to address these criteria. In Section 4.3 we give an overview on these contributing theories, along seven groups of quality criteria as follows:

Cross-cutting aspects of architecture modeling are discussed in different quality criteria, as **rights and responsibilities (Cm1)**, **guidelines coverage (Do3)** and **goals coverage (Ex2)** as well as **transformation coverage (Do2)**. Cross-cutting aspects can be reflected by **dispersive** types along the work of Guarino and Welti [GW00a, GW00b].

Temporal modeling pertains to both the type-level and the instance level. **Non-rigid** types can be used to reflect the life-cycle of architecture elements during the transformation of the EA. **valid times** can be used to plan the transformation of the EA, designating when a particular architecture element is planned to become valid. The perspective of bi-temporal modeling contributes to a solution targeting quality criterion **traceability of changes (Cm2)** according to which it is relevant to know, when a certain information was edited (**transaction time**).

Specialization by constraint and **derived properties** are relevant for operationalizing EA goals to more concrete questions, which can be used as basis for an automated analysis in the sense of quality criterion **analysis coverage (Te2)**. Such analyses require dedicated mechanisms for querying EA information models and to perform computations, i.e., to derive values.

Configurable viewpoints employ both computations as well as queries and provide the basis for **visualization generation (Te1)**, which requires operational definitions for the visualizations' underlying viewpoints. This definition builds on the formal understanding of viewpoints as notation or representation functions acting on the EA information model. Our fundamental work [Er06b] outlines how such formal understanding can be established via model-transformations. The generating transformations support a parameterization with according visual primitives and thereby allow to customize the visualization. Via such customization **representational fit (Us1)** is ensured.

Multi-perspectivity in the sense of quality criterion **(Cp1)** is relevant for EA modeling, especially in avoiding different qualities of potential *conflicts* between perspectives. Going into details with specific conflicts called *inconsistencies* in Section 4.3.4 we used the framework of Dijkman et al. [Di06b, DQS08] to understand necessary prerequisites for perspectives making inconsistent statements. In respect to the changing EA management-relevant problems that the EA modeling languages have to address, it is relevant that our development method provides mechanisms to ensure a once developed set of EA modeling languages is **sustainable (Us2)**.

Glossary development is relevant to ensure that the EA models are understood by their prospective users in the sense of quality criterion **glossary support (Cp2)**. Any using organization is expected to form a specific linguistic community, such that concepts to support the adaptation of the glossary to the terminology in line with criterion **terminological fit (Us3)** are needed.

Domain-specific modeling covers the EA from both a **business and IT (Do1)** perspective. This in particular pertains to the modeling primitives that are supported, such that relevant types of the domain are covered as well as prevalent characteristics. Thereby, the users of the model language are empowered to cover their **EA concerns (Ex1)** with meta-conceptualizations that correspond to the domain under consideration.

As a final fundamental concept, we revisited our contributions from [Bu07b] and the one of Ernst [Er10], in which we introduced and successfully applied the notion of the *EA management pattern* onto developing EA management functions in general and EA modeling languages. Discussing the liabilities of the pattern-based approach, we prepare the exposition of our solution and delineate how the EA management patterns can—in application of the research method from Section 2.3—contribute to the development of our solution.

The Guide is definitive. Reality is frequently inaccurate.

Notice at Megadodo Publications – The Restaurant at the End of the Universe

CHAPTER 5

BEAMS: Building Blocks for EA Management Solutions

Our development method uses a *design theory nexus instantiation* in three steps for developing an organization-specific EA management function: **characterize situation**, **configure the EA management function**, and **analyze the EA management function**. In the first step, the enterprise architect and the stakeholders designate the organizational environment of the EA management function [BMS10j, Bu10a], supply information on the used terminology, and select the EA management-related problems. Based on the organization-specific terminology, a **glossary adaptation** is performed. The identified problems provide the basis used during **information modeling**. In the second step actual EA management processes are established. During **viewpoint definition** particular viewpoints for providing and collecting information from participating organizational roles (**participants**) are defined. Steps one and two are repeated until all current EA management-relevant problems are addressed. The final step of the development method analyzes the performance of the EA management function and identifies needs for adaptation, if the selected goals could not be achieved. Figure 5.1 gives an overview of the development method.

In Section 5.1 we explain our basic understanding of an EA management function constituted from management methods and corresponding EA modeling languages. We introduce in Section 5.2 the three types of building blocks that provide re-usable solutions for EA modeling languages. These building blocks are used by the development method, which we describe and exemplify in Section 5.3. The design theory nexus instantiation underlying our approach, i.e., the organized library of building blocks, itself is not fixed, but has to evolve, if new proven EA management solutions become available in practice. In Section 5.4 we briefly revisit the research method of this thesis and show how it is useful for developing and evolving the organized library of building blocks. Final Section 5.5 summarizes the employed mechanisms for preserving consistency in the development method and prepares their formal discussion in Chapter 6.

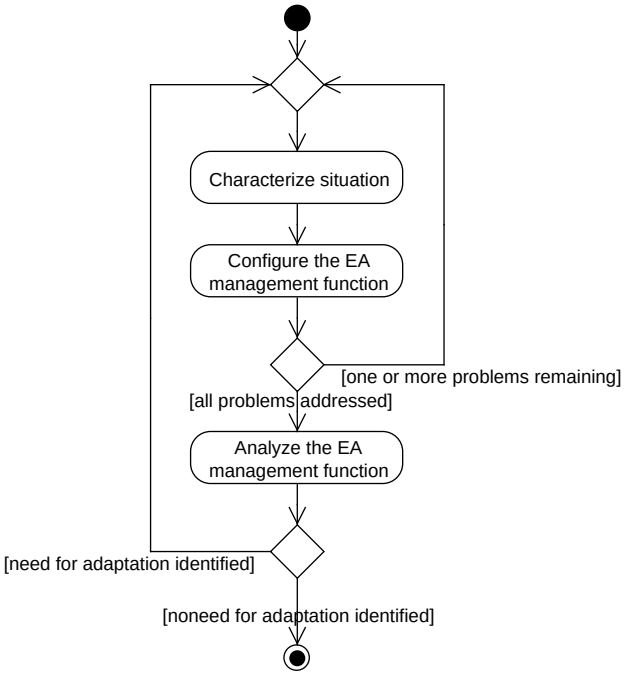


Figure 5.1: Overview of the major steps of the development method

5.1 The basic structure of an EA management function

In an EA management function, the EA modeling languages are used by different participants to execute the tasks that they are responsible for. Figure 5.2 delineates how TASKS, PARTICIPANTS, VIEWPOINTS, and INFORMATION MODELS in an EA management function interrelate. Regarding the viewpoints, we distinguish between ones used for representing and ones used for modeling the corresponding information.

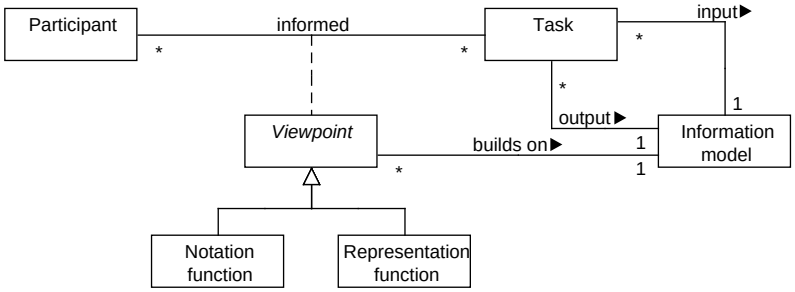


Figure 5.2: Interplay of tasks, participants, viewpoints, and information models

The general quality criteria for EA modeling languages as raised in Section 4.1 have different implications on the nature of our development method, as any correct application of the method should lead to EA modeling languages that fulfill all quality criteria. To prepare these considerations, we introduce a framework of EA modeling language constituents. Central thereto is the **information model**, which in line with our considerations in [Er06b] specifies the syntax of a modeling language via MODEL ELEMENTS. Semantics is considered as a function assigning exactly one predicator to each MODEL ELEMENT. This predicator acts as surrogate for a corresponding part of the conceptualization in the sense of Dijkman et al. [DQS08]. The notation is described as a representation function assigning one VISUALIZATION ELEMENT[1] to each MODEL ELEMENT. In consequence, a predicator (or the thereby represented architectural property) is assigned a particular visualization element by the modeling language. Different languages in turn can cover different sets of predicators and can assign different visualization elements thereto. Figure 5.3 depicts the conceptual model for modeling languages. The architecture property is therein shown in gray, as it represents a purely intrapersonal concept, whereas the three basic constituents of the language—syntax (INFORMATION MODEL), semantics (SEMANTICS FUNCTION), and notation (REPRESENTATION FUNCTION)—are highlighted.

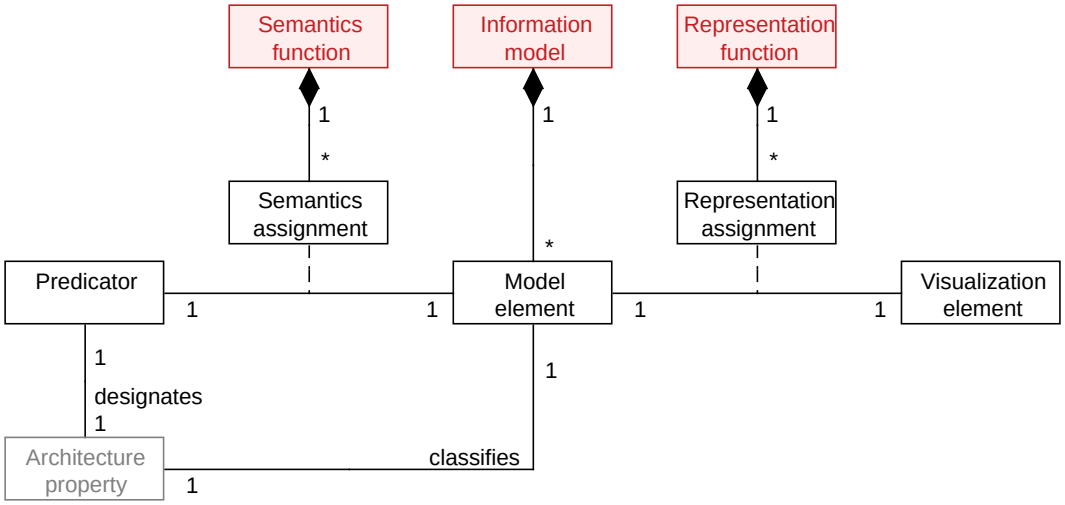


Figure 5.3: General constituents of an EA modeling language

In the combination of different EA modeling languages, the quality criteria relating to *comprehensibility appropriateness*, see Section 4.1.2, and to *user knowledge appropriateness*, see Section 4.1.6, are of particular interest. **Multi-perspectivity support (Cp1)** requires consistency between the descriptions made in different perspectives. This means any two perspectives that cover at least one common architectural property, i.e., *overlap* in the sense of Dijkman [Di06b], make statements that do not contradict each other. In order to achieve this, our development method facilitates the creation of a common information model for all perspectives, such that the known values of architecture properties are reflected in the instantiation of a single and consistent model. The same applies with respect to quality criterion **glossary support (Cp2)** according to which each architectural property is predicated in an

[1] In line with Kamlah and Lorenzen [KL67] arbitrary *speech acts* can notate an element. With the focus of the thesis we constrain our subsequent considerations to visualization elements.

unambiguous and comprehensible manner. Our development method ensures that the single information model is complemented with a single and consistent glossary of interrelated predicators. The terminology, which builds on these predicators, has to be adapted to the organization-specific terms in order to ensure the **terminological fit (Us3)** of the EA modeling languages. This fit pertains to all employed languages, as different languages can bring along elements assigned to the same predicator. Figure 5.4 displays the conceptual model bringing together different EA modeling languages backed in a consistent terminology.

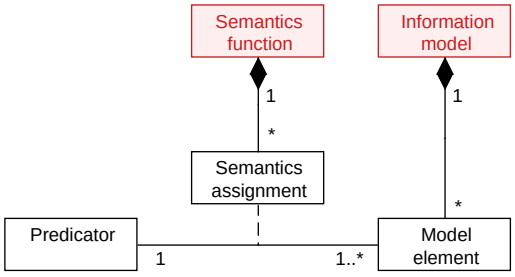


Figure 5.4: Semantics assignments in a set of EA modeling languages

The different perspectives on the EA are realized in different **viewpoints** on the information model. In the sense of quality criterion **representational fit (Us1)**, we have to revisit the differentiation between the **notation function** and the **representation function** outlined in Section 4.3.5. For each single modeling language the notation function establishes a bijective mapping between MODEL ELEMENTS and VISUALIZATION ELEMENTS, i.e., one-to-one assigns visualization elements to model elements. The representation function of a modeling language contrariwise does not establish one-to-one relationships, but can provide an aggregated perspective on the information model elements, e.g. by abstracting relationships or calculating sums. Put in terms of our conceptual model for the constituents of an EA modeling language, we can describe the general distinction between notation functions and representation functions via different cardinality constraints as shown in Figures 5.5 and 5.6, respectively.

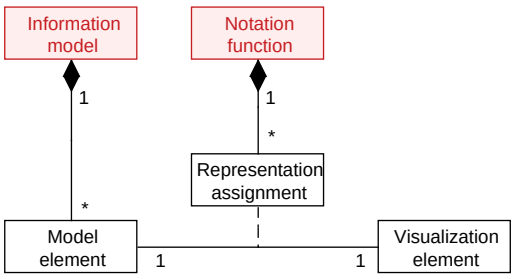


Figure 5.5: Notation function

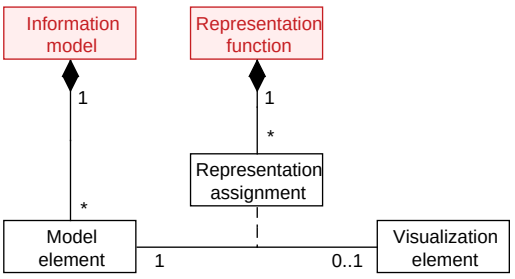
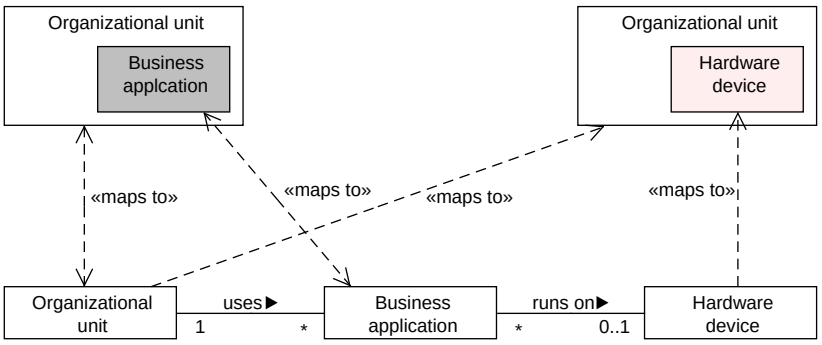


Figure 5.6: Representation function

In Example 5.1 we illustrate the distinction between the different types of functions by linking two visualizations grounded in a common set of MODEL ELEMENTS. These elements can be understood to constitute the organization-specific EA information model. The two viewpoints

are based on sub-models of this information model, the so-called **viewmodels**[2]. The viewmodel of the first viewpoint is derived from the information model by selection, while the second viewpoint's viewmodel also employs aggregation mechanisms.

Example 5.1: Notation and representation function. An organization has decided for an EA information model consisting of the concepts ORGANIZATIONAL UNIT, BUSINESS APPLICATION, and HARDWARE DEVICE as well as their relationships to each other. Based on this information model two viewpoints are established: one depicting the organizational units using business applications (left) and one depicting the organizational units' indirect utilization of the hardware devices (right). The first viewpoint is thereby based on a notation function, i.e., a bijective mapping. The second viewpoint builds on a representation function, omitting the intermediary business applications.



Additionally, the supplied EA modeling languages restrict the part of the organization-specific information model that a participant has access to. From this perspective, the **rights and responsibilities assignment (Cm1)** pertains to the relationships between the modeling language and the management process, in which the language is used.

The information model and the viewmodels target the EA in an embracing manner, covering identifiable and substantial architecture properties on various levels of abstraction as well as layers, **covering both business and IT (Do1)**. In addition to these substantial properties also a variety of cross-cutting aspects are of interest, as alluded to in quality criteria **Do2 transformation coverage** and **Do3 guidelines coverage**. Depending on the kinds of cross-cutting aspects incorporated in the EA model, we distinguish different states of the EA, for which we introduced in [Bu09e] the following terminology:

[2] A similar concept is discussed from a strictly technical perspective by Smith in `http://msdn.microsoft.com/de-de/magazine/dd419663.aspx`, last accessed 04-05-2011, and Fowler in `http://martinfowler.com/eaaDev/PresentationModel.html`, last accessed 04-05-2011.

- The **current state** of the EA represents the status quo of the architecture as described at a certain point in time.
- The **planned state** of the EA represents a future state of the architecture as intended to be at a specific point in time in the future. This state is developed in the EA management process for a specific point in time, as the result of the project-related changes performed up to that specific future date.
- The **target state** of the EA describes a long-term perspective as envisioned at a certain point in time following the strategies and goals of the enterprise. According to the discussions of Buckl et al. in [Bu09c] there is no need to have projects defined for transforming the current or planned state towards the target one.

The distinction between different architecture states implies a twofold understanding of time in EA management, more precisely the continuous planning and enactment processing incorporated therein. In [Bu08c, Bu09e] we introduce an understanding of two time-related dimensions applying onto EA descriptions, namely:

- the time dimension 'described for', designating if an EA state is current, planned for a specific time, or target, and
- the time dimension 'described at', designating the point in time, to which the EA state references, i.e., when it was created.

Figure 5.7 displays the interplay of different EA states along the different dimensions of time.

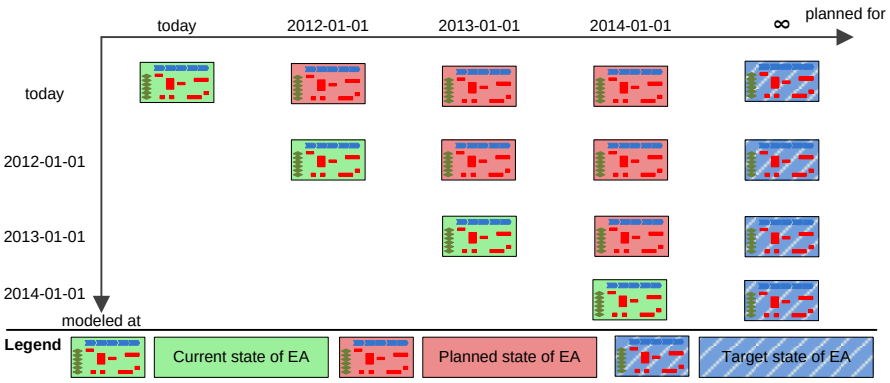


Figure 5.7: Interplay of current, planned, and target states of the EA [Bu09e, page 14]

The differentiation of two time dimensions is a key characteristics of **bi-temporal** modeling, making it necessary that some or all language primitives explicitly account for the points in time, that they were described at (transaction time) and described for (valid time). Following the discussions undertaken by Aier and Gleichauf in [AG10] and by us in [Bu08c, Bu09e], an additional quality in respect to the EA states has to be introduced for developing EA plans. Quality criterion **Cm2 traceability of changes** demands for an EA modeling language that allows storing design decisions, i.e., past plans for the EA. In line with these considerations, we can refine our definition for the term **enterprise architecture**. This definition builds on the general definition for the term "architecture" given in the ISO Standard 42010 [In07].

Definition: Enterprise architecture
The enterprise architecture is the fundamental organization of an enterprise, embodied in its components ranging from business to IT infrastructure, their relationships to each other, and to the environment. The enterprise architecture exists at any point in time and is planned as well as changed exclusively via projects in the boundaries given by principles and standards. The change heads towards a target state, outlined by strategies, visions and goals, which are measured via metrics and KPIs.

Any organization-specific EA modeling language institutes the linguistic primitives needed to address the specific EA management problem in a given context. Organization-specificity on the one hand applies to the semantics assignments and the representation assignments, which have to match the organization's terminology and visual identity. Organization-specificity on the other hand entails problem-specificity. In line with quality criteria **Ex1 concern coverage** and **Ex2 goal coverage** we regard any problem to be the application of an EA management-relevant goal on a specific EA concern. The problem's underlying concern is reflected by corresponding MODEL ELEMENTS representing identifiable architecture elements, whereas goals target intended characteristics of the architecture elements. Reasoning on goals is nevertheless only possible, when corresponding MODEL ELEMENTS are used. Therefore, we come back to the distinction between different qualities of an element to formulate the assumption, that any MODEL ELEMENT representing a goal-specific characteristic, not the goal itself, does not supply an *identity condition* (IC). In Example 5.2 we provide intuitive counterexamples for goal elements having the meta-property of identity.

Example 5.2: Identity conditions in EA modeling. Two organizational units having the same number of employees can be considered equivalent in respect to the perspective reified in the goal 'reduce labor costs'.
Conversely, the same organizational unit generating different revenue at two different points in time is not considered self-equivalent with respect to the goal 'increase revenue per period'.

Above counterexamples advocate for a distinction between MODEL ELEMENTS that reflect architecture elements and those that reflect goals. The analysis framework from Section 3.2 indicates that goals are particular instantiations of a more general principle of describing EAs. Similar to goals, also projects or standards themselves supply an IC, but relate to dependent characteristics that do not supply an IC. The building block underlying our development method have to account for this fact.

5.2 Building blocks for EA management functions

The development method centrally relies on solutions, which have proven themselves useful in practice for addressing particular management problems. These solutions form the **building blocks** used to develop an EA management function or EA modeling languages, respectively. In the following we differentiate between the embracing development method for EA management functions and the particular method for developing EA modeling languages for multi-viewpoint EA management. This yields a distinction between two types of building blocks, namely the *method building block* (MBB) and the *language building block* (LBB). The underlying method-language dichotomy is discussed e.g. in [AS09, page 55] as well as [SW08, page 81] and aligns with the differentiation between the pattern types that we introduced in [Bu07b]. The MBBs describe the steps to be undertaken, the decisions to be made, and the participants to be involved in order to address a specific EA management problem in a given organizational context. The LBBs conversely provide the language primitives used for documenting, planning, communicating, and analyzing states of the EA in respect to a given EA management problem.

The different EA management approaches, discussed in Section 3.3, document useful solutions on different levels of abstraction and implicitly or explicitly describe the corresponding specifics of the solution's application. Two key specifics are the context, delineating facilitators and impediments for the particular solution, and the problem that the solution addresses. Because of the differing levels of abstraction in respect to the solutions also the granularity of the described problems and contexts varies. This leads to a situation, in which a 'classic' contingency-based approach to determine the solutions relevant in a particular context and for a specific problem is not sufficient. Any contingency approach is, as discussed by Pries-Heje and Baskerville in [PHB08], confined to design situations with *symmetric criteria*. This means that each alternative solution is assessed with respect to all criteria, yielding densely populated *decision matrices*. If in contrast selected criteria do not apply to all alternative solutions, but are only relevant for a few or only a single solution, these criteria are called *asymmetric*. Pries-Heje and Baskerville devise in [PHB08] the concept of the *design theory nexus* to address this special quality of such criteria. A design theory nexus links together a set of competing approaches that apply to particular design problems in given contexts. The linked approaches are not reduced to alternative solutions but retain their level of abstraction also with respect to the described context and problem. These competing approaches are design theories in the sense of Walls et al. [WWES92]. A selection of such theories, if consistent, can be used to develop an organization-specific EA management function as well as EA modeling languages contributing thereto. Figure 5.8 outlines how a design theory nexus instantiated on the development of EA management functions is structured.

The MBBs provide re-usable sequences of tasks, that can be applied to different EA modeling languages. Therefore, the building block's descriptions do not directly employ information models and viewpoints, but introduce corresponding variables that are bound during the configuration of the organization-specific EA management function [BMS10h]. Figure 5.9 shows how the tasks contained in an MBB are linked to according variables. The PARTICIPANT VARIABLES are used to designate organization-specific actor roles executing a particular task, while the TRIGGER VARIABLES describe conditions leading to the initiation of the task. For our subsequent considerations, the two highlighted types of variables are of special interest. The INFORMATION MODEL VARIABLES are used to describe the conceptualizations of the EA, on which a task operates. This conceptualization can be arbitrary, but there can also be

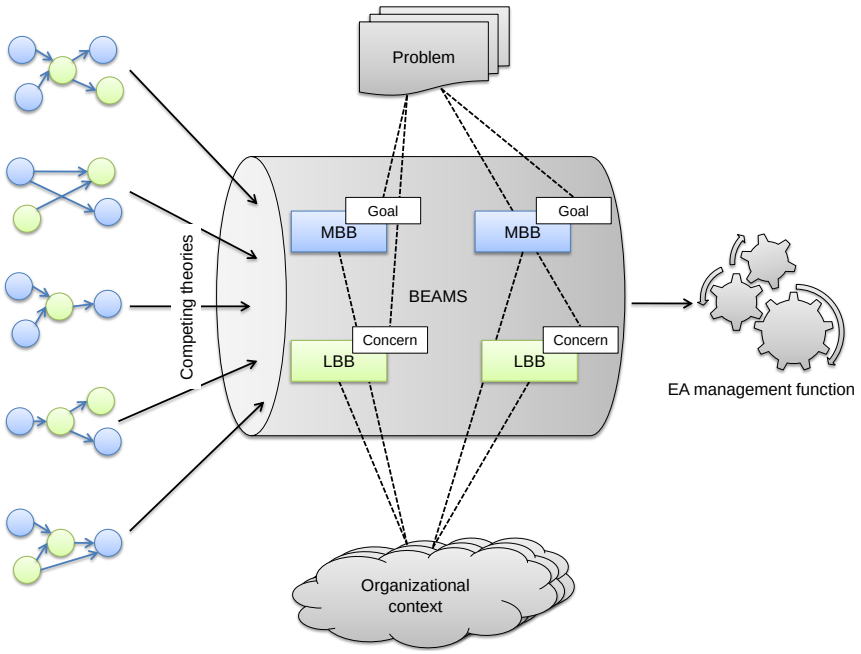


Figure 5.8: Design theory nexus instantiation for EA management functions

tasks that require a particular concept to be part of the conceptualization. For example, tasks that target PROJECTS have to specify that the corresponding information model variable is bound to an information model that encompasses the project concept. The LOWER relationship is used to specify such fact. The associated information model specifies the minimum required conceptualization. This means that any valid value assignment for the variable has to consistently embed the lower information model. Corresponding VIEWPOINT VARIABLES describe, how information according to the selected conceptualizations is communicated to the participants of a task.

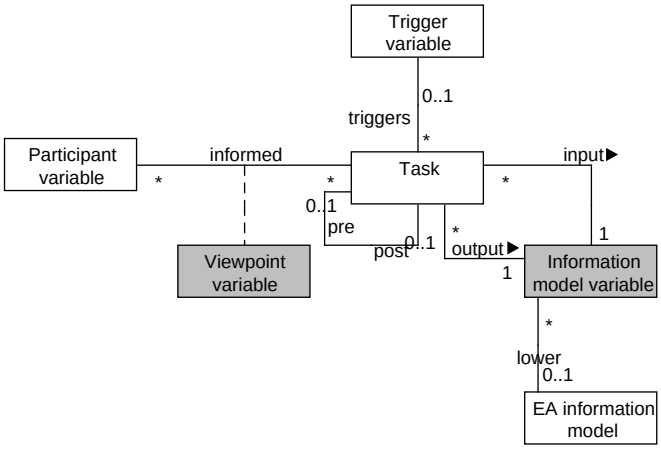


Figure 5.9: General structure of an MBB

During the development of an organization-specific EA management function, the variables of the different types are bound to actual organization-specific roles, triggering conditions, information models, and viewpoints. For the considerations on language development, especially latter two bindings are of interest. The actual information model bound to a specific sequence of tasks has to be designed to cover a particular EA management-relevant problem of the using organization. It nevertheless also must embed in the context of the using organization's embracing understanding of the EA, i.e., must **consistently embed** in the whole organization-specific information model. Further, the used viewpoints must be designed to convey and to make accessible the relevant information, as contained in the corresponding **information model fragments**. Finally, the particular languages used for the method have to be backed by an underlying glossary.

Our development method targets three aspects of EA modeling languages, namely the information model, the semantics function, and the representation or notation function, with corresponding building blocks. The general notion of the LBB is accordingly refined towards the *information model building block* (IBB) supplying an information model, the *viewpoint building block* (VBB) supplying a notation function, and the *glossary building block* (GBB) supplying a semantics function. These building blocks are complemented with techniques facilitating their integration and combination. Using the techniques and building blocks of the different types a consistent set of EA modeling languages can be developed. Figure 5.10 shows the interplay of the different practice-proven building blocks complementing the development method. Elements of the language framework pertaining to the development method are therein marked with ○, while syntactical constituents are marked with △, and notational constituents with □.

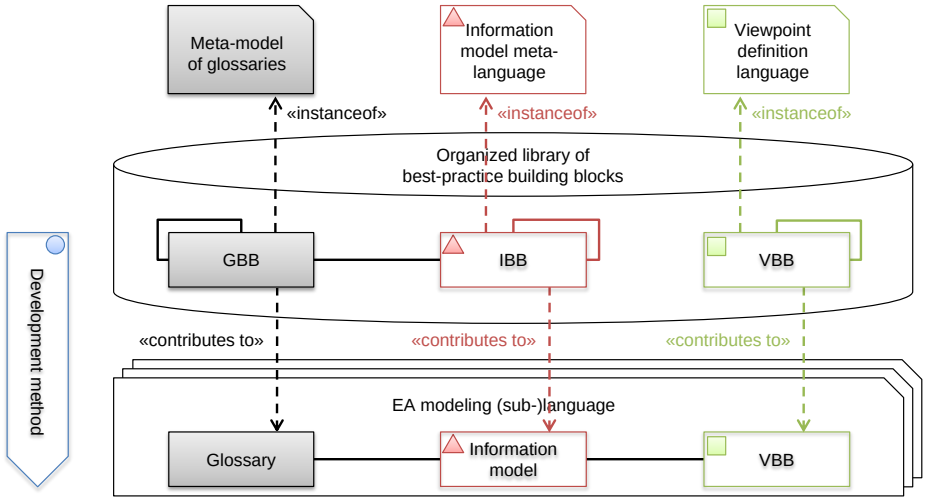


Figure 5.10: Language framework for EA modeling languages

In Chapter 6 we introduce two dedicated meta-languages, namely the *information model meta-language* (IM2L) and the *viewpoint definition language* (VDL) for specifying the syntax and the notation of EA modeling languages. Additionally, we provide a glossary meta-model for describing glossary elements, i.e., for supplying a textual semantics definition. In line

with our previous discussions, we establish a distinction between both types of IBBs: the ones building around identifiable (substantial) architecture elements (**concern IBBs**) and the ones centering around cross-cutting architecture characteristics (**cross-cutting IBBs**).

Definition: Information model building block
An information model building block contains a practice-proven information model reflecting a particular EA concern (concern IBB) or cross-cutting aspect of the EA (cross-cutting IBB).

Figure 5.11 displays the different types of IBBs and shows how they correspond to goals and concerns (depicted with highlighted background). These concepts are summarized under the term **area-of-interest**. An IBB reifies a general **area-of-interest** in the EA [In07], which can either be a **concern** or a **cross-cutting aspect**:

- A **concern** describes a part of the architecture in termes of identifiable and substantial architecture elements.
- A **cross-cutting aspect** targets specific characteristics, e.g. goals [BMS10i] or projects [Bu11c], that can be applied onto an arbitrary part of the architecture.

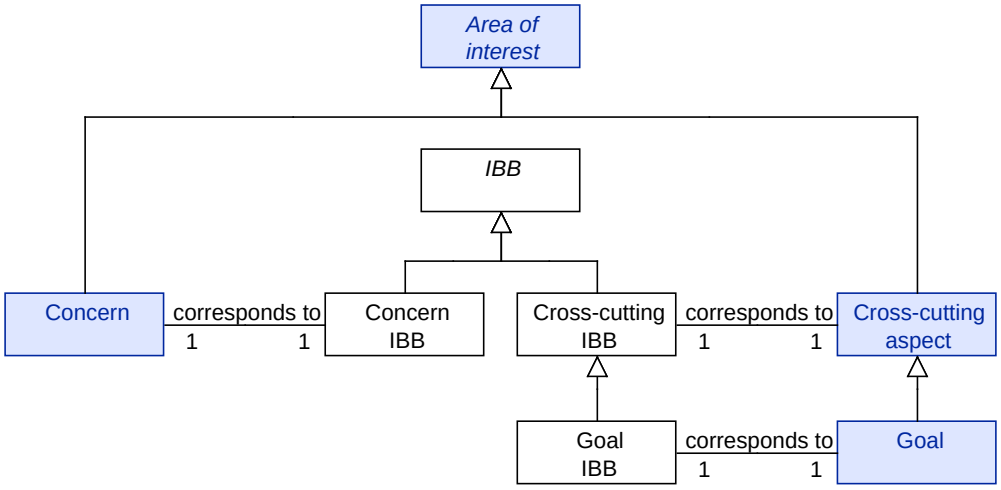


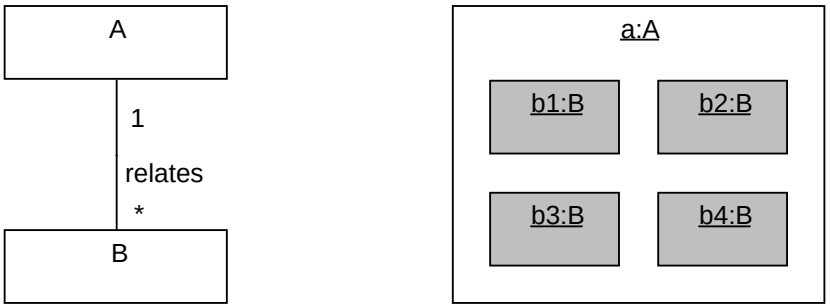
Figure 5.11: Different types of IBBs and contextualizing concepts

An IBB defines (a part of) the syntax of an EA modeling language by specifying the corresponding MODEL ELEMENTS. During the development of an organization-specific information model, cross-cutting IBBs corresponding to goals and questions, projects, or standards are applied to concern IBBs. The different values that an architecture property can take are codified into instantiations of the corresponding MODEL ELEMENT. If this concept, for example, is a PROPERTY[3] these instances can be identified with the range of the corresponding data type, e.g. INTEGER or STRING. Similar considerations also apply for TYPES and RELATIONSHIPS.

[3]We use the different typeset to avoid confusion with the architecture property.

Another important aspect of developing EA modeling languages from practice-proven solutions is the notational aspect. Any language brings along symbolic representations for the MODEL ELEMENTS and their relationships, which must reflect the specific notational expectations of the corresponding users. For our method, this means that the users must be able to select and adapt practice-proven notations. The VBBs provide such practice-proven representation assignments derived from the analyzed EA management approaches. These assignments are of abstract nature, i.e., do not relate to a particular underlying information model, but to an abstract conception of the information to be represented. In Example 5.3 we explain the notion of the viewmodel along the example of the *clustered visualization*, which according to Wittenburg [Wi07] is frequently used to represent EA information[4].

Example 5.3: Viewmodel of a viewpoint. A clustered visualization depicts instances of an outer TYPE A and instances of an inner TYPE B, which are related via a RELATIONSHIP.



The viewmodel supplies enough information to create the visualization but not more, such that the representation assignments constitute a bijective representation function. It would nevertheless be possible to create the visualization in cases, where additional information was supplied. In mathematical terms this means that any particular kind of visualization, i.e., any VBB, can be applied on any information model capable of supplying the **sufficient information**. To support graphical modeling, the underlying information model must not supply additional information, i.e., has to supply only the minimal **necessary information**. Such information model is closely related to the viewmodel of the corresponding VBB. The exact correspondence between the information models and the viewmodel yields a distinction between notation functions and graphical representation functions. While the former relate visualizations to a necessary and sufficient information model, latter functions can build equivalently on any sufficient information model. Any information model that can consistently embed one of the visualization's necessary and sufficient information models is itself sufficient to create the visualization.

[4] Wittenburg calls this type of visualization "cluster map" [Wi07, page 78–79] consistent with his denomination of the graphical representations of EA-related information as "software maps".

Definition: Viewpoint building block
A viewpoint building block defines a single or a set of representation assignments between the model elements in the viewmodel and the visualization elements.

In line with our considerations in [BGS10] we detail on the contribution that a VBB makes with respect to specifying a viewpoint. According to the provided contribution, we can distinguish different types of VBBs as follows:

- **Symbol VBBs** that assign a mapping between a MODEL ELEMENT and a visible VISUALIZATION ELEMENT, i.e., a symbol.
- **Structural VBBs** that assign a complex structure of MODEL ELEMENTS to a set of interrelated VISUALIZATION ELEMENTS. Such building blocks are mostly not self-contained but reference other VBBs as sub-assignments.
- **Decorating VBBs** that assign a mapping between a MODEL ELEMENT and a VISUAL PROPERTY of a VISUALIZATION ELEMENT, which in turn results from a mapping specified in a different VBB.
- **Hybrid VBBs** that specify a combination of the above. Such VBBs can be constituted from other VBBs, e.g. a structural VBB referencing a symbol VBB to which additionally a decorating VBB is applied.

Any viewpoint is configured in terms of at least one structural VBB that determines the overall make-up of the corresponding visualization. This understanding aligns with the principle of the *base map* as established by Wittenburg in [Wi07] used to describe the basic nature of any *software map*. It is nevertheless not necessary for a viewpoint to build on an isolated structural VBB, as also a hybrid VBB containing at least one structural VBB can serve as **base VBB** for a viewpoint.

The GBBs are used to provide a consistent underlying understanding of the concepts employed in the IBBs. Central thereto is the notion of the predicator, which is complemented with a textual description of the according semantics. This description can further link to related predicators, although the relationships established thereby are not typed, but give indications on a corresponding connectedness. A GBB is defined as follows:

Definition: Glossary building block
A glossary building block introduces a predicator with a textual description of the entry's meaning. Further relationships to other GBBs are supplied textually.

The three types of building blocks are used to define information models, viewpoints, and glossaries, of which the former two are bound to corresponding variables of the MBB. Figure 5.12 illustrates this variable binding using the stereotype «c» to designate relationships that result from configuring the EA management function based on building blocks. The actual binding in an EA management function thereby is not arbitrary, but has to obey the following constraints:

- **Pre-post consistency**: For any sequence of two tasks (related via pre-post) the input information model bound to the second task must **consistently embed** into the output information model bound to the first task.

- **Representation consistency**: For any viewpoint that realizes a representation function the underlying viewmodel (information model) must consistently embed into the input information model bound to the corresponding task.
- **Notation consistency**: For any viewpoint that realizes a notation function the underlying viewmodel (information model) must consistently embed into the output information model bound to the corresponding task.
- **Information model consistency**: For any set of methods, i.e., task sequences, their corresponding information models must consistently embed into the organization-specific information model.
- **Lower bound consistency**: For any information model variable, the information model specified as LOWER must consistently embed into the information model supplied as value for the variable.

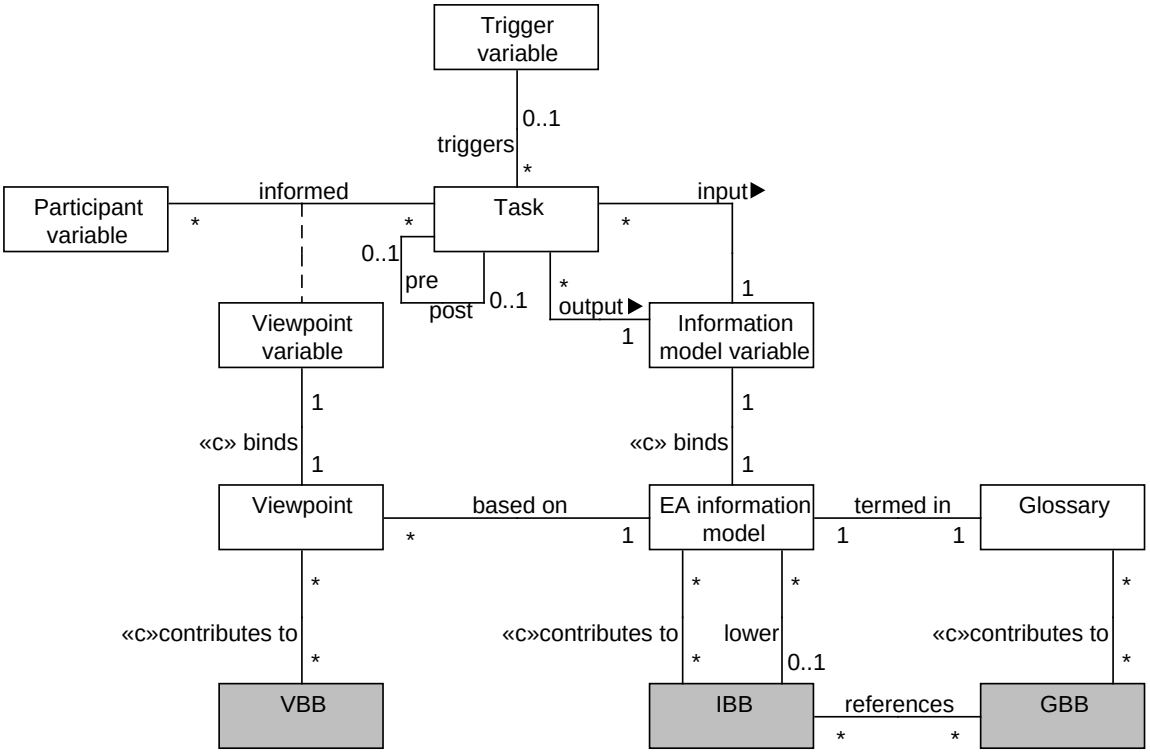


Figure 5.12: Binding of information models and viewpoints to MBB variables

The relevant consistency constraints are operationalized via the **consistent embedding**-relationship. This relationship establishes a partial ordering between information models, such that one information model embeds in another information model, if all EA models instantiating the first model are also valid in respect to the second one. The embedding-relationship is complemented with two more relationships between information models or building blocks thereof. These relationships, according to Dijkman [Di06b] arise from concepts reflecting the

same architecture property. In an EA information model, a concept is represented by a MODEL ELEMENT, which reifies a particular way of perceiving the underlying property, i.e., the phenomenon. For the context of the method, the predicator, i.e., the glossary entry, identifies the corresponding property. Two information models can relate in the following ways:

- They overlap, if they share at least one predicator.
- They conflict with each other, if they overlap, but make contradictory statements in respect to the MODEL ELEMENTS assigned to overlapping predicators.

Consistent embedding and **conflict** are closely related to each other in the sense that for two conflicting information models it is not possible to describe a third information model embedding both. Contrariwise, an embedded information model completely overlaps the embedding one without raising conflicts. In addition to the discussed relevance of the embedding-relationship in respect to configuring the EA management method, embeddings between IBBs can be used to determine possible ways of evolving an EA management function. During the adaptation and evolution of the EA management function, we can proceed from one IBB to an IBB that consistently embeds the first one. The transitivity of the embedding-relationship ensures that this refinement of the covered area-of-interest has minimum impact on the consistencies defined above.

The design theory nexus instantiation maintains a net of IBB-relationships. Figure 5.13 shows the meta-model underlying the relationships and exemplifies a net of IBB-relationships. Each of these relationships is therein set effective by at least one predicator.

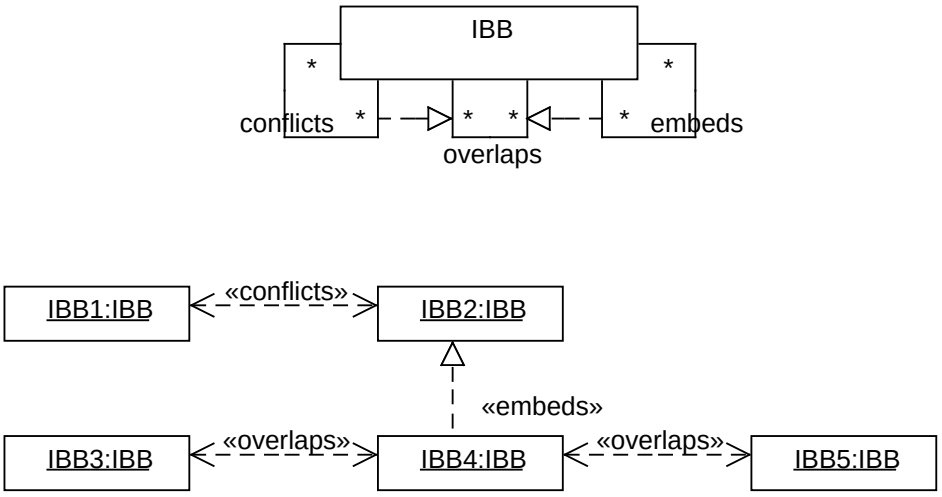


Figure 5.13: Modeling IBB-relationships

´

The IBB-relationships constitute the organizing structure of the method's underlying library of IBBs, which is part of the **organized library** of building blocks. In order to ensure usability of the method, any newly added IBB has to be embedded into the net of IBB-

relationships, if overlapping. It is thereby not sensible to demand the contributors of IBBs to establish these relationships manually, as this would aggravate any extension of the organized library of building blocks. To resolve the aforementioned difficulty, we decide to derive these relationships from the relationships between predicators and model elements as well as the containment of model elements in the IBBs.

5.3 Using the building blocks to develop and evolve EA modeling languages

The development method uses the building blocks and in particular their relationships to each other in three distinct activities to develop and evolve EA modeling languages, namely **information modeling**, **viewpoint definition**, and **glossary adaptation**. Latter activity is elemental, meaning that the users of the method browse through an existing glossary and rename its entries. Thereby, the corresponding predicators are replaced with organization-specific predicators, that consistently inherit all semantic assignments. The activity of information modeling initially starts with an empty information model and iteratively applies the following four steps:

1. Select an EA management-relevant goal to pursue and choose cross-cutting IBB reflecting the goal on an adequate level of abstraction.
2. Select an EA concern on which the goal applies and choose admissible concern IBB reflecting the concern on an adequate level of abstraction.
3. Integrate the cross-cutting IBB with the concern IBB to build a problem-specific information model by specifying the goal-relevant MODEL ELEMENTS.
4. Integrate the problem-specific information model with the already existing information model. (omitted in first iteration)

After above steps have been executed once, an information model is maintained by the method. This information model is used to determine, whether a concern IBB is admissible or not. The latter is the case, when the IBB conflicts with an already integrated one. Another subtlety applies to this step, as the concern descriptions change with the adaptation of the glossary, thus ensuring that the users perform the selection based on their adapted terminology. Both the admissibility checks and the consistent adaptation of the glossary control the evolution of the EA modeling languages and ensure **sustainability (Us2)** of the organization-specific approach.

In the activity of viewpoint definition we apply a two step approach to develop the viewpoint that is used to convey information to a task's participant:

1. Select base VBB and establish admissible links from its viewmodel to the model of available information.
2. Repeat: add VBB to detail existing viewpoint and establish admissible links from the corresponding viewmodel to the model of available information.

Two conceptions in the former approach deserve special attention. Firstly, it can be the case that in some tasks of the EA management function not all information is available. This particularly applies for documentation-related tasks. For such tasks the viewpoint is confined to the actually available information, which means that the VBBs have to be configured against a reduced information model. Secondly, while there are in general only a few restrictions on the VBB to apply for a specific task, the configuration of the corresponding viewpoint is restricted in respect to the intended usage thereof. For a task-participant-relationship, according to which the participant has to provide information about an EA concern, the configured viewpoint must supply representation assignments that constitute a notation function for the information under consideration. The participants must have the possibility to model the corresponding part of the enterprise using the established modeling language. In this sense, any mapping configuration is analyzed in respect to its suitability to maintain a bijective relationship with the underlying information model. This suitability has to be re-evaluated with every adaptation of the information model, as changes to one area-of-interest can influence overlapping areas-of-interest.

The three activities outlined above are employed by the development method for EA management functions in its three major steps: **characterize situation**, and **configure the EA management function**, and **analyze the EA management function**. In the first step, the enterprise architect and the stakeholders designate the organizational environment of the EA management function [BMS10j, Bu10a, Bu11e], supply information on the terminology, used for **glossary adaptation**, and select the relevant EA management-related problems. For these problems, the IBBs are identified and configured during **information modeling**. In the second step actual EA management processes are established. Therein, the activity **viewpoint definition** is executed to develop appropriate viewpoints for participating organizational roles (**participants**). Figure 5.14 gives an overview of the development method. Therein, the activities that are supported by mechanisms and techniques established in this thesis are highlighted.

The overall development method builds on pre-existing management processes, information models, viewpoints, and glossaries. This organization-specific groundwork can be the result of a previous application of the development method, but can also result from other activities that have lead to the development of an EA management, e.g. from IT service management or business process modeling activities. We subsequently assume that an organization-specific information model and an organization-specific glossary constructed from building-blocks already exists. This is especially true for an empty information model and glossary.

5.3.1 Characterize situation

In the first step the enterprise architects describe the organizational context with special emphasis on the relevant stakeholders. This information is used to identify stakeholders that can rise an EA management-related problems which they would like to have addressed. The enterprise architects use their knowledge of the organizational structures to identify personal roles that have interests in the EA or particular parts thereof. The catalog of **participants** and **stakeholders** as supplied with the development method provides a relevant starting point for the identification. For any particular EA stakeholder, the enterprise architects iterate over

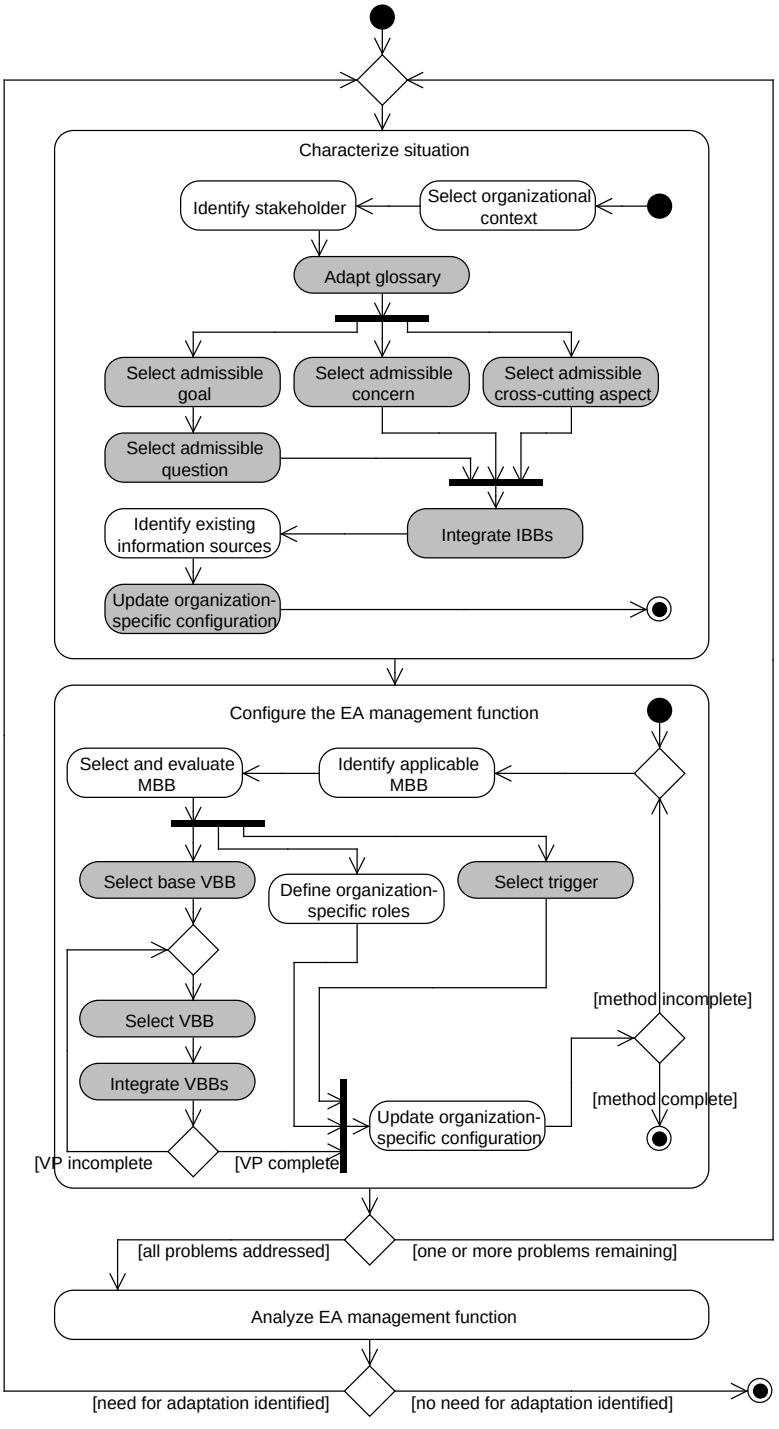


Figure 5.14: Development method for EA management functions

the textual descriptions of the stakeholder's corresponding **areas-of-interest**, identifying the relevant glossary entries and related GBBs.

Example 5.4: Applying BEAMS—Identify stakeholders. The newly appointed enterprise architects of the financial service provider BS&M are faced with a complex IT landscape that has grown over the last years. This landscape, more precisely its contained IT systems, is documented in a *Configuration Management Database* (CMDB) established at BS&M during a former ITIL initiative. Due to the barely controlled growth, recent projects had difficulties to completely understand and predict the impact that changes to the IT landscape have on the business support provided. Against that background, the enterprise architects identify the project portfolio managers as the potential stakeholders of the EA management initiative.

The enterprise architects prepare an interview with the particular stakeholders. Central to the preparation is to revisit the EA management-related problems already raised by the group of people as well as to (re-)familiarize with the terminology used by the group. Based on this knowledge, the enterprise architects can decide to perform glossary adaptations and change the predicators for the relevant architecture conceptualizations to the terms used by the prospective stakeholders. Previous adaptations made for communicating with the particular group, e.g. from a former problem elicitation, can be used to guide the necessary adaptations. Thereby, the enterprise architects have to aim at not to 'revert' the terminology to an old wording. In particular, this means that agreed terms should not be re-mapped to the stakeholder-specific terms, that were previously abandoned. Nevertheless, having this mapping ready can be helpful for interviewing the group of people.

Example 5.5: Applying BEAMS—Adapting the glossary. Working through the textual description of project portfolio managers, the enterprise architects get an overview of the terms relevant to this particular stakeholder group. Key terms in this respect are PROJECT, PROJECT PROPOSAL, and BUSINESS APPLICATION. Reading through the definitional descriptions of the corresponding concepts, the enterprise architects understand, that according to the organization's prevalent terminology the term IT SYSTEM is used to predicate BUSINESS APPLICATIONS. Therefore, the enterprise architects introduce IT SYSTEM as synonym to the organization-specific glossary, replacing the basic term from BEAMS.

Together, the enterprise architects and the stakeholders browse and discuss the different goals that potentially are of relevance. Having identified one particular goal of interest, the center of attention shifts towards the concern in the EA, to which the goal applies. The stakeholders

and the enterprise architects browse the catalog of possible concerns and select one, onto which the previously specified goal should be applied. Both browsing processes are supported by BEAMS and its underlying techniques in different ways:

- The catalog of areas-of-interest (goals, concerns, and cross-cutting aspects) is restricted to areas-of-interest, whose conceptualization can be **consistently embedded** into the already existing conceptualization of the organization. Thereto, the IBBs corresponding to any particular area-of-interest are compared with the organization's existing information model. If one IBB cannot be embedded, it is removed from the catalog of admissible areas-of-interest. In all other cases, the area-of-interest is marked as fully admissible.
- The catalog of areas-of-interest is prioritized according to the coverage of potentially relevant glossary entries. This means that for any area-of-interest, the corresponding IBB, more precisely the contained model elements are analyzed. For any model element that uses a **predicator** relevant to the stakeholders or a predicator synonymous to such predicator, the area-of-interest is ranked higher. The areas-of-interest which correspond to the largest number of relevant concepts (as represented by predicators) are prioritized highest.
- The catalog of areas-of-interest is presented using the accepted terminology in the enterprise. For concepts that are yet not covered by predicators in the glossary of the existing EA management function, the adapted terms supplied by the enterprise architects during stakeholder identification are used.

The restricted and prioritized catalog is presented by the enterprise architects during the discussion with the stakeholders.

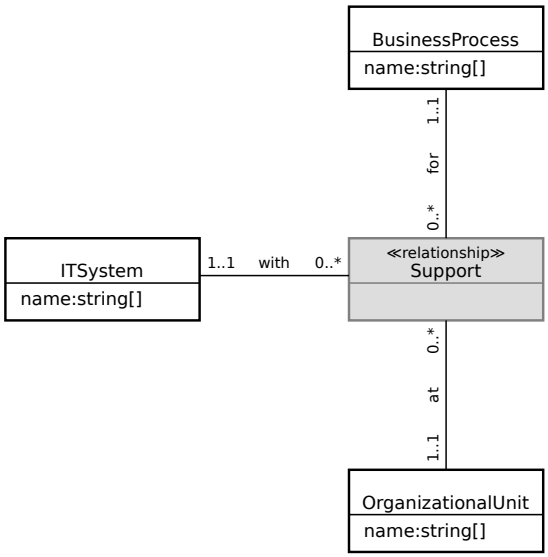
Example 5.6: Applying BEAMS—Selecting goal, concern, and cross-cutting aspect. Based on the selection of the stakeholder and thereby of the relevant glossary entries, BEAMS prioritizes the goals "ensure compliance", "improve project execution", and "increase transparency" highest. Former goal yields several references to the concept IT SYSTEM, whereas latter goals relate to PROJECT and PROJECT PROPOSAL. Reading through the details, the team of project portfolio managers opts for achieving transparency first.

Regarding the potentially interesting concerns, BEAMS prioritizes the IT SYSTEM-related ones. With a larger number of such concerns, the enterprise architects ask the project portfolio managers to specify another architectural layer, which is of interest with respect to the IT Systems. Selecting the "business & organization"-layer five concerns are proposed:

- IT systems support Business Processes at Organizational Units
- IT systems support Business Processes for Products
- IT systems support Business Processes for Products at Organizational Units

- Organizational Units host IT Systems
- Organizational Units use IT Systems

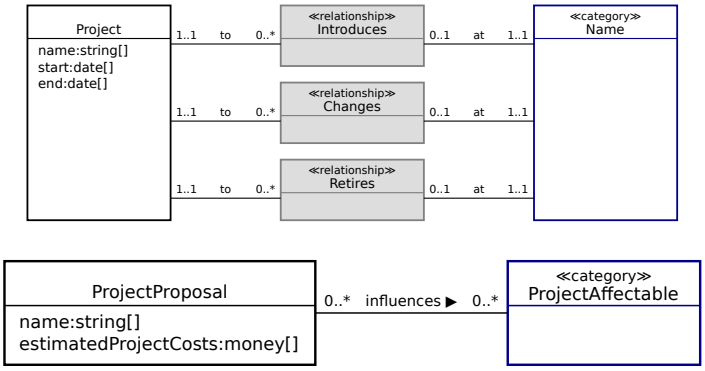
After some discussion, the project portfolio managers decide for "IT systems support Business Processes at Organizational Units", which according to their opinion best reflects the kind of business impact, that they would like to understand. Further, the enterprise architects are able to elicit that both current projects as well as the current project proposals are of interest for the project portfolio managers.



The stakeholders select one goal, concern, and (optional) cross-cutting aspects, which constitute a single problem that the EA management process is intended to address. The enterprise architects can refine the area-of-interest describe by any of the elements or the combination thereof by applying a **filter function**. Thereto, **specialization by constraint** is applied to specify onto which part of the organization the corresponding conceptualization applies. This information is elicited by the enterprise architects during stakeholder interviews. In addition, the enterprise architects define the time-reference for the described problem, if not already determined by the nature of the goal or the involved cross-cutting aspects. Finally, the enterprise architects have to understand, which parts of the corresponding architecture information has to be historized.

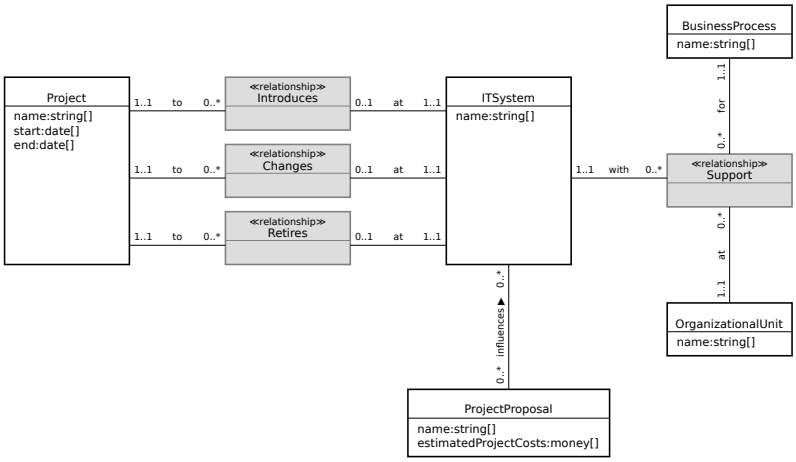
Example 5.7: Applying BEAMS—Selecting IBBs. Based on the information about the role of the project portfolio managers in projects, the enterprise architects decide to use the more detailed cross-cutting IBB "Projects introduce, change, and retire Architecture Elements" for model-

ing the impact of projects, whereas for project planning the more abstract IBB "Project proposals affect Architecture Elements" is chosen. This yields three IBBs, of which the following two are cross-cutting.



In the next step the enterprise architects choose a particular operationalization of the goal into a question reified in a corresponding IBB. The selection of this question is supported by BEAMS indicating how different ways of operationalization result in different information demands, i.e., are less or more complex to maintain. After the operationalization has been chosen, the enterprise architects integrate the concern IBB and the cross-cutting IBBs into a problem-specific information model fragment. This fragment is subsequently embedded into the already existing information model of the organization. This embedding is a **consistent embedding** in terms of the previously introduced understanding.

Example 5.8: Applying BEAMS—Integrating IBBs. With "increasing transparency" as goal of EA management, no operationalization has to be chosen. This reduces the step to integration of the IBBs. Knowing that the project's effects on the IT SYSTEMS are to be considered, the enterprise architects chose to designate the IT system as being PROJECT AFFECTABLE. The integration results in the following information model.



The enterprise architects repeat above steps to compile a comprehensive, organization-specific information model addressing the EA management-related problems raised by the stakeholders of the EA management function. The organization-specific configuration stores the information model as well as the glossary. The embedding of a new information model fragment into the organization's specific information model can necessitate further adaptations to the already defined EA management function. A new information model fragment can change the nature of the information model that valid EA descriptions according to the old information model are not valid according to the new model (**weak embedding**). This can e.g. be caused by the introduction of an additional mandatory relationship end. After the activity **characterize situation** is finished, the organization-specific information model as well as a complementing organization-specific glossary are established and provide the underpinning of the EA management function.

5.3.2 Configure the EA management function

The configuration of the actual management process details which tasks are executed in EA management by whom on which information and which events cause the tasks to be triggered. Most of these concepts have a strong method focus and are hence only of limited interest for the development of the EA modeling languages. The viewpoints used to convey information to the participants of EA management are nevertheless important. From the perspective of EA information, every single management task can be understood from the perspective of pre-conditions and post-conditions that specify which information is available before the task is executed and which information is provided after task execution. Rephrasing this in terms of our framework from Section 6.1, we can say that any task takes one area-of-interest as its input and provides another area-of-interest as its output[5]. Following example explains this relationship between tasks and areas-of-interest represented in information models.

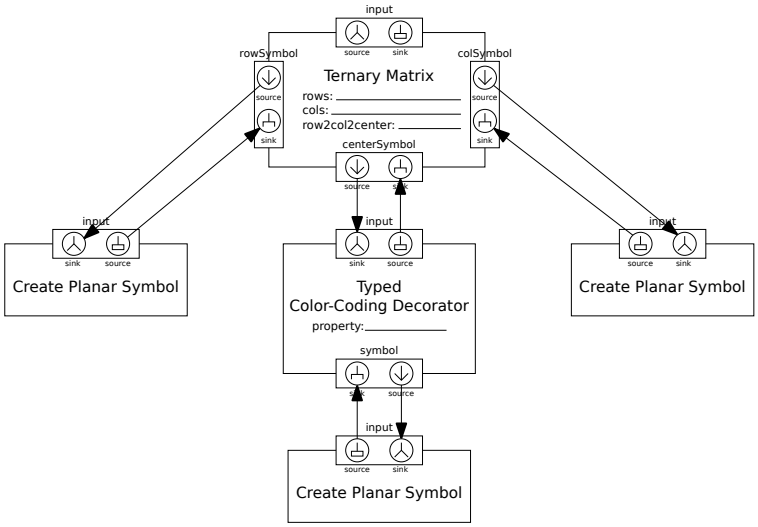
Example 5.9: Applying BEAMS—Identifying viewpoints. Having identified the BUSINESS PROCESSES as necessary input for the EA management, a process for gathering information about the supported processes and the used IT SYSTEMS is set up. Central thereto is an interview, in which the enterprise architects asks the process owners about the systems used to support their processes. As part of the interview one viewpoint for modeling the process support is needed. This viewpoint is textual in its nature, realized in an office document.

In a later task the project portfolio managers are informed over the effect of their PROJECTS on the current application landscape. The enterprise architects chose to employ a graphical viewpoint for doing so, representing both the business support provided by the IT SYSTEMS as well as the influence that the projects exert on the IT systems.

[5] Precisely, we should say that the input as well as the output is information according to one area-of-interest. We abstain from that complicated terminology to facilitate readability.

The viewpoints used by the participants of a task can be distinguished into two types, namely ones realizing a **notation function** and ones realizing a **representation function**. This distinction influences the configuration of the viewpoints assigned to the task and its corresponding participants. BEAMS supports the actual configuration of the viewpoint in a stepwise manner. At first, the user has to specify a **structural VBB** or **hybrid VBB** containing a structural VBB as base VBB for the viewpoint. Stepwise, the user can add other VBBs to the base VBB, thereby extending the viewpoint. Syntactic compatibility rules ensure that the viewpoint configuration remains executable, with respect to an underlying viewmodel. This viewmodel is evolved in parallel, i.e., with every extension new model elements are added to the viewmodel. Complementing the configuration of the 'raw' viewpoint, the user can also supply values for the visual parameters of the VBBs, as e.g. colors or symbol types. Following example illustrates the composition of a hybrid VBB from smaller constituents.

Example 5.10: Applying BEAMS—Designing hybrid VBBs. The application landscape, i.e., the BUSINESS PROCESS support provided by the IT SYSTEMS at particular ORGANIZATIONAL UNITS is displayed using a "ternary matrix" VBB as base VBB. Several "create planar symbol" VBBs are used to display the symbols on both axis as well as in the middle of the map. In addition a "typed color-coding decorator" VBB is applied on the "create planar symbol" VBB to color the rectangles representing the IT SYSTEMS. Thereby, the systems are distinguished by type between "affected" and "not affected". The configured hybrid VBB looks as follows.



After the configuration of the hybrid VBB is finished, the user completes the viewpoint by binding the informational parameters to corresponding model elements in the input information model. In this binding, the user specifies filter functions via specializations by constraints and aggregations, as long as they are admissible with respect to the nature of the viewpoint, i.e., the viewpoint realizing a notation function or representation function. This means that

with any binding, it has to be checked, if and how the configured viewmodel can be consistently embedded into the available information model. We distinguish two types of embedding, namely:

- **updating embedding** which means that the viewpoint can be used to update the underlying EA description and
- **reading embedding** which means that the viewpoint can be used to display information from the underlying EA description.

If a notation function is expected, the checking has to ensure that the viewpoint is defined in a manner that its viewmodel supports updating embedding. A representation function is realized by any viewpoint that supports reading embedding of its viewmodel. The actual quality of a viewpoint, i.e., the type of embedding (**reading embedding** or **updating embedding**), has to be re-evaluated, whenever a new information model fragment is integrated in the organization-specific information model. It can be the case that due to an added mandatory property or relationship end, a formerly updating embedding is reverted to a reading embedding. In this case, the enterprise architect is informed that the expected quality of the viewpoint (notation function) is not longer given and the viewpoint definition has to be adapted following the steps outlined above.

5.3.3 Reverse engineering the configuration of an existing EA management function

In the light of above considerations on configuration of an EA modeling language being an iterative endeavor, in which the previously specified information model is taken as basis, no specific adaptation method for EA modeling languages are needed. The development method according to its nature already supplied the relevant mechanisms and steps. Nevertheless, when it comes to adapting EA modeling languages from an existing EA management function developed without building blocks, further steps have to be taken, namely **reverse engineer information model fragments**, **consolidate glossary**, and **integrate information model fragments**. These steps are applied to reverse engineer the building block-based configuration that best approximates the modeling languages underlying a current EA management function.

Reverse engineer information model fragments

In the first step, the information model fragments underlying the current EA management-function are elicited from the source of information, available in the organization. During the elicitation, the users have to assume that each of the different sources provides only a partial image on the overall architecture of the enterprise, which especially means that the underlying conceptualizations are fragmentary and potentially contradicting. Two primary sources of information are taken into account, during the reverse engineering:

- existing model repositories, databases as well as other tabular data resources, and
- existing visualizations of the EA or parts thereof.

For the first kind of sources, the corresponding schemata are investigated either from explicit specifications or by analyzing the corresponding instance data. From the schemata one or more information models covering the relevant concepts can be derived. The termini used to labeling the concepts (tables, columns, etc.) are regarded a predicators according to the framework provided later in Section 6.1. Per source of information, these predicators constitute a single glossary. During reverse engineering these sources, integrating the resulting glossaries is not advisable, as this can compromise the semantics underlying the data sources.

With respect to the second type of sources, the VBBs are used to facilitate reverse engineering. Understanding the basic make-up of a visualization, the matching structural VBB is chosen as base VBB. In further steps, for each of the required sub-transformations of the base VBB corresponding VBBs are instantiated, thereby stepwise composing the viewpoint, more particular the according hybrid VBB, under consideration. Any VBB commits to a viewmodel, which is in turn elicited from the hybrid VBB. The viewmodel is in the next step refined to an information model, by linking the un-predicated concepts to corresponding glossary entries and GBBs. In doing so, especially a legend supplied along the visualization can be helpful. In the case that no such legend is available—a case frequent according to our observations [Bu09d]—the authors and users of the visualization have to be interviewed to supply additional information for interpreting the symbols. Further, the usage context of the visualization (read vs. update) is elicited.

Consolidate glossary

In this step the different glossaries, derived during reverse engineering of information model fragments, are consolidated. Understanding the terms currently used as synonyms for each other or for predicators provided in a GBB, possible integration points are derived. If different information sources yield different terms mapping to the same BEAMS predicator, the users have to decide, which of these predications should be used in the EA management function. Criteria to be taken into account are on the one hand the number of instances of the concept under consideration, available in the according information sources, and on the other hand the number of stakeholders using the particular term. Both numbers are therein used as indications towards the familiarity with the term in the linguistic community of the organization.

In a similar vein, the method users have to resolve conflicts with respect to homonyms used in predicating concepts in different information sources. Two central challenges in this context are the analysis for homonyms and the identification of potential namespace conflicts between different linguistic communities. Discussing with proponents of the involved linguistic communities, the method users define a resolution by renaming all of the conflicting terms but one. Making optimal re-use of the practice-proven building blocks in the renaming, the terminology of the GBBs should be adopted as far as possible. For any renamed predicator as well as for any term that was not found in the GBBs, corresponding definitory statements interlinked with the other predicators have to be developed in close cooperation with representatives from the according linguistic community. Thereby, the entries of an organization-specific glossary are developed.

Integrate information model fragments

In the final step the different information model fragments are integrated into a comprehensive information model. During this integration, we have to ensure consistency of the resulting model. This challenge is addressed by the integration and aggregation operations described in Section 6.2.3. During any particular integration, the source of the fragment to integrate is taken into account: while for visualizations (viewpoints) used for reading access only, arbitrary integration mechanisms can be chosen, updating viewpoints and technical information sources are integrated via **updating embedding**. This means that as far as possible the viewmodels and information models should be integrated without adaptation. If, nevertheless, due to one of these sources taking a more abstract perspective on the EA, integration can only be achieved based on aggregating existing information, the corresponding source is marked as deprecated. Thereby, the method users indicate that the corresponding type of source is not longer meant to be used and has to be replaced as part of the consistent operation of an integrated EA managementfunction.

The three steps outlined above can be applied to understand how current EA management endeavors and activities interrelate and how they embed into the embracing context of an organization-specific EA management function. In an optional step, reverse-engineering can proceed to forward engineering, i.e., to development, binding the hybrid VBBs elicited in the first step to the integrated information model resulting from the last step. Thereby, the method users ensure that the prevalent visualizations are supported based on the integrated information model of the organization. This step nevertheless does not need to be taken and alternatively, the development method can start with identifying the stakeholders of the EA management function. While omitting the optional step clearly helps to reduce the effort spent, it can cause the enterprise architects to 'reinvent the wheel' during the development method, whenever a new viewpoint is defined. Moreover, the creation of new viewpoints superseding already existing ones can cause resentments against the EA management function from the side of the users of the prevalent EA management-related facilities.

5.4 Developing and evolving the organized library of building blocks

The **organized library** of building blocks is the central knowledge base on which the development method of BEAMS builds. The development and evolution of this library is the critical success factor for the development method. **BEAMS administrators** are stakeholders concerning with the development of this knowledge base. They seek to incorporate novel solutions for EA management-related problems into the organized library, once these solutions have proven to be applicable.

In a first step of administration, the BEAMS administrators **identify a new EA modeling language**. In any EA management project or endeavor, EA modeling languages are created and extended. To understand what specific EA management-related problem a language aims at, the BEAMS administrators analyze the design rationale of the language. In case, no such rationale is provided, the administrators conduct interviews with the language designers as well as with the stakeholders rising the language demands. Thereby, the EA management-related

problems have to be elicited. During the elicitation, the BEAMS administrators establish a mapping between the organization's terminology and the terms presented in the GBBs. In particular, synonyms and homonyms have to be identified to facilitate the translation between the different terminologies. Terms, that are not represented by GBBs, have to be converted into GBBs by supplying a description and linking this to the set of existing GBBs. Based on the terminological mapping, the BEAMS administrators can translate the EA management-related problem using the terms of BEAMS. Subsequently, the identified problem is decomposed into the cross-cutting, e.g. goal-dependent aspect, and the concern, i.e., the structural aspect. The administrators of BEAMS thereto revisit the collection of cross-cutting aspects as well as the one of known concerns, in order to determine, whether the cross-cutting aspect or the concern are related to already documented areas-of-interest. In the sense of Dijkman et al. [DQS08], different types of relationships between the areas-of-interest can be identified. Especially **overlap** is of key interest:

- A complete overlap in respect to an existing concern renders the new solution irrelevant for the evolution of BEAMS.
- A partial overlap in respect to an existing concern makes the corresponding concern a good starting point for eliciting the IBB.
- A complete or partial overlap in respect to an existing cross-cutting aspect, in particular a goal, requires the BEAMS administrators to familiarize themselves with the existing IBBs covering the aspects.

If none of above relationships can be established, both concern and cross-cutting aspects are assumed to be completely new.

In a second step, the BEAMS administrators **document pattern candidates** for EA management. The languages conceptualization is thereto reverse-engineered from the visualizations, repository models, or information resources. To facilitate a consistent and redundancy-free modeling of the conceptualizations, a distinction between rigid and non-rigid types as well as dispersive and substantial ones can be employed to devise an ontologically sound information model. In a similar vein, the viewpoint of the modeling language is documented as a V-pattern that exemplifies the visualizations derived from the viewpoint. The BEAMS administrators use the terminology of BEAMS to describe both types of patterns, i.e., to establish links to the relevant GBBs.

The pattern candidates are documented in a candidate catalog similar to the pattern catalog for EA management [Ch10]. The candidate catalog is administered using the methods described by Ernst in [Er10]. In particular, the user community of BEAMS is involved to find other application cases for the pattern candidates or for trying the candidates to address a related problem. Once, three or more application cases have been gathered, the pattern candidate matures into a pattern. New as well as newly matured pattern provide the basis for the subsequent steps of the administration method. In these steps, the BEAMS administrators decompose the patterns into building blocks for the organized library.

The BEAMS administrators **analyze the V-pattern** and understand from which VBBs, they are composed. Therefore, the reverse-engineering method as described in Section 5.3 can be applied. Whenever the viewpoint cannot be explained based on the existing VBBs, the administrators add an additional VBB to the knowledge base. Therefore, the visualization ele-

ments in the exemplary visualization have to be determined and the corresponding viewmodel has to be established. Central thereto is the understanding of the bijective mapping between the visualization elements and the model elements of the viewmodel. Both the visualization elements and the viewmodel can be used to determine, whether the newfound hybrid VBB should be incorporated as a whole into the organized library, or not. Following indications hint towards the inseparable nature of the VBB:

- The VBB uses or is parameterized with visualization elements that are used in no other VBB or cannot be supplied as parameter values to any existing VBB.
- The VBB uses an not documented combination of visualization rules. The set of employed rules can further not be decomposed into sets used by existing VBBs.
- The VBB is parameterized with an unprecedented combination of viewmodel parameters. Especially consistency rules between these parameters can provide an indication towards the uniqueness of the VBB, e.g. when the rules cannot be composed from rules as employed in existing VBBs.

During this activity, typical decorations of the viewpoint as color-coding or resizing are removed to understand the basic nature of the analyzed viewpoint. The remaining VBB is added to the organized library of building blocks.

In another activity, the BEAMS administrators **analyze the I-pattern**. In case of overlapping concepts the analysis greatly benefits from identifying relationships to existing IBBs. Therefore, equivalently predicated model element are analyzed for **embedding**-relationships to model elements in IBBs. Of particular interest are such relationships, where a model element from BEAMS embeds an element in the I-pattern. In such case, the BEAMS administrators 'subtract' the existing element from the embedding one. For an universal type this means that all properties and relationship ends that can be explained by the universal type from BEAMS are removed and the remaining type is analyzed. Central to this analysis is to determine, whether the remaining type is dispersive or substantial. It is very likely, that at this point a dispersive type reifying a not support cross-cutting aspect remains. If it is not possible to describe the remaining model elements with a domain terminology, i.e., if there is no native predication for the concept, the BEAMS administrators incorporate the not-reduced I-pattern as IBB into the knowledge base.

5.5 Summary

In this chapter we introduced the basic structure of an EA management function constituted from methods and languages. The former describe the tasks to be taken, the participants responsible for these tasks, and the triggering conditions initiating a sequence of tasks. The languages are employed to convey information to the participants or to facilitate the participants in making their EA related knowledge explicit. In an actual EA management function the languages are aligned to the information demands of the stakeholders, more precisely to the EA management-related problems that the stakeholders seek to be addressed. Furthermore, an integrated EA management function demands the different languages to be consistent and requires that any information processed in one task is gathered in a previous one.

As means to design an organization-specific EA management function we introduce different types of building blocks that contribute practice-proven solutions for recurring problems in EA management applying in distinct contexts. The **method building blocks** provide prescriptions for the management process on an abstract level, where:

- **participant variables** act as placeholders for organization-specific roles,
- **viewpoint variables** replace actual stakeholder-specific viewpoints,
- **trigger variables** designate configurable triggering conditions, and
- **information model variables** act as placeholders for the EA conceptualization, on which the tasks operate.

Both the viewpoint variables and the information model variables are relevant in respect to EA modeling languages as they refer to the language's notation and syntax, respectively. Corresponding building blocks, the **viewpoint building blocks** and the **information model building blocks**, are used to define the viewpoints and the information models underlying an organization-specific EA management function. The actual configuration of viewpoints and information models is nevertheless not arbitrary, but has to fulfill different kinds of consistency constraints, namely **pre-post consistency**, **representation consistency**, **notation consistency**, **information model consistency**, and **lower bound consistency**. The development method has to ensure that these different types of constraints are fulfilled in a configured EA management function. Thereto, it builds on the notion of **consistent embedding** between **areas-of-interest** represented in information model fragments. We further distinguish two types of embedding as:

- **updating embedding**, where information corresponding to the embedded information model can be read from and written to the embedding information model, and
- **reading embedding**, where information corresponding to the embedded information model can be read from the embedding information model.

Based on techniques to determine whether and how an information model fragment embeds, the enterprise architects can leverage the development method for creating an organization-specific EA management function based on a consistent set of EA modeling languages. This particularly addresses the challenges of consistently integrating IBBs into a comprehensive information model as well as of aggregating EA information for visualization and visual modeling. In addition, techniques for relating predicators and terms in different glossaries have to be provided to facilitate the identification of synonyms as well as the resolution of homonyms. In Chapter 6 we establish and formally define these techniques based on two dedicated metalanguages for EA modeling.

The final section of this chapter discusses how the organized library of building blocks is developed and evolved. Central thereto is the research method as already outlined in Section 2.3. In addition, three different types of relationships (**overlap**, **conflict** and **consistent embedding**) are used to organize the IBBs for facilitating their use in the development method.

We use crude tools to fashion better tools, and then our better tools to fashion more precise tools, and so on.

Chairman Sheng-ji Yang, Looking God in the Eye – Sid Meier's Alpha Centauri

CHAPTER 6

Foundations of BEAMS

The organized library of language building blocks supplies three different types of building blocks that are used to develop organization-specific EA modeling languages. These modeling languages relate to the described part of the reality, i.e., the **universe of discourse**. The work of Guizzardi [Gu05] provides the basis for a formal way to understand this relationship, which is subsequently called **ontological commitment**. Central thereto is the **conceptualization** that underlies a modeling language. A conceptualization represents the intra-subjective structuring principles implicitly used by modelers during the creation of a model describing a corresponding part of reality. Any conceptualization useful to create communicable descriptions of a universe of discourse is reified in a modeling language. Figure 6.1 gives an overview of the correspondences between language, conceptualization, universe of discourse, and model. It further complements the general framework correspondences with the ones between the concepts of the ISO standard 42010 [In07].

The quality criterion **Cp1 multi-perspectivity support** adds a complexity, when simultaneously more than one conceptualization is employed. In Section 6.1 we further the framework of Buckl et al. [BKS10] to establish a sound and formal basis for defining consistency between different conceptualizations. Based on the extended framework, we establish the foundations for the organized library of building blocks and introduce two dedicated meta-languages. The IM2L is used to describe information models as well as fragments and building blocks thereof. It supplies particular concepts for modeling EAs but also supplies a formal understanding of the **consistent embedding** introduced in Chapter 5. The VDL provides concepts and techniques to operationalize the transformation-based notion of the viewpoint in line with Section 4.3.5. Via specialized techniques, the VDL also supports the integration of VBB into a comprehensive viewpoint definition. A meta-model for describing glossaries and building blocks thereof concludes the exposition of foundations in Section 6.4.

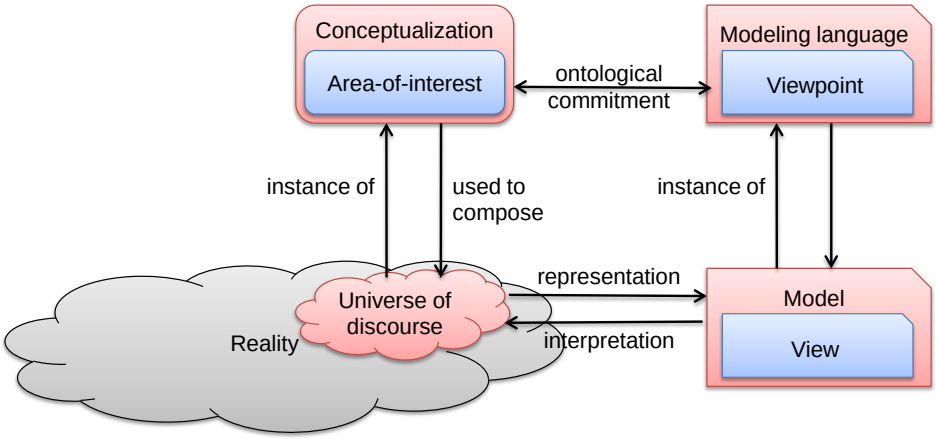


Figure 6.1: Conceptual framework for supporting multi-viewpoint EA management

6.1 A framework for multi-viewpoint EA modeling languages

A conceptualization represents a set of structuring principles used by modelers during the creation of a model of a corresponding part of reality [Gu05]. The universe of discourse thereby does not need to be confined to a single world, but can represent different *possible worlds*. Guizzardi's formal set-theoretic definition of a conceptualization reflects this fact.

Definition: Conceptualization
A conceptualization $z \in \mathcal{Z}$ is defined as a tuple consisting of

- the set of possible worlds $\mathcal{K}_z$,
- the set of covered individuals[a] $\mathcal{I}_z$, and
- the set of conceptual relationships $\mathcal{C}_z \subseteq \bigcup_{n \in \mathbb{N}} (\mathcal{K}_z \to \mathbb{P}(\mathcal{I}_z^n))$.

[a] Both substantial objects and relationship objects are considered "individuals" here.

The set $\mathcal{C}_z$ requires additional explanation. In contrast to the relationship objects contained in $\mathcal{I}_z$ the relationships in the set $\mathcal{C}_z$ represent *concepts*, to which an instance or a tuple of instances belongs. Example 6.1 helps to clarify this distinction.

Example 6.1: Conceptual relationships vs. relationships. In one particular reality, i.e., possible world, the contained individuals are nominated as "my car", "me", and "my ownership", of which the former two are substantial objects, whereas the latter is a relationship object. The concepts are {"my car"}, {"me"}, and {⟨"my car", "my ownership", "me"⟩}, mirroring predicators as "Audi A3", "human", and "ownership".

In this definition the term *concept* is understood both *contextually* and *by extension*. This means that a concept is defined by its corresponding individuals in relation to the *possible worlds* in the universe of discourse. In EA management possible worlds are the different architectural states as discussed in Chapter 5. The notion of *concept* further deserves attention from a type-oriented perspective. Thereto, we build on Guizzardi's postulate 4.1 (cf. [Gu05, pages 99–101]) stating that "each individual must be an instance of a [concept]". The **extension function** maps a concept given in the context of a particular world to the set of all its corresponding individuals. We write $\mathbb{E}_\kappa[c]$ to denote the extension function applied onto a concept $c \in \mathcal{C}_z$ in the context of a world $\kappa \in \mathcal{K}_z$. For any particular conceptualization z, the following condition holds:

$$\forall c \in \mathcal{C}_z, \kappa \in \mathcal{K}_z : \mathbb{E}_\kappa[c] \subseteq \mathcal{I}_z.$$

Subsequently, we revisit the meta-properties as well as meta-relationships discussed in Section 4.3.2 and redefine them using the extension function. For reasons of convenience we introduce two operators for quantifying over the corresponding *possible worlds*. Let ψ_κ be a predicate over κ as free variable, we write:

- $\Box_\kappa \psi_\kappa$ as abbreviation for $\forall \kappa \in \mathcal{K}_z : \psi_\kappa$ and
- $\Diamond_\kappa \psi_\kappa$ as abbreviation for $\exists \kappa \in \mathcal{K}_z : \psi_\kappa$.

We use the extension function to operationalize the meta-property of *rigidity* of a concept $c \in \mathcal{C}_z$ in a particular conceptualization:

- The concept is **rigid**, if $\Box_{\kappa,\kappa'}\ \forall i \in \mathbb{E}_\kappa[c] : i \in \mathbb{E}_{\kappa'}[c]$ holds.
- The concept is **non-rigid**, if $\Diamond_{\kappa,\kappa'}\ \exists i \in \mathbb{E}_\kappa[c] : i \notin \mathbb{E}_{\kappa'}[c]$ holds. In particular, we distinguish between two specific kinds of non-rigid concepts as follows:
  - The concept is **anti-rigid**, if $\Box_\kappa \Diamond_{\kappa'}\ \forall i \in \mathbb{E}_\kappa[u] : i \notin \mathbb{E}_{\kappa'}[u])$ holds.
  - The concept is **semi-rigid**, if it is non-rigid but not anti-rigid.

For the notion of identity, we take a related approach. We call a concept c **identity providing**, if we can supply a binary predicate Λ^c for which the following holds:

$$\Box_{\kappa,\kappa'}\ \forall i \in \mathbb{E}_\kappa[c], j \in \mathbb{E}_{\kappa'}[c] : \Lambda^c_{\kappa,\kappa'}(i,j) \Leftrightarrow i = j.$$

In line with Guizzardi [Gu05, page 100] we introduce the meta-relationships **specialization** between different concepts. A concept c_2 specializes concept c_1, if for all admissible contexts $\mathcal{K}$ the extension of c_1 is a subset of the one of c_2. In terms of the extension function, we can write specialization $<: \subseteq \mathcal{C} \times \mathcal{C}$ as

$$c_2 <: c_1 \Leftarrow \Box_\kappa\ \mathbb{E}_\kappa[c_2] \subseteq \mathbb{E}_\kappa[c_1].$$

In respect to the arity of the concept, we distinguish two different types of concepts. Concepts, whose extension consists of single individuals, are according to Guizzardi called *substantial concepts*, whereas n-ary concepts with $n > 1$ are called *relationship concepts*. This distinction is of additional importance regarding the specialization meta-relationships, as it is confined to specializations between concepts of the same arity.

A modeling language l can be used to reify a conceptualization in a tuple $\langle \mathcal{S}_l, \iota_l, n_l \rangle$ [Gu05, pages 81–82]. It consists of syntax $\mathcal{S}_l$, a semantic interpretation function i_l and a notation function n_l. The semantic interpretation function is defined as $\iota_l : \mathcal{S}_l \rightarrow \mathcal{I}_l \cup \mathcal{C}_l$, where $\mathcal{I}_l$ denotes the set of covered individuals and $\mathcal{C}_l$ denotes the set of supported concepts. The **ontological commitment** between a language l and a conceptualization z operationalizes in line with Buckl et al. [BKS10] as follows:

$$l \sim z \Leftrightarrow \mathcal{I}_l = \mathcal{I}_z \wedge \mathcal{C}_l = \mathcal{C}_z.$$

Above formalization allows to reason on quality criterion **Cp1 multi-perspectivity support**. Therefore, we have to extend the concepts *conceptualization* and *modeling language* towards the corresponding terms of the ISO standard 42010 [In07], namely **area-of-interest** and **viewpoint**. According to Buckl et al. [BKS10], latter concepts are the as *intensional* counterparts of the former ones. Central to this perspective is the notion of the **type**, which is the intensional counterpart to the *concept*. A type defines a particular set of **instances** in a given context, whereas the concept relates a set of individuals in a given context. We exemplify the underlying correspondences between the intensional and extensional counterparts along the notion of the extension function, whose intensional counterpart is the **instantiation operator** introduced by us in [Er06b]. Let t denote a type, then $\mathbb{I}_\kappa[t]$ designates the application of the instantiation operator onto the type in a given context κ. We introduce two shorthanded notations of the instantiation operator as follows. Firstly, we allow applying the operator on more than one type, i.e., a set of types $\mathcal{T}'$, defining

$$\forall\, \mathcal{T}' \subseteq \mathcal{T}_\zeta : \mathbb{I}_\kappa[\mathcal{T}'] := \bigcup_{t \in \mathcal{T}'} \mathbb{I}_\kappa[t].$$

Secondly we use an abbreviated form to denote a mapping from a single or a set of types $\mathcal{T}'$ to the set of instances that exist in different contexts.

$$\forall\, \mathcal{K}' \subseteq \mathcal{K}_\zeta : \mathbb{I}_{\mathcal{K}'}[\mathcal{T}'] := \bigcup_{\kappa \in \mathcal{K}'} \mathbb{I}_\kappa[\mathcal{T}'].$$

Based on the notion of the type, we define an **area-of-interest** in an EA.

Definition: Area-of-interest
An area-of-interest $\zeta \in \mathcal{A}$ is a tuple consisting of

- the set of admissible EA states $\mathcal{K}_\zeta$,
- the set of types $\mathcal{T}_\zeta$, and
- the set of covered instances $\mathcal{I}_\zeta := \mathbb{I}_{\mathcal{K}_\zeta}[\mathcal{T}_\zeta]$.

Above definition accounts for the intensional nature of the area-of-interest, instead of being observation-based, as the conceptualization is. A type t contained in an area-of-interest corresponds to a concept c in the conceptualization as:

$$\Box_\kappa \mathbb{I}_\kappa[t] \subseteq \mathbb{E}_\kappa[c].$$

This means, that from an intensional point of view a type can 'intend' to cover only a subset of a concept's individuals that exist in a given reality. From the intensional perspective the notion of the **specialization** has to be revisited. A type t_1 can intentionally specialize another type t_2, which means that t_1 is defined to instantiate to instances of t_2. Contrariwise, it can unintentionally be the case that $\mathbb{I}[t_1] \subseteq \mathbb{I}[t_2]$ holds, which we subsequently call that t_2 **subsumes** t_1, denoted as $t_1 <: t_1$. In order to support consistent reasoning on types, we introduce two additional types. The **bottom type** $\bot$ exists in any area-of-interest with $\mathbb{I}[\bot] = \emptyset$ and its complement the particular **top type** $\top$ for an area-of-interest with $\mathbb{I}[\top] = \mathcal{I}$.

In line with Buckl et al. [BKS10], we define a viewpoint as the intensional counterpart of a modeling language. Let in the following Υ designate the set of primitive graphical types, e.g. rectangles and lines, from the **visualization model** introduced in [Er06b].

Definition: Viewpoint
A viewpoint supplies conventions for the construction, interpretation, and use of a view reflecting information about a particular area-of-interest in the architecture. Formally, a viewpoint v is a tuple $v = \langle \mathcal{T}_v, \iota_v, n_v, \mathcal{K}_v \rangle$ with:

- the (abstract) syntax denoted by the set $\mathcal{M}_v$ of types,
- the semantics interpretation function for the types and their instances $\iota_v : \mathbb{I}_{\mathcal{K}_v}[\mathcal{T}_v] \cup \mathcal{T}_v \rightarrow \mathbb{I}_{\mathcal{K}_v}[\overline{\mathcal{T}}_v] \cup \overline{\mathcal{T}}_v$,
- the notation function[a] $n_v : \mathbb{I}_{\mathcal{K}_v}[\mathcal{T}_v] \rightarrow \mathbb{I}[\Upsilon]$, and
- the set of admissible EA states $\mathcal{K}_v$.

[a] For reasons of simplification, we assume a bijective function here. When the model elements are from the corresponding viewmodel which in turn can be embedded in an arbitrary richer information, this assumption does not pose a restriction.

In the definition we employed two sets of types relevant for the viewpoint, namely the types $\mathcal{T}_v$ constituting the viewpoint's **viewmodel** and the types $\overline{\mathcal{T}}_v$ of an underlying area-of-interest. The correspondence of the two sets allows to extend the **ontological commitment**-relationship between a conceptualization and a modeling language according to Buckl et al. [BKS10] towards an **addresses**-relationship between an area-of-interest and a viewpoint. Central thereto is the set of types covered by the viewpoint, i.e., the range of the **embedding** $\bar{\iota}_v : \mathcal{T}_v \rightarrow \overline{\mathcal{T}}_v$, which is a canonical sub-function of the semantics interpretation function. The addresses-relationship between a viewpoint v and a conceptualization ζ is defined as follows:

$$v \sim \zeta \Leftrightarrow \mathcal{K}_v = \mathcal{K}_\zeta \wedge dom(\bar{\iota}_v) = \mathcal{T}_\zeta.$$

The nature of the embedding $\bar{\iota}_v$ influences the quality of the viewpoint and determines, whether the viewpoint realizes a **notation function** or a **representation function** on the underlying area-of-interest. In addition the notion of **consistency** between different areas-of-interest pertains to the embedding. In order to avoid inconsistencies between different viewpoints the actual embeddings of the viewmodels into the underlying information model models must be **consistent embedding**-relationships in the sense of Section 5.2. In the following, we formalize the notion of the consistent embedding according to Buckl et al. [BKS10], further taking into account the distinction between **updating embedding** and **reading embedding**. Let

in the following $\mathcal{T}_1$ and $\mathcal{T}_2$ denote two sets of types, than a specific kind of embeddings can be described by an injective mapping $\tau : \mathcal{T}_1 \to \mathbb{P}(\mathcal{T}_2)$ between the types. This mapping reads as: 'given a type t_1, $\tau(t_1)$ is the type or set of types, to which all information supplied by instances of t_1 is distributed'. Explaining what 'distributed' means, we use the graphical notation of Dijkman in [Di06b]. We further assume that the types are contained in a corresponding area-of-interest, although also types in the viewmodel of a viewpoint could be mapped. Type t_2 as shown in Figure 6.2 covers all information contained in type t_1 and even more, such that $\tau(t_1) = t_2$ is an admissible mapping. Types t_4 and t_5 shown in Figure 6.3 together supply the information contained in type t_3, yielding an admissible mapping $\tau(t_3) = \{t_4, t_5\}$. Embeddings based on mappings of the latter type realize an **aggregation by composition**. For the exemplary types this reads a $t_3 = t_4 \uplus t_5$.

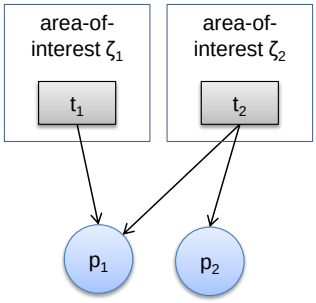


Figure 6.2: Embedding via generalization

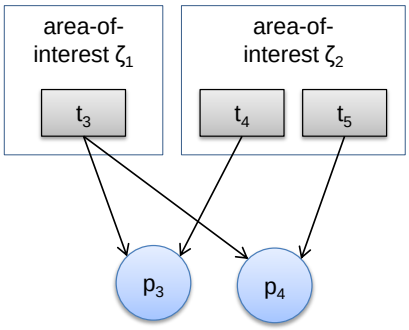


Figure 6.3: Embedding via aggregation by composition

A mapping $\tau : \mathcal{T}_1 \to \mathbb{P}(\mathcal{T}_2)$ of types gives to a consistent embedding, if the mapping of types is unambiguous, i.e.,

$$\forall\, t, t' \in \mathcal{T}_1 : t \neq t' \Leftrightarrow \tau(t) \cap \tau(t') = \emptyset$$

holds. In this case, we can establish a partial and bijective function between the instances of the a type $t \in \mathcal{T}_1$ and instances of the types $\tau(t)$. In this sense, the mapping defines an **updating embedding** denoted as <::. If an area-of-interest ζ_1 consistently embeds into an area-of-interest ζ_2, this means that ζ_2 covers at least all EA states covered by ζ_1 and if the mapping of types is unambiguous. Formal this reads as:

$$\begin{aligned} \zeta_2 \;<::\; \zeta_1 \Leftrightarrow\; & \mathcal{K}_{\zeta_1} \subseteq \mathcal{K}_{\zeta_2} \wedge \\ & \forall\, t, t' \in \mathcal{T}_{\zeta_1} : t \neq t' \Leftrightarrow \tau(t) \cap \tau(t') = \emptyset \wedge \\ & \forall\, t \in \mathcal{T}_{\zeta_1} : t = \biguplus_{t' \in \tau(t)} t'. \end{aligned}$$

In a similar sense, we can establish a consistent embedding-relationship between a viewpoint v and an area-of-interest ζ based on a mapping $\tau : \mathcal{T}_v \to \mathbb{P}(\mathcal{T}_\zeta)$. The viewpoint's relevant set of types is determined by the viewmodel, yielding the following formalization:

$$\begin{aligned} \zeta \;<::\; v \Leftrightarrow\; & \mathcal{K}_v \subseteq \mathcal{K}_\zeta \wedge \\ & \forall\, t, t' \in \mathcal{T}_v : t \neq t' \Leftrightarrow \tau(t) \cap \tau(t') = \emptyset \wedge \\ & \forall\, t \in \mathcal{T}_v : t = \biguplus_{t' \in \tau(t)} t'. \end{aligned}$$

For the question, whether two areas-of-interest can be integrated into a single multi-perspective approach to EA modeling, the consistent embedding-relationship provides a conceptual basis. Two areas-of-interest ζ_1 and ζ_2 can be **consistently integrated**, if there exists an area-of-interest ζ_3 in which both areas-of-interest can be embedded based on a single type mapping $\tau : \mathcal{T}_{\zeta_1} \cup \mathcal{T}_{\zeta_2} \to \mathbb{P}(\mathcal{T}_{\zeta_3})$, in short:

$$\exists\, \zeta_3 \in \mathcal{A} : \zeta_3 \;<::\; \zeta_1 \wedge \zeta_3 \;<::\; \zeta_2.$$

If in a viewpoint v the embedding $\bar{\iota}_v$ realizes an updating embedding based on a mapping $\tau : \mathcal{T}_v \to \mathbb{P}(\overline{\mathcal{T}}_v)$, the corresponding viewpoint realizes a partial bijective mapping from $\mathbb{I}[\mathcal{T}_v]$ to $\mathbb{I}[\overline{\mathcal{T}}_v]$. Together with the 'built-in' bijection $n_v : \mathbb{I}[\mathcal{T}_v] \to \mathbb{I}[\Upsilon]$ any mapping from the instantiated viewmodel to the graphical primitives is bijective, such that the viewpoint realizes a **notation function**.

A **representation function** does not require such strict mappings, i.e., the embedding $\bar{i}_v$ does not need to be bijective. Nevertheless, the embedding is not allowed to be arbitrary but has to satisfy certain constraints. We define the corresponding **reading embedding** (denoted as $\sqsubset::$) between a viewpoint v and an area-of-interest ζ via a type mapping:

$$\tau : \mathcal{T}_v \to \mathbb{P}(\mathbb{P}(\mathcal{T}_\zeta) \setminus \{\emptyset\}) \setminus \{\emptyset\}.$$

This type mapping supplies a *conjunctive* normal form for the mapping of types, i.e., specifies alternative mappings for each single type from $\mathcal{T}_v$. Two additional ways of mapping types, i.e., two ways of **aggregation**, are thereby described. On the one hand, a single type from v can be specialized to more than one type in ζ, with each of the specializing types providing a superset of the information that is made available in the specialized one. On the other hand, a single type from v can be decomposed into more than one type in ζ such that different decomposing types together provide more than the information that is made available in the decomposed one. In line with Dijkman [Di06b], we illustrate the different kinds of relationships between corresponding types from v and ζ, namely **aggregation by weak generalization** and **aggregation by partial composition**, in Figures 6.4 and 6.5, respectively.

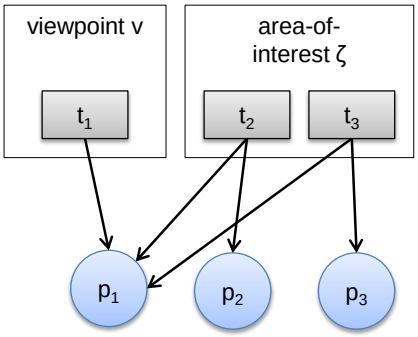


Figure 6.4: Embedding via aggregation by weak generalization

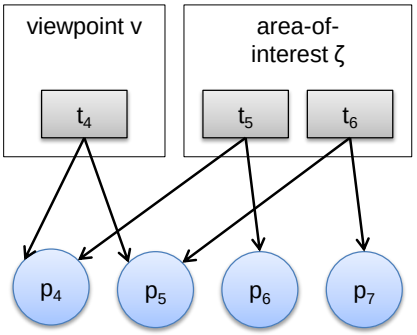


Figure 6.5: Embedding via aggregation by partial composition

The different kinds of aggregations of types are subsequently denoted by two operators on types. We write:

- $t_1 = t_2 \sqcup t_3$ to denote aggregation by weak generalization, and
- $t_4 = t_5 \sqcap t_6$ to denote aggregation by partial composition.

Both kinds of aggregation can be applied together in a single mapping, reading for example as $t_2 \sqcup (t_3 \sqcap t_4)$. This means that either an instance of t_2 or a partial composition of instances of t_3 and t_4 maps to an instance of t_1. Any such mapping τ gives rise to a **reading embedding** between a viewpoint v and an area-of-interest ζ as follows:

$$\begin{aligned} \zeta \ \sqsubset\!: \ v \Leftrightarrow \ & \mathcal{K}_v \subseteq \mathcal{K}_\zeta \ \wedge \\ & \forall\, t, t' \in \mathcal{T}_v : t \neq t' \Leftrightarrow \tau(t) \cap \tau(t') = \emptyset \ \wedge \\ & \forall\, t_1 \in \mathcal{T}_v : t_1 = \bigsqcup_{\mathcal{T}' \in \tau(t_1)} \bigsqcap_{t_2 \in \mathcal{T}'} t_2. \end{aligned}$$

A similar relationship can be established between two areas-of-interest ζ_1 and ζ_2 as follows:

$$\begin{aligned} \zeta_1 \ \sqsubset\!: \ \zeta_2 \Leftrightarrow \ & \mathcal{K}_{\zeta_1} \subseteq \mathcal{K}_{\zeta_2} \ \wedge \\ & \forall\, t, t' \in \mathcal{T}_{\zeta_1} : t \neq t' \Leftrightarrow \tau(t) \cap \tau(t') = \emptyset \ \wedge \\ & \forall\, t_1 \in \mathcal{T}_{\zeta_1} : t_1 = \bigsqcup_{\mathcal{T}' \in \tau(t_1)} \bigsqcap_{t_2 \in \mathcal{T}'} t_2. \end{aligned}$$

A viewpoint v, whose embedding $\bar{\iota}_v$ is realized based on a mapping $\tau : \mathcal{T}_v \to \mathbb{P}(\mathbb{P}(\mathcal{T}_\zeta) \backslash \{\emptyset\}) \backslash \{\emptyset\}$ supplies an injective function from $\mathbb{I}[\mathcal{T}_v]$ to $\mathbb{I}[\overline{\mathcal{T}}_v]$. Together with the 'built-in' bijection

$$n_v : \mathbb{I}[\mathcal{T}_v] \to \mathbb{I}[\Upsilon]$$

any mapping from the instantiated viewmodel to the graphical primitives is only injective, such that the viewpoint realizes a **representation function**.

Aggregation by weak generalization gives rise to an alternative definition of the subsumption-relationship between different types. The aggregating type can read from the aggregated ones, but instances cannot replace the the aggregated types regarding update operations. We extend the **reading embedding** also to the level of individual types, such that we can denote the reading relationships as described in Figure 6.4 via **weak subsumption** $t_2 \sqsubset\!: t_1 \wedge t_3 \sqsubset\!: t_1$. For the aggregation by partial composition, a more complex correspondence with the weak subsumption-relationship can be established via two intermediary types t_5' and t_6':

$$\exists t_5', t_6' : ((t_5' \neq t_5) \vee (t_6' \neq t_6)) \wedge t_5 \sqsubset\!: t_5' \wedge t_6 \sqsubset\!: t_6' \wedge t_4 = t_5' \uplus t_6'.$$

Two areas-of-interest ζ_1 and ζ_2 that can both be reading embedded into a third area-of-interest ζ_3 based on a common type mapping $\tau : \mathcal{T}_{\zeta_1} \cup \mathcal{T}_{\zeta_2} \to \mathbb{P}(\mathbb{P}(\mathcal{T}_{\zeta_3}) \setminus \{\emptyset\}) \setminus \{\emptyset\}$ can be **weakly integrated**. If such integration is possible, information modeled in accordance to ζ_1 or ζ_2 can be migrated to information modeled according to ζ_3. The migration can nevertheless change the nature of viewpoints building on the original areas-of-interest. The missing bijection between the instances of the original types and the types in the target area-of-interest ζ_3 can invalidate the bijectivity of a viewpoint's underlying embedding $\bar{\iota}$ such that a notation function 'downgrades' to a representation function.

6.2 Information model meta-language (IM2L)

Chapter 5 already delineated several characteristics of the EA domain, which pertain to the meta-language used to specify the information models and the information model building blocks. In the following, we revisit these characteristics to derive requirements for a meta-language for EA information modeling and discuss to which extent current multi-purpose meta-modeling facilities are able to satisfy these requirements, as also discussed in [BMS10b, BMS10g, BMS10i].

The paradigm of object-oriented EA information modeling requires a set of **(C0) syntactic primitives** that must be supported by the *information model meta-language* (IM2L). Central thereto are **(C0a)** TYPES, **(C0b)** PROPERTIES, and **(C0c)** n-ary RELATIONSHIPS. For both properties and relationship ends, cardinality constraints are necessary, i.e., it must be possible to specify a **(C0d)** LOWER BOUND and an UPPER BOUND. Further, the IM2L must support **(C0e) nominating properties** that supply a name for an identifiable part of the enterprise. **(C0f)** Specialization mechanisms support modeling EA information on different levels of detail, whereas **(C0g)** RELATIONSHIP TYPES are specify properties of relationships.

In line with Section 4.4 we require additional IM2L concepts for:

(C1) Domain specific modeling: Properties of type **(C1a)** ENUMERATION are frequently used to designate different states of an architectural element or to classify such elements in different groups. Particular domain specific primitive **(C1b)** DATA-TYPES, as money, date, or time, must be supported by the IM2L. Furthermore, **(C1c)** HIERARCHIES are a prominent concept in EA modeling used to reflect tree-like structures, as organizational structures, business process hierarchies, or component-based application architectures.

(C2) Temporal modeling: EAs evolve over time, resulting in changes that pertain to both the instance level and to the type level. The IM2L must therefore supply mechanisms for historizing EA states and creating EA plans, i.e., must support **(C2a) bi-temporal modeling**, that distinguished between **valid time** and **transaction time**. Further, the conceptualization and classification of architectural elements can change over time such that the distinction between **(C2b) rigid** and **(C2c) non-rigid**, in particular **semi-rigid** and **anti-rigid**, types must be supplied.

(C3) Cross-cutting aspects: Similar cross-cutting aspects, as **projects** or **goals** can be applied onto different architecture concepts. The IM2L must supply mechanisms to add dispersive MIXIN TYPES representing cross-cutting aspects to types. Further, it must be possible to package different concepts that together reify a cross-cutting aspect into a re-usable extension.

(C4) Specialization by constraint: EA modeling applies to different **areas-of-interest** that can contain the same types distinguished only on property-level. Different stakeholders of the EA management can have different information demands in respect to the types, but can also have different understandings of relevant parts. The IM2L must supply a mechanism to constrain the relevant part of the EA by specializing types to sub-types that carry a particular property.

(C5) Glossary development: EA modeling is performed in the context of corresponding linguistic communities. The IM2L must supply mechanisms to consistently adapt the predication for the different EA concepts without changing the underlying syntactical structure.

(C6) Multi-perspectivity: The **areas-of-interest** of different EA stakeholders do not exist in isolation but relate to each other in manifold ways, e.g. representing different levels of granularity of overlapping regarding selected concepts. The IM2L must support the consistent integration of different areas-of-interest, represented in EA information models, into one comprehensive description of the relevant EA concepts. Thereby, applicability of the particular stakeholder-specific viewpoints has to be ensured.

In the following, we analyze general purpose meta-languages as the *Unified Modeling Language* (UML) infrastructure specification [Ob10c], the *Object Role Model* (ORM) [Ha09], or the *MEMO Meta Modeling Language* (MML) [Fr09] against above characteristics. This analysis builds on our findings from [BMS10g] and shows in Table 6.1 that especially 'sophisticated' characteristics as temporal modeling and dispersive types are not well supported in the three meta-languages.

		UML	ORM	MML
C0	**C0a** (TYPE)	●	●	●
	C0b (PROPERTY)	●	◐[1]	●
	C0c (RELATIONSHIP)	●	●	◐[2]
	C0d (CARDINALITY CONSTRAINT)	●	●	●
	C0e (NOMINATING PROPERTY)	○	○	○
	C0f (SPECIALIZATION)	●	●	●
	C0g (RELATIONSHIP TYPE)	●	●	○
C1	**C1a** (ENUMERATION)	●	●	○
	C1b (domain-specific DATA-TYPE)	○	○	◐[3]
	C1c (ENUMERATION)	●	●	○
C2	**C2a** (**bi-temporal modeling**)	○	○	○
	C2b (**rigid type**)	●	●	●
	C2c (**non-rigid type**)	○	◐[4]	○
C3 (**dispersive type**)		○	○	○
C4 (**specialization by constraint**)		◐[5]	○	○
C5 (external **predicator**)		○	○	○

[1] ORM does not support typed properties with primitive types.
[2] MML supports only binary relationships.
[3] MML supports modeling of TIME.
[4] ORM supports *unary fact types* which can be used to express non-rigid types.
[5] The *Query/View/Transformation* (QVT) specification [Ob05] supplies *templates* that can be used to designate restrictions with respect to value assignments.

Table 6.1: Support for meta-language characteristics

C6 Multi-perspectivity does not directly demand additional IM2L concepts, but requires that the concepts support formal underpinnings for **embedding** and **aggregation** as discussed in Section 6.1. Due to the prevalence of UML and UML-like meta-languages for describing EA information models (cf. discussions in [BMS10g]), we decided to use the UML infrastructure, or more precisely the *Meta Object Facility* (MOF) [Ob06a] as basis for the design of the IM2L. This decision is further promoted by the work of Guizzardi [Gu05], who conducted an extensive analysis of the ontological underpinnings of object-oriented modeling facilities. The different aspects of the IM2L are expatiated subsequently, starting with an exposition of the language constituents, i.e., of the syntax and semantics in Section 6.2.1. The

considerations on language semantics are furthered in Section 6.2.2, discussing the intricacies of **subsume**-relationships on the model elements of an information model. Section 6.2.3 discusses how different information models or building blocks thereof can be integrated and can be adapted based on operators that are consistent with the embedding-relationships introduced before. Final section 6.2.4 describes how the syntactic primitives of the IM2L are notated, i.e., represented graphically.

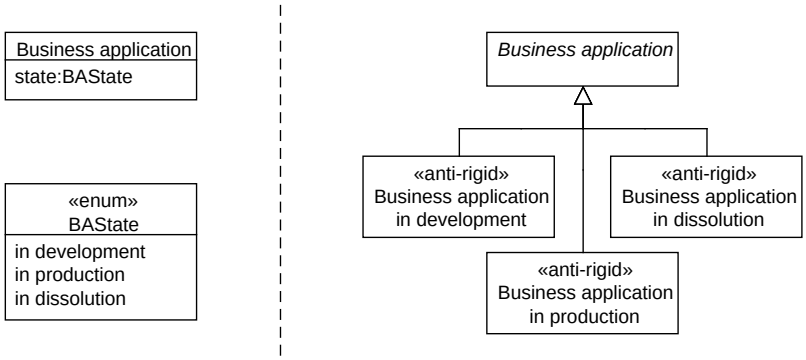
6.2.1 Syntax of the IM2L

Central to the IM2L is the concept of the type, which is further detailed via distinction of *universals* proposed by Guizzardi in [Gu05]. He introduces two different kinds of universals:

- *substantial universals*, whose instances represent substance, i.e., isolated phenomena from the universe of discourse, and
- *relator universals*, whose instances relate instances of other universals, i.e., are dependent on the corresponding phenomena.

From an ontological point of view, this distinction is fully sufficient to cover all kinds of phenomena in the universe of discourse. We further the discussions and account for a differentiation between types that supply or carry at least one PROPERTY or are related to other types, and ones, whose instances carry in themselves the sufficient information. Types of the latter kind are reflected in object-oriented modeling languages as *enumerations*. In EA modeling properties of an enumeration type are used to represent different *states* or *phases* that the instances of some other type can be in. In the mindset of Guizzardi [Gu05], enumeration-typed properties should not exist, but corresponding semi-rigid or anti-rigid universals should specialize the universal under consideration. Example 6.2 delineates this stringency.

Example 6.2: Modeling life-cycle states. The information model should reflect that a BUSINESS APPLICATION can be in different life-cycle states during its lifetime, ranging from IN DEVELOPMENT to IN DISSOLUTION. Two modeling alternatives, one using an enumeration-typed property and one anti-rigid types, can be applied as follows.



Both modeling alternatives are equivalent in respect to the provided information. Therefore, and due to the frequent use of enumerations in EA descriptions, we decide to introduce the concept ENUMERATION as type to the IM2L. Nevertheless, in respect to evolving EA information models, the conceptual and informational equivalence of both modeling alternatives has to be incorporated in the **subsume**-relationships. The meta-properties of **rigidity** and **identity** are of interest for UNIVERSAL TYPES, i.e., types that carry at least one PROPERTY. In line with Guizzardi [Gu05] we detail the meta-property of identity: types that carry an IC can be further distinguished into types that supply the IC and ones that inherit the IC. We can more precisely formulate that a type supplies an IC, if it carries an IC, which is not carried by all types that it specializes. An additional prerequisite for supplying an IC is according to Guarino and Welti [GW00b] being rigid. This means that any non-rigid type can carry an IC which has to be inherited from another type, ultimately a rigid one supplying the IC. Figure 6.6 shows the UML inheritance hierarchy, in which the different kinds of types participate. The diagram further introduces abbreviations for the meta-properties: $+\mathbf{I}$ and $\neg\mathbf{I}$ for carrying an IC or not; $+\mathbf{R}$ and $\neg\mathbf{R}$ for rigid and non-rigid with $\sim\mathbf{R}$ and $-\mathbf{R}$ for anti-rigid and semi-rigid, respectively.

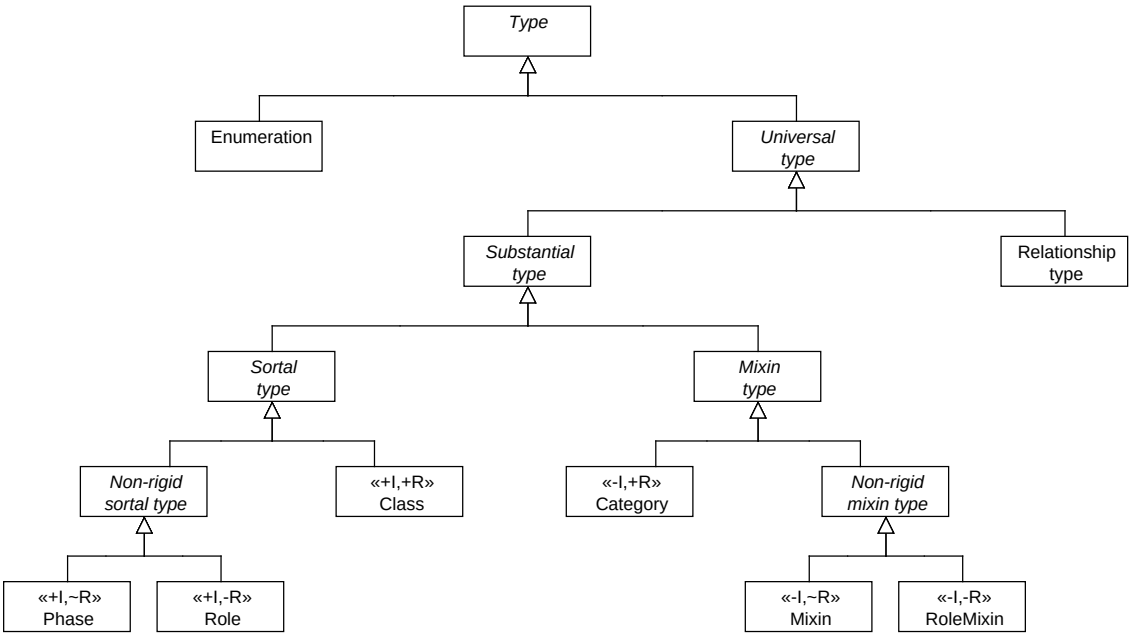


Figure 6.6: Kinds of types reflecting their meta-properties

From the perspective of the *denotational semantics* temporal information in EA modeling has to be discussed. Let in the following $\mathcal{T}$ denote the set of points in time pertaining to both the future $\mathcal{T}^+$ and the past $\mathcal{T}^-$. For any instance i of a universal type, two particular **transaction times** $\tau_c(i) \in \mathcal{T}^-$ and $\tau_d(i) \in \mathcal{T}^- \cup \{\infty\}$ are of interest—the time of its creation and of its deletion. The transaction time under consideration, e.g. the time of visualizing the EA, is part of the context together with an information on the relevant **valid time**: $\mathcal{K} = \mathcal{T}^- \times \mathcal{T}$.

We distinguish between two kinds of universal types, firstly ones not explicitly accounting for a valid time and **time-dependent** ones. Latter supply two valid times $\overleftarrow{\tau}, \overrightarrow{\tau}$ designating the start-point and the end-point for the period of validity. This period of validity can change with each transaction such that both valid times are functions $\overleftarrow{\tau}, \overrightarrow{\tau} : [\tau_c, \tau_d] \to \mathcal{T}$ of the current time τ such that

$$\forall i \in \mathcal{I}, \tau \in [\tau_c(i), \tau_d(i)] : \overleftarrow{\tau}(\tau) \leq \overrightarrow{\tau}(\tau).$$

The model element of the PROPERTY plays an important role for describing EA information models. Any **universal type** can be equipped with properties further specifying characteristics of the instances of the type. As any type itself is a reification of a specific property, the distinction between a property and a type is not a sharp one. In contrast, it depends on the level of abstraction taken in describing the EA. This is exemplified by the different ways to reflect life-cycle information, which further rises the idea that the level of abstraction taken can depend on the intended usage of the information model for representing the architecture or parts thereof. A property in the information model does not only supply a name, but also a data-type, i.e., specifies the admissible values, that an instance of the universal type can assign to the property. In the context of EA modeling, several data-types are of interest, namely **enumerations** and **primitive data-types**. Data-types of the latter group are listed in the following, and are defined in terms of the XML schema specification[1]:

- The boolean data-type represents the values of two-valued logic: *true* and *false*.
- The date data-type represents exactly one day on the timeline, starting on the beginning moment of each day, up to but not including the beginning moment of the next day.
- The dateTime data-type represents instants of time on the timeline.
- The decimal data-type represents numeric values that can be expressed as a fraction of an integer by a non-negative power of ten, i.e., expressible as $i/10^n$, where i and n are integers and $n \geq 0$.
- The integer data-type represents numeric values being either drawn from the natural numbers or their negatives.
- The money data-type represents numeric values that can be expressed as a decimal value together with a currency code, designating the currency in which the corresponding amount of money is counted.
- The string data-type represents character strings, with a character being the atomic unit of text.

Any primitive data-type d supplies an equivalence relationships $= \subseteq \mathbb{I}[d] \times \mathbb{I}[d]$, which holds for identical values from the type's domain.

Enumerations specify sets of admissible values, the so-called ENUMERATION LITERALS, represented as text elements. In addition to specifying a data-type, a property also supplies two cardinality constraints, called **lower bound** and **upper bound**, respectively. The cardinality

[1] For additional information on data-types in XML schemata see `http://www.w3.org/TR/xmlschema11-2`, last accessed 04-05-2011.

constraints specify the minimum and maximum number of expected values assigned to the property at a particular instance. For any property, the lower bound can take values 0 and 1, indicating that the property is **optional** or **mandatory**. The upper bound can take values 1 and ∞ indicating a **unique** and a **multi-valued** property, respectively. For any tuple $\langle l, u \rangle$, we define the **multi-valued instantiation operator** $\mathbb{I}^{l,u}$ on a type d as follows:

$$\mathbb{I}^{l,u}[t] = \begin{cases} \bigcup_{i=l}^{u} \prod_i \mathbb{I}[t] & \text{if } l > 0 \\ \{\emptyset\} \cup \mathbb{I}^{1,u}[t] & \text{if } l = 0 \end{cases}.$$

In this sense, we interpret a property p supplied by a universal type u as a function, depending on the context κ:

$$p : \mathbb{I}_\kappa[u] \times \mathcal{K} \to \mathbb{I}^{p.\text{lower},p.\text{upper}}[p.\text{type}].$$

Any universal type t that carries an IC also carries at least one property, whose values act as nominators for th type's instances. The values of the **nominating properties** $p_1 \dots p_n$ provide the basis for the IC as introduced in Section 4.3.2:

$$\forall i_1, i_2 \in \mathbb{I}[t] : \Lambda_t(i_1, i_2) \Leftrightarrow \bigwedge_{j=1}^{n} p_j(i_1) = p_j(i_2).$$

In above expression, we surpressed the time-dependency of properties. This is possible, as we demand that the value of the nominating property does not change over time. The nominating property therefore has a special denotational semantics in respect to the temporal context of the model. Given the transaction time $\tau_c(i) \in \mathcal{T}^-$, when the particular instance $i \in \mathcal{I}$ was created, and the time $\tau_d(i) \in \mathcal{T}^- \cup \{\infty\}$, when the instance was deleted, the following condition holds for a nominating property p:

$$\forall \tau_t, \tau_t' \in [\tau_c(i), \tau_d(i)]; \tau_v, \tau_v' \in \mathcal{T} : p(i, \langle \tau_t, \tau_v \rangle) = p(i, \langle \tau_t', \tau_v' \rangle).$$

The values of all other properties can change over time. We further distinguish properties, whose values depend on the transaction time only, and ones dependent on both transaction time and valid time. Latter properties are not subject to particular constraints, while for not time-dependent ones the following constraint holds:

$$\forall i \in \mathcal{I}; \tau_t \in [\tau_c(i), \tau_d(i)]; \tau_v, \tau_v' \in [\overleftarrow{\tau}(\tau_t), \overrightarrow{\tau}(\tau_t)] : p(i, \langle \tau_t, \tau_v \rangle) = p(i, \langle \tau_t, \tau_v' \rangle).$$

For the model element of the RELATIONSHIP, we distinguish in line with Guizzardi [Gu05]:

- **Material relationships** are relationships carried and reified in a corresponding instance of a specific universal type.
- **Derived relationships**[2] are relationships that are derived based on certain other relationships of the involved substantial types.

Example 6.3 illsustrates the distinction between the two types of relationships.

[2] Guizzardi calls this kind of relationships *formal relationships*.

Example 6.3: Types of relationships. The fact that a BUSINESS APPLICATION is hosted at a specific ORGANIZATIONAL UNIT is dependent on a DEPLOYMENT, i.e., a material quality in the universe of discourse. The fact that one BUSINESS APPLICATION has less users than another BUSINESS APPLICATION is not a material quality, but a consequence of both applications having assigned certain numbers of users.

In Example 6.3 the DEPLOYMENT is a RELATIONSHIP TYPE, relating business applications and organizational units. Any kind of relationship, either material or derived, is specified by its RELATIONSHIP ENDS. A relationship end originates at the substantial type participating in the relationship and further supplies cardinality constraints in terms of a **lower bound** (from $\{0, 1\}$) and **upper bound** (from $\{1, \infty\}$). These constrain the minimum and maximum number of relationship instances in which a single instance of the corresponding substantial type can participate. Relationship ends with lower bound 0 are called **optional**, whereas ones with lower bound 1 are called **mandatory**. In respect to the upper bound we distinguish between **unique** ends (upper $= 1$) and **multi-valued** ends (upper $= \infty$). Any particular relationship has at least two relationship ends. For most of our subsequent considerations both material and derived relationships are treated in a uniform way. In line with the argumentation of Halpin [Ha09] the uniform treatment is abandoned for special cases of n-ary relationships. For any $n > 2$ any n-ary relationships with unique relationship ends can be decomposed into m-ary relationships, when $m - 1 > 1$ is the number of incoming unique relationship ends. While for reasons of model utilization, we do not impose a restriction on derived relationships, we demand that ones reified in a relationship type cannot be decomposed. The denotational semantics of a relationship is introduced in the following based on a partitioning of the relationship's ends $\mathcal{E}$ according to their cardindality constraints into:

- mandatory and unique relationship ends $\mathcal{E}^{1,1}$,
- optional and unique relationship ends $\mathcal{E}^{0,1}$,
- mandatory and multi-valued relationship ends $\mathcal{E}^{1,\infty}$, and
- optional and multi-valued relationship ends $\mathcal{E}^{0,\infty}$.

A relationship r is interpreted as a relation in a mathematical sense. Therefore, the admissible instances of r are confined to the following space of values:

$$\mathbb{I}_\kappa[r] \subseteq \prod_{e \in \mathcal{E}^{1,\infty}} \mathbb{I}_\kappa^{1,\infty}[e.\text{type}] \times \prod_{e \in \mathcal{E}^{0,\infty}} \mathbb{I}_\kappa^{0,\infty}[e.\text{type}] \to \prod_{e \in \mathcal{E}^{1,1}} \mathbb{I}_\kappa^{1,1}[e.\text{type}] \times \prod_{e \in \mathcal{E}^{0,1}} \mathbb{I}_\kappa^{0,1}[e.\text{type}],$$

being total in respect to $\mathcal{E}^{1,\infty}$. In above definition we assume that for any non-empty set $\mathcal{I}$ the following holds $\mathcal{I} \times \emptyset = \mathcal{I}$.

The denotational semantics of a relationship has to be detailed regarding the temporal context. Any relationship r has two transaction times $\tau_c(r) \in \mathcal{T}^-$ and $\tau_d(r) \in \mathcal{T}^- \cup \{\infty\}$ assigned, representing the time of its creation and deletion. These transaction times relate to the transaction times of the corresponding instances $\mathcal{I}$ of the related substantial types:

$$max_{i \in \mathcal{I}} \tau_c(i) \leq \tau_c(r) \leq \tau_d(r) \leq min_{i \in \mathcal{I}} \tau_d(i).$$

If the relationship further is **time-dependent**, the valid times of the relationship have to satisfy the following constraint:

$$\forall \tau_t \in [\tau_c(r), \tau_d(r)] : max_{i \in \mathcal{I}} \overleftarrow{\tau}_i \leq \overleftarrow{\tau}_r \leq \overrightarrow{\tau}_r \leq min_{i \in \mathcal{I}} \overrightarrow{\tau}_i.$$

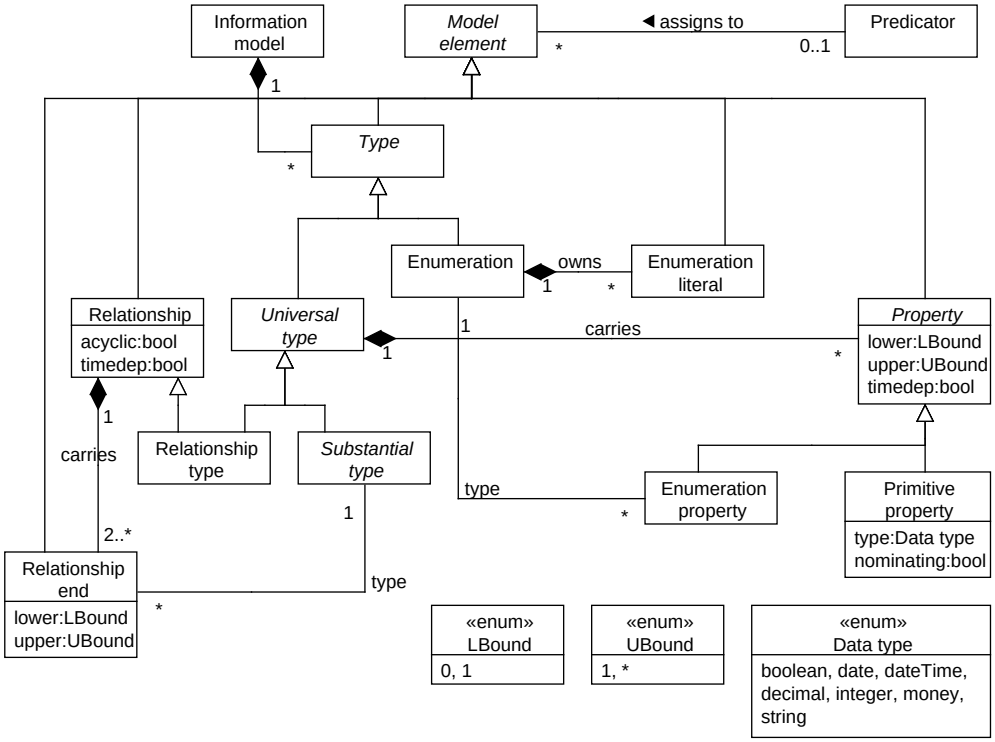


Figure 6.7: Basic model elements of the IM2L

Figure 6.7 displays the MODEL ELEMENTS of the IM2L and embeds them into the context of the information model as well as the semantics assignments by predicators. The MODEL ELEMENTS according to the IM2L have the following meaning:

- A MODEL ELEMENT is a concept directly or transitively owned by the information model. A model element represents an architecture property, that can be predicated, or a corresponding (named) value assignment.
- A TYPE describes a common characteristics of different instances in the architecture. According to the meta-properties that apply to this characteristics, different kinds of types can be distinguished.
- A UNIVERSAL TYPE is a type, whose instances can carry additional properties designating and specifying them. The instances can further be unknown at the time of EA information modeling.
- An ENUMERATION is a type, whose instances (the ENUMERATION LITERALS) do not carry additional properties and are known at the time of EA information modeling.

- An ENUMERATION LITERAL represents a named instance of an enumeration.
- A SUBSTANTIAL TYPE is a universal type, whose instances represent substance, i.e., architectural phenomena that exist in themselves.
- A PROPERTY describes a characteristic of the instances of the owning universal type. An property is explicitly bound to a set of admissible values by typing. In this respect, the ENUMERATION PROPERTY and the PRIMITIVE PROPERTY can be distinguished.
- A RELATIONSHIP END specifies that a substantial type participates in a relationship with other substantial types. The relationship end thereby specifies the exact quality of the participation and acts as predicable designator for the different participating roles in a relationship.
- A RELATIONSHIP is the hub for different RELATIONSHIP ENDS. It designates the interrelating quality and can be 'made material' in a RELATIONSHIP TYPE.
- A RELATIONSHIP TYPE is a universal type, whose instances represent relationships between other universal types.

Substantial types can relate to each other via a **specialization** relationship. Specialization designates that the specializing type shares the ICS, properties, and relationships of the specialized type. Against the background of the distinction between different kinds of substantial types, certain restrictions in respect to the specialization apply, of which some are already discussed by Guarino and Welti [GW00a]:

- No MIXIN TYPE can specialize a SORTAL TYPE, as it would thereto inherit the IC of this type.
- Every SORTAL TYPE can specialize at most one other SORTAL TYPE, as it otherwise would inherit different and potentially conflicting ICs.
- No **rigid** type can specialize a **non-rigid** type, as a rigid type would fail to act as a non-rigid one with respect to changing contexts.
- No SORTAL TYPE can specialize a RELATIONSHIP TYPE and vice-versa, as being independent substance and dependent relation are inconsumerable.
- Every **non-rigid** SORTAL TYPE must specialize another SORTAL TYPE, as the non-rigid type carries an IC, but cannot supply one.
- The specialization relationships between the universal types must form an acyclic graph.

If not otherwise stated by one of above restriction, a substantial type can specialize arbitrarily many substantial types. Applying this rules to the general set of substantial types introduced above, we can derive the supported specialization relationships between the universal types in the IM2L as shown in Figure 6.8.

We revisit the concept **specialization by constraint** as discussed by Date and Darwen in [DD00, pages 278–281]. In line with our understanding of subsumption being that the instantiation of a subsumed type is a subset of the instantiation of its subsuming counterpart, specialization by constraint derives a subsumed type by constraining its general counterpart. Such constraints can apply on the properties and relationship ends of the type. Let t be a

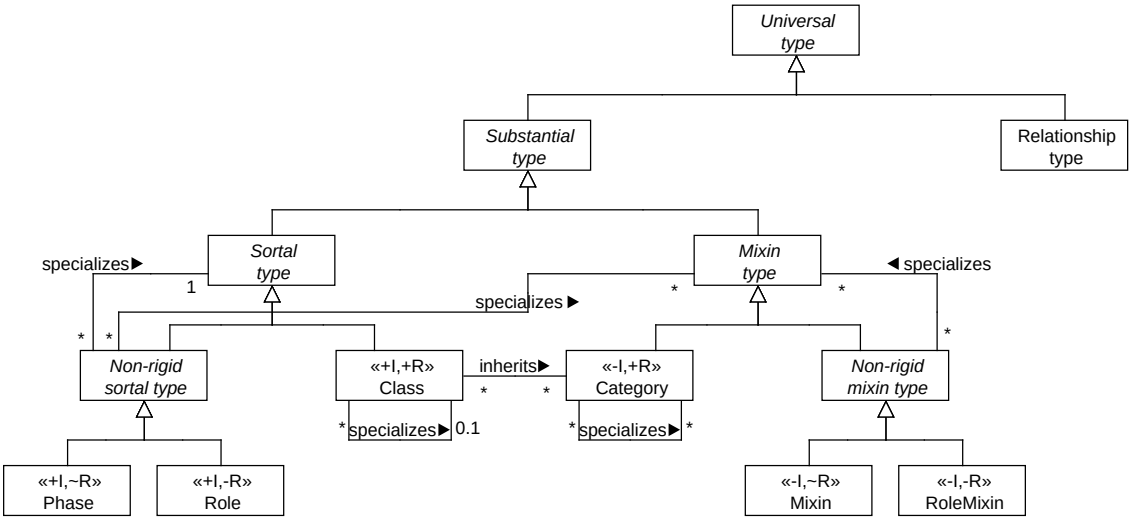


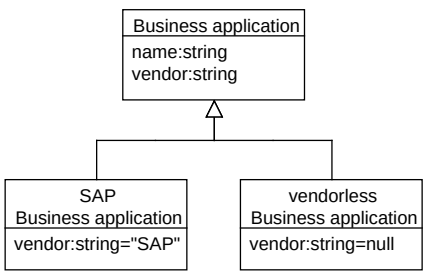
Figure 6.8: Admissible inheritance relationships between the different types of universal types

universal type carrying a property p of type d. Via specialization by constraint we can define a type t' based on a restriction of the admissible values for the property to the set $\mathcal{I}'_d \subset \mathbb{I}[d]$ or $\mathcal{I}'_d = \emptyset$ as follows

$$\mathbb{I}_\kappa[t'] = \{i \in \mathbb{I}_\kappa[t] \mid i.p \in \mathcal{I}'_d\}.$$

We notate above specialization as $t' = t[p/\mathcal{I}'_a]$. In Example 6.4 we revisit the enterprise modeling business applications and organizational units once again.

Example 6.4: Specialization by constraint. A group of stakeholders in the enterprise is especially interested in BUSINESS APPLICATIONS of vendor SAP. Thereto, the group specializes by constraint the class business application to the relevant SAP business applications. A different group of stakeholders is—for reasons of data quality—interested in the applications that have no vendor specified. The EA information model supplying the two specializations by constraint reads as follows:



Specialization by constraint can also be used to restrict a type to its instances that relate to a specific relationship. Let e_1 be a relationship end carried by a relationship r and targeting a substantial type t, we can specialize t by constraint to $t' = t[e_1 \downarrow r']$. Therein, r must subsume r', which in turn can be a derived via specialization by constraint, e.g. by constraining the owning type of an associated relationship end. Let r be a binary relationship with ends e_1 and e_2 of types t_1 and t_2, respectively, we can specialize r by constraint to r' based on a type t'_1 subsumed by t_1 with

$$\mathbb{I}_\kappa[r'] = \{i \in \mathbb{I}_\kappa[r] \mid i.e_1 \in \mathbb{I}_\kappa[t'_1]\}.$$

In terms of a corresponding operation, we write $r' = r[e_1 \downarrow t'_1]$. The formal mechanisms described above are represented by dedicated language concepts that adopt the *template* mechanism from the QVT specification [Ob05]. Figure 6.9 details the syntax of the IM2L in respect to constrained substantial types:

- A CONSTRAINED SUBSTANTIAL TYPE is itself a substantial type and relates to the base type from which it specializes. Further it supplies at least one VALUE CONSTRAINT.
- A RELATIONSHIP END CONSTRAINT is a specific value constraint specifying that one relationship end is confined to a particular RELATIONSHIP.
- A ENUMERATION PROPERTY CONSTRAINT is a specific value constraint demanding that one enumeration property is assigned only the selected admissible ENUMERATION LITERALS.
- A PRIMITIVE PROPERTY CONSTRAINT is a specific value constraint specifying that one primitive property takes a selected value, which in turn is represented textually.

Different rules have to hold, when VALUE CONSTRAINTS are applied. Firstly, the targeted RELATIONSHIP END or PROPERTY must be carried by the substantial type specified as BASE TYPE or by one of its generalizing types. Secondly, the value or type specification must match the type specification of the ENUMERATION PROPERTY or RELATIONSHIP END. This means that the selected RELATIONSHIP specified by a RELATIONSHIP END CONSTRAINT must be subsumed the end's original relationship. As the bottom type $\perp$ is subsumed any type, we can constrain a relationship end to this type, meaning that only not related instances are targeted. For any ENUMERATION PROPERTY CONSTRAINT this means that the selected ENUMERATION LITERALS must be owned by the property's corresponding enumeration or be the empty set $\emptyset$. In the following we describe these constraints in formal terms, using an auxiliary function selfOrSuper(). Given a universal type t the following holds:

$$\mathsf{selfOrSuper(t)} = \{t' \mid t <: t'\}.$$

The constraints are:

```
context: RelationshipEndConstraint
inv: selfOrSuper(targets.targets)->includes(carries.baseType)
inv: type <: targets.carries or type = null

context: EnumerationPropertyConstraint
inv: selfOrSuper(targets.carries)->includes(carries.baseType)
inv: targets.type.owns->includesAll(values)
```

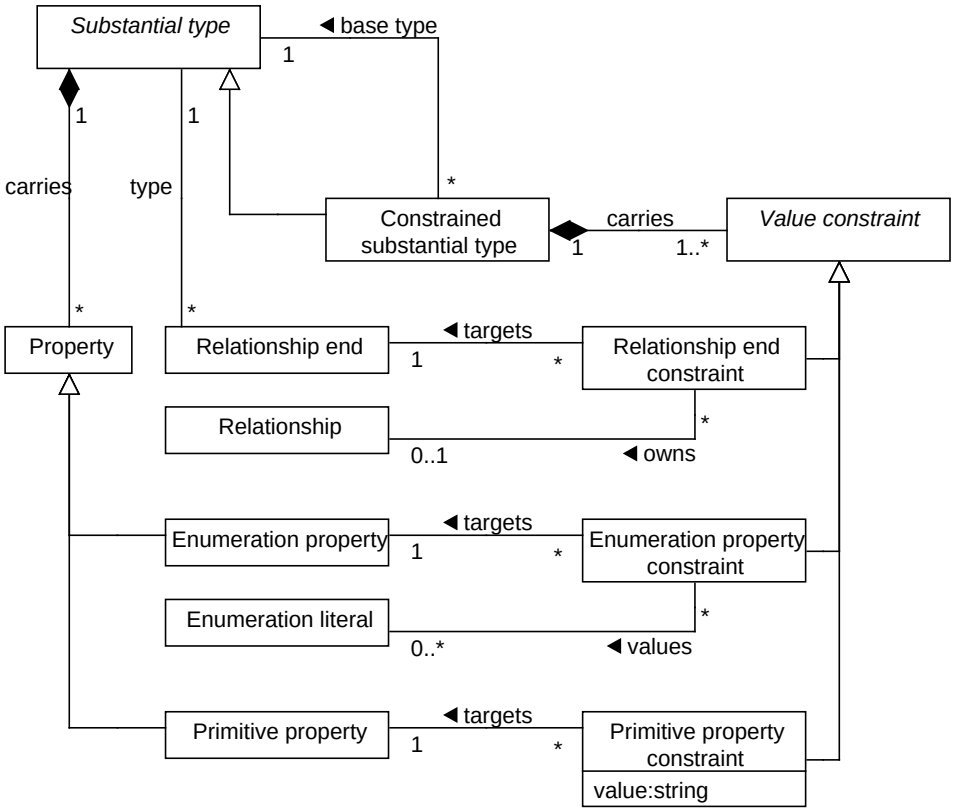


Figure 6.9: IM2L concepts for specifying a CONSTRAINED SUBSTANTIAL TYPE

The CONSTRAINED RELATIONSHIP is supplied with VALUE INVERSE CONSTRAINTS that restrict the admissible types for a RELATIONSHIP END. Figure 6.10 details the corresponding syntax of the IM2L.

For a RELATIONSHIP END INVERSE CONSTRAINT two particular constraints have to hold. Firstly, the targeted RELATIONSHIP END must be carried by the corresponding BASE TYPE. Secondly, the selected SUBSTANTIAL TYPE must be subsumes by the relationship end's type, as only thereby a restriction of the admissible values is achieved. Again the bottom type $\perp$ can be used to constrain to relationships with unbound ends. In formal terms, these constraints read as:

```
context: RelationshipEndInverseConstraint
inv: selfOrSuper(targets.carries)->includes(carries.baseType)
inv: type <: targets.targets or type = null
```

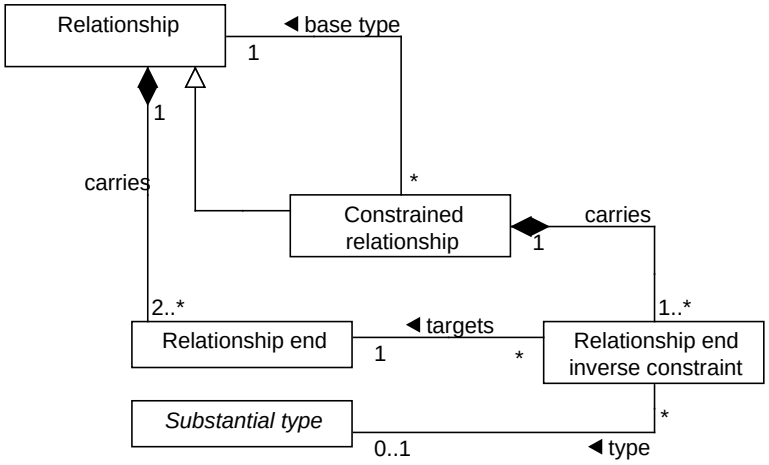


Figure 6.10: IM2L concepts for specifying a CONSTRAINED RELATIONSHIP

6.2.2 Consistent embedding-relationships

In this section we present a formalization of the **consistent embedding**-relationships between information models and delineate how the relationships between information models can be derived from subsumption relationships between the constituting model elements. Therefore, we build on the distinction between two types of *subsumption* as discussed by Pierce [Pi02, pages 181–185]:

- *Subsumption in respect to reading access*: for this relationship, we assume that instances of the model elements to relate are accessed read-only. In the following this kind of subsuming is alluded to as **weak subsumption**. For two MODEL ELEMENTS m_1 and m_2 the fact that m_2 is weakly subsumed m_1 is denoted as $m_2 \sqsubset: m_1$.
- *Subsumption in respect to reading and writing access*: for this relationship, we assume that instances of the model elements to relate are read and written. This kind of subsuming is in the following alluded to as **strict subsumption**. For any two MODEL ELEMENTS m_1 and m_2, of which the former strictly subsumes the latter, this is denoted as $m_2 <: m_1$.

Burger and Gruschko discuss in [BG10] different kinds of *meta-model* adaptations and discuss to which extent these adaptations change the denotational semantics of the meta-model. Applying their discussions on EA information models, we revisit the requirements for weak and strict subsumption. In particular, they discuss how mandatory properties and relationship ends are of interest regarding information model adaptations. If the adapted information model suppresses a mandatory model element, it is not longer possible to use instantiations of the adapted information model to update data according to the underlying information model. In such cases, we are confined to weak subsumption. Conversely, strict subsumption demands that any mandatory model element from the subsumed information model is also present

in the subsuming one. Subsequently, we apply this understanding on the level of the individual model elements, raising special conditions for strict subsumption of RELATIONSHIPS, UNIVERSAL TYPES, and RELATIONSHIP TYPES.

For any two model elements m_1 and m_2 strict subsumption entails weak subsumption, shortly denoted as:

$$\frac{m_2 <: m_1}{m_2 \sqsubset: m_1} \text{ (WEAKEN)}.$$

The following characteristics hold for strict subsumption [Pi02, pages 182–183]:

- strict subsumption is *reflexive*, meaning that for any model element m the relationship $m <: m$ holds.
- strict subsumption is *transitive*, meaning that for any model elements m_1, m_2, and m_3 the following rule holds

$$\frac{m_1 <: m_2 \qquad m_2 <: m_3}{m_1 <: m_3} \text{ (S-TRANSITIVITY)}.$$

From the strict subsumption, we can further derive a principle of identity as the strict subsumption-relationship is *anti-symmetric*. This means that if two model elements mutually subsume each other, they are:

$$\frac{m_1 <: m_2 \qquad m_2 <: m_1}{m_1 \simeq m_2} \text{ (S-EQUIVALENCE)}.$$

In above rule a **structural equivalence**-relationship $\simeq$ is introduced. We thereby account for the fact that the strict subsumption-relationship $<:$ is a structural relationship. This means that two model elements are compared with respect to their structure to determine, if one element subsumes the other. The corresponding predicators are not taken into account. Especially in respect to the embedding into the viewmodels of viewpoints, structural subsumption is a relevant concept. For the evolution of information models, this relationship is contrariwise not sufficient.

From the reflexivity of the strict subsumption we infer that weak subsumption is also reflexive. Further, we demand that weak subsumption is also transitive, i.e., that the following rule holds:

$$\frac{m_1 \sqsubset: m_2 \qquad m_2 \sqsubset: m_3}{m_1 \sqsubset: m_3} \text{ (W-TRANSITIVITY)}.$$

In the following we explore the different kinds of subsumption-relationships connecting the concepts of the IM2L. The subsumption-relationship is of particular interest regarding temporal aspects. A time-dependent model element can be subsumed by both a time-dependent and a not time-dependent one, while subsumption in the opposite direction is not possible. The operational semantics in a situation, where a not time-dependent element subsumes a time-dependent one uses the **transaction time** also as the **valid time**. In respect to reading, i.e., to weak subsumption, the time-dependence doe not need to be accounted for.

Primitive properties

One primitive property subsumes another, if their types and cardinality constraints correspond. The subsuming property p_1 can make more strict prescriptions when it comes to cardinality constraints than the subsumed property p_2. For strict subsumption, the data-types of both properties have to be identical. For two properties, strict subsumption is defined as follows:

$$\begin{aligned} p_2 <: p_1 \Leftrightarrow \quad & p_2.\text{type} = p_1.\text{type} \wedge \\ & p_2.\text{lower} \leq p_1.\text{lower} \wedge \\ & p_1.\text{upper} \leq p_2.\text{upper} \wedge \\ & \neg p_2.\text{timedependent} \Rightarrow \neg p_1.\text{timedependent}. \end{aligned}$$

In respect to weak subsumption between properties, we have to account for reading compatibility between different primitive data-types. Thereby, we derive weak subsumption-relationships between these types, albeit data-types are no first-class model elements for the IM2L. These relationships are employed in the subsequent definition of weak subsumption between properties. An integer can be read as a decimal:

$$\mathsf{integer} \sqsubset: \mathsf{decimal}.$$

Similarly, we allow that any date is read as value of type dateTime:

$$\mathsf{date} \sqsubset: \mathsf{dateTime}.$$

Finally, a value of any type $d \in \{\mathsf{int}, \mathsf{bool}, \mathsf{float}, \mathsf{date}, \mathsf{dateTime}, \mathsf{money}\}$ can be represented as sequence of chars, i.e., a string:

$$d \sqsubset: \mathsf{string}.$$

The subsuming property p_1 can employ a weak counterpart of the primitive data-type supplied by the subsumed property p_2. In formal terms this reads as follows:

$$\begin{aligned} p_2 \sqsubset: p_1 \Leftrightarrow \quad & p_2.\text{type} \sqsubset: p_1.\text{type} \wedge \\ & p_2.\text{lower} \leq p_1.\text{lower} \wedge \\ & p_1.\text{upper} \leq p_2.\text{upper} \end{aligned}$$

Enumerations and enumeration properties

Any enumeration specifies a finite number of mutually distinct ENUMERATION LITERALS. The literals determine the instantiation of the enumeration, such that a structural subsumption-relationship depends on the number of supplied literals. This means that an enumeration e_1 strictly subsumes an enumeration e_2, if they have the same number of literals, formally:

$$e_2 <: e_1 \Leftrightarrow |e_1.\text{owns}| = |e_2.\text{owns}|.$$

It further holds that for any two enumerations $e_2 <: e_1$ also entails $e_1 <: e_2$, formally:

$$e_2 <: e_1 \Leftrightarrow e_1 <: e_2 \Leftrightarrow e_2 \simeq e_1.$$

In the sense of weak subsumption, the subsuming enumeration e_1 can make more strict prescriptions than the subsumed one e_2, i.e., can own less literals:

$$e_2 \sqsubset: e_1 \Leftrightarrow |e_1.\text{owns}| \leq |e_2.\text{owns}|.$$

We further establish a weak subsume relationship between enumerations and the primitive type string, in such sense that any enumeration literal can be read as the string that represents its predicator. Let $\mathbb{E}(\mathcal{M})$ be the set of all enumerations, the following holds:

$$\forall e \in \mathbb{E}(\mathcal{M}) : e \sqsubset: \mathsf{string}.$$

Based on above structural subsume-relationships between enumerations, we can reason on subsume-relationships between ENUMERATION PROPERTIES, formally defined as:

$$\begin{aligned} p_2 <: p_1 \Leftrightarrow\ & p_2.\text{type} \simeq p_1.\text{type} \wedge \\ & p_2.\text{lower} \leq p_1.\text{lower} \wedge \\ & p_1.\text{upper} \leq p_2.\text{upper} \wedge \\ & \neg p_2.\text{timedependent} \Rightarrow \neg p_1.\text{timedependent}. \end{aligned}$$

Above rules can be weakened with respect to the employed type relationship, reading as:

$$\begin{aligned} p_2 \sqsubset: p_1 \Leftrightarrow\ & p_2.\text{type} \sqsubset: p_1.\text{type} \wedge \\ & p_2.\text{lower} \leq p_1.\text{lower} \wedge \\ & p_1.\text{upper} \leq p_2.\text{upper} \end{aligned}$$

Relationships

A RELATIONSHIP acts as 'hub' and owns at least two RELATIONSHIP ENDS. Two relationships r_1 and r_2 participate in a subsume-relationship:

$$\begin{aligned} r_2 <: r_1 \Leftrightarrow\ & \forall e_2 \in \mathcal{E}^{0,1}(r_2)\ \exists e_1 \in \mathcal{E}^{0,1}(r_1) : e_2.\text{type} <: e_1.\text{type} \wedge \\ & \forall e_2 \in \mathcal{E}^{1,1}(r_2)\ \exists e_1 \in \mathcal{E}^{1,1}(r_1) : e_2.\text{type} <: e_1.\text{type} \wedge \\ & \forall e_2 \in \mathcal{E}^{0,\infty}(r_2)\ \exists e_1 \in \mathcal{E}^{0,\infty}(r_1) : e_2.\text{type} <: e_1.\text{type} \wedge \\ & \forall e_2 \in \mathcal{E}^{1,\infty}(r_2)\ \exists e_1 \in \mathcal{E}^{1,\infty}(r_1) : e_2.\text{type} <: e_1.\text{type} \wedge \\ & \neg r_2.\text{timedependent} \Rightarrow \neg r_1.\text{timedependent}. \end{aligned}$$

The weak subsume-relationship is less constrained with respect to the possible evaluations:

$$\begin{aligned} r_2 \sqsubset: r_1 \Leftrightarrow\ & \forall e_2 \in \mathcal{E}^{0,1}(r_2)\ \exists e_1 \in \mathcal{E}^{0,1}(r_1) : e_2.\text{type} \sqsubset: e_1.\text{type} \wedge \\ & \forall e_2 \in \mathcal{E}^{1,1}(r_2)\ \exists e_1 \in \mathcal{E}^{0,1}(r_1) \cup \mathcal{E}^{1,1}(r_1) : e_2.\text{type} \sqsubset: e_1.\text{type} \wedge \\ & \forall e_2 \in \mathcal{E}^{0,\infty}(r_2)\ \exists e_1 \in \mathcal{E}^{0,\infty}(r_1) : e_2.\text{type} \sqsubset: e_1.\text{type} \wedge \\ & \forall e_2 \in \mathcal{E}^{1,\infty}(r_2)\ \exists e_1 \in \mathcal{E}^{0,\infty}(r_1) \cup \mathcal{E}^{1,\infty}(r_1) : e_2.\text{type} \sqsubset: e_1.\text{type}. \end{aligned}$$

In the sense of above conditions, **specialization by constraint** as introduced in Section 6.1 implies a subsume-relationship. Let r be a relationship carrying a relationship end e of type t. From $t' <: t$ it follows that $r[e \downarrow t'] <: r$.

Universal types

For any universal type, the subsume-relationship builds on the carried properties as well as the targeting relationship ends. We can write the strict subsume-relationship between two universal types u_1 and u_2 as follows:

$$\begin{aligned} u_2 <: u_1 \Leftrightarrow\ & \forall p_1 \in u_1.\text{carries}\ \exists p_2 \in u_2.\text{carries} : p_1 <: p_2 \wedge \\ & \forall e_1 \in u_1.\text{targeting}\ \exists e_2 \in u_2.\text{targeting} : e_1 <: e_2 \wedge \\ & \neg u_2.\text{timedependent} \Rightarrow \neg u_1.\text{timedependent} \wedge \\ & \forall p_2 \in u_2.\text{carries}\ (p_2.\text{lower} = 1 \Rightarrow \exists p_1 \in u_1.\text{carries} : p_1 <: p_2) \wedge \\ & \forall e_2 \in u_2.\text{targeting}\ (e_2.\text{lower} = 1 \Rightarrow \exists e_1 \in u_1.\text{targeting} : e_1 <: e_2). \end{aligned}$$

The corresponding weak subsume-relationship between u_1 and u_2 is defined as

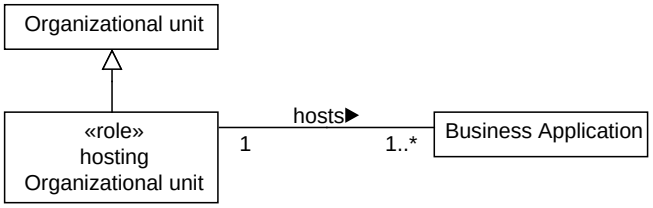
$$u_2 \sqsubset: u_1 \Leftrightarrow \begin{array}{l} \forall p_1 \in u_1.\text{carries}\ \exists p_2 \in u_2.\text{carries} : p_1 \sqsubset: p_2 \wedge \\ \forall e_1 \in u_1.\text{targeting}\ \exists e_2 \in u_2.\text{targeting} : e_1 \sqsubset: e_2. \end{array}$$

The strict subsume-relationship is consistent with the inheritance relationship introduced above. Any universal type u_2 inheriting from another universal type u_1 conversely is subsumed by this type. The exact kind of the types, distinguished along the dimensions of **rigid** vs. **non-rigid** and **dispersive** or not, is further relevant for the subsume-relationship. Let in the following $\mathbb{U}$ be the set of universal types[3] and let $\mathbb{U}^{+R}$ be the rigid, $\mathbb{U}^{\neg R}$ be the non-rigid, $\mathbb{U}^{+I}$ the non-dispersive and $\mathbb{U}^{\neg I}$ the dispersive types. We can formalize three prerequisites for a subsume-relationship as follows:

- No non-dispersive type can subsume a dispersive type $\forall u \in \mathbb{U}^{\neg I}\ \nexists u' \in \mathbb{U}^{+I} : u \sqsubset: u'$.
- No non-rigid type can subsume a rigid type $\forall u \in \mathbb{U}^{+R}\ \nexists u' \in \mathbb{U}^{\neg R} : u \sqsubset: u'$.
- Every non-rigid and non-dispersive type must be subsumed by a rigid and non-dispersive type $\forall u \in \mathbb{U}^{+I} \cap \mathbb{U}^{\neg R} \exists u' \in \mathbb{U}^{+I} \cap \mathbb{U}^{+R} : u \sqsubset: u'$.

In line with the arguments brought forward by Guizzardi in [Gu05], we in addition can establish more sophisticated subsume-relationships involving non-rigid types. Two particular relationships are of interest here. Firstly, we revisit the aforementioned correspondence between anti-rigid types (PHASES) and a substantial type with an ENUMERATION PROPERTY . Secondly, we establish mutual subsume-relationships to reflect the correspondence between a substantial type with an **optional** RELATIONSHIP END and a semi-rigid type (ROLE). Example 6.5 illustrates later kind of correspondence.

Example 6.5: Deriving roles from optional relationships. The enterprise seeks to model that organizational units may but not necessarily do host business applications. Thereto, they introduce a distinction between 'normal' organizational units and ones taking the role of 'hosting' applications. The corresponding information model looks as follows.



[3] In later discussions, we will understand $\mathbb{U}$ as an operator filtering the model elements to the universals only.

This subsume-relationships results from **specialization by constraint**. Let therein t be a universal type with an optional relationship end e of type r, we can derive a type $t' = t[e \downarrow r]$. For instances of t' the corresponding relationship end is not longer optional, yielding that $t' <: t$ holds.

In a similar way, we apply specialization by constraint to a type t holding an enumeration property p, of which the corresponding enumeration supplies enumeration literals $l_1 \dots l_n$. We can derive types $t_1 \dots t_n$ defined as follows: $t_i = t[p/l_i]$ for $1 \leq i \leq n$. In case the enumeration property is optional, we can further derive another type $t_\emptyset = t[p/\emptyset]$. Any instance of t can in this respect be described as instance of one of the corresponding specialized types, which in turn have to be designated as **anti-rigid**, as the property can change its value.

Relationship types

The subsume-relationship targeting two relationship types r_1 and r_2 is a refinement both of the subsume-relationship between two universal types and of the subsume-relationship between two relationships. We introduce two function reducing the relationship type to its underlying universal type $\bar{u}(r)$ or its underlying relationship $\bar{r}(r)$. The strict subsume-relationship between the two relationship types is defined as

$$r_2 <: r_1 \Leftrightarrow \bar{u}(r_2) <: \bar{u}(r_1) \wedge \bar{r}(r_2) <: \bar{r}(r_1).$$

Relationship type r_1 weakly subsumes relationship type r_2, if

$$r_2 \sqsubset: r_1 \Leftrightarrow \bar{u}(r_2) \sqsubset: \bar{u}(r_1) \wedge \bar{r}(r_2) \sqsubset: \bar{r}(r_1).$$

Information models

Based on the subsume-relationships defined above, we can derive the conditions under which one information model represented as set $\mathcal{M}_1$ of model elements consistently embeds into another information model $\mathcal{M}_2$. Textually described the embedding-relationship demands that any type in the first information model is subsumed by a corresponding one in the second information model. Therefore, three conditions for the strict embedding-relationship can be formulated depending on the ENUMERATIONS $\mathbb{E}(\mathcal{M}_1) \subseteq \mathcal{M}_1$, UNIVERSAL TYPES $\mathbb{U}(\mathcal{M}_1) \subseteq \mathcal{M}_1$, and RELATIONSHIPS $\mathbb{R}(\mathcal{M}_1) \subseteq \mathcal{M}_1$ contained in the information models. These relationships read as:

$$\begin{aligned} \mathcal{M}_2 <: \mathcal{M}_1 \Leftrightarrow \quad & \forall e_1 \in \mathbb{E}(\mathcal{M}_1)\ \exists e_2 \in \mathbb{E}(\mathcal{M}_2) : e_1 <: e_2 \wedge \\ & \forall u_1 \in \mathbb{U}(\mathcal{M}_1)\ \exists u_2 \in \mathbb{U}(\mathcal{M}_2) : u_1 <: u_2 \wedge \\ & \forall r_1 \in \mathbb{R}(\mathcal{M}_1)\ \exists r_2 \in \mathbb{R}(\mathcal{M}_2) : r_1 <: r_2. \end{aligned}$$

Similarly, we can establish a weak embedding-relationship between information models $\mathcal{M}_1$ and $\mathcal{M}_2$ based on the weak subsume-relationships between their model elements:

$$\begin{aligned} \mathcal{M}_2 \sqsubset: \mathcal{M}_1 \Leftrightarrow \quad & \forall e_1 \in \mathbb{E}(\mathcal{M}_1)\ \exists e_2 \in \mathbb{E}(\mathcal{M}_2) : e_1 \sqsubset: e_2 \wedge \\ & \forall u_1 \in \mathbb{U}(\mathcal{M}_1)\ \exists u_2 \in \mathbb{U}(\mathcal{M}_2) : u_1 \sqsubset: u_2 \wedge \\ & \forall r_1 \in \mathbb{R}(\mathcal{M}_1)\ \exists r_2 \in \mathbb{R}(\mathcal{M}_2) : r_1 \sqsubset: r_2. \end{aligned}$$

Predicated subsume-relationship

The **predicated subsume**-relationships further considerations on subsumption to the level of predicators. Such relationships are necessary, as pure structural subsume is likely to raise 'false positives', when it comes to relationships between different model elements. Example 6.6 illustrates that fact.

Example 6.6: Structural vs. predicated subsumes. Consider an EA information model, containing two classes 'business application' and 'person', respectively. Both classes each bear only one property 'name' as shown in the figure below.

The business application structurally subsumes the person and vice versa, making them structurally equivalent.

We have to introduce an additional relationship that can be applied on all model elements. We consider two model elements m_1 and m_2 as **equivalently predicated** (denoted as $m_1 \triangleq m_2$), if they are assigned to the same predicator. An assignment to the same predicator is a necessary prerequisite for two model elements participating in a predicated subsume-relationship. For any two model elements m_1 and m_2 (not being enumerations) a **updating subsume** holds, if:

$$\frac{m_1 <: m_2 \qquad m_1 \triangleq m_2}{m_1 <:: m_2} \text{ (U-SUBSUME)}.$$

In a similar vein, we can establish a **reading subsume**-relationship on arbitrary model elements m_1 and m_2 (not being enumerations) as follows:

$$\frac{m_1 \sqsubset: m_2 \qquad m_1 \triangleq m_2}{m_1 \sqsubset:: m_2} \text{ R-SUBSUME)}.$$

A minor exception from above rules has to be made in respect to the predicated subsume between two enumerations e_1 and e_2. This relationship depends on the predicators of the contained ENUMERATION LITERALS and formally reads as follows:

$$e_2 <:: e_1 \Leftrightarrow e_1 \triangleq e_2 \wedge e_1.\text{owns} = e_2.\text{owns}.$$

The equality in the second condition means that the two sets contain an equal number of elements, such that we find a bijection that connects any literal from the first set to an

equivalently predicated one in the second set. A similar consideration applies to the reading subsume between two enumerations e_1 and e_2, which depends on the predicators of the ENUMERATION LITERALS, reading as follows:

$$e_2 \sqsubset:: e_1 \Leftrightarrow e_1 \triangleq e_2 \wedge \forall l_1 \in e_1.\text{owns}\ \exists l_2 \in e_2.\text{owns} : l_2 \triangleq l_1.$$

6.2.3 Aggregation-relationships

Information models represent conceptualizations of at least one particular area-of-interest in the EA. A comprehensive EA information model underlying one or more consistent EA modeling languages is designed to cover different areas-of-interest. Against the background of the different ways how conceptualizations can be related by **aggregation**, we establish techniques for integrating information models. Thereto, we discuss the operational semantics of the different types of aggregation as introduced in Section 6.1, namely **aggregation by composition** (denoted as $\uplus$), **aggregation by weak generalization** (denoted as $\sqcup$), and **aggregation by partial composition** (denoted as $\sqcap$). It would be possible to establish these types of aggregation on the different model elements that constitute an information model. Such embracing set of aggregation operations is nevertheless way to expressive for the utilization scenario of the IM2L. In addition, mechanisms of that generality would imply a rich formalism for integrating arbitrary operations on the instantiations of all admissible primitive types. Therefore, we restrict our considerations on aggregating UNIVERSAL TYPES and RELATIONSHIPS, or appropriate specializations thereof. In terms of primitive operations, such aggregations realize *selections* and *joins* on information models [Da00].

Aggregation by composition

Aggregation by composition describes a case, where all the information supplied by instances of two distinct model elements is represented in instances of one model element in an unambiguous manner. This means that changing the represented information in an instance of the aggregating element directly maps to a changes in the instances of the underlying model elements. The requirement of being unambiguous demands that given an instance of the aggregating element, there must be mechanisms to determine which instances of the two aggregated model elements are meant, which necessitates an identity condition on the model elements. Thereto, aggregation by composition is restricted to work on SORTAL TYPES only. According to our rules on subsuming between non-dispersive types, it immediately follows that the aggregating model element also is a SORTAL TYPE. Let t_2 and t_3 be sortal types and $t_1 := t_2 \uplus t_3$ denote the aggregated type. The unambiguous mapping demands that $t_2 <: t_1$ and $t_3 <: t_1$ hold. From this we can derive that, whenever one of the aggregated types is either anti-rigid or semi-rigid that the same also applies for the aggregating type.

Going down to the individual PROPERTIES and RELATIONSHIP ENDS that detail the nature of a sortal type, the following rules delineate how aggregation by composition is operationalized:

1. All properties of each of the aggregated sortal types, if uniquely predicated, are also present in the aggregating sortal type.
2. All properties shared by both aggregated sortal types are unified, if possible, to one property in the aggregating sortal type.

3. All relationship ends of each of the aggregated sortal types, if uniquely predicated, are also present in the aggregating sortal type.
4. All relationship ends shared by both aggregated sortal types are unified, if possible, to one relationship end in the aggregating sortal type.

Rules 2 and 4 imposes a condition for synthesizing a new sortal type from two underlying sortal types. In respect to equivalently predicated properties and relationship ends aggregation by composition is only possible, if one property (or relationship end) subsumes the other:

$$\begin{aligned}\exists t_1 = t_2 \uplus t_3 \Leftrightarrow \quad & (\forall p_2 \in t_2.\text{owns}, p_3 \in t_3.\text{owns} : p_2 \triangleq p_3 \Leftrightarrow p_2 <: p_3 \vee p_3 <: p_2) \wedge \\ & (\forall e_2 \in t_2.\text{targeting}, e_3 \in t_3.\text{targeting} : e_2 \triangleq e_3 \Leftrightarrow e_2 <: e_3 \vee e_3 <: e_2).\end{aligned}$$

Aggregation by composition can further be applied sortal types that have be derived by **specialization by constraint**. Let t be a sortal type carrying two properties p_1 as well as p_2. Further, the sortal type is target of relationship ends e_1 and e_2 carried by relationships r_1 and r_2. Let in addition be $\mathcal{I}_1', \mathcal{I}_1'' \subseteq \mathbb{I}[p_1.\text{type}]$ and $\mathcal{I}_2' \subseteq \mathbb{I}[p_2.\text{type}]$. Further $r_1' <: r_1$ and $r_2' <: r_2$ holds. In this context the following simplification rules on specialized sortal types apply:

$$t[p_1/\mathcal{I}_1'] \quad \uplus \quad t[p_1/\mathcal{I}_1''] \quad = \quad \begin{cases} \bot & \text{if } \mathcal{I}_1' \neq \mathcal{I}_1'' \\ t[p_1/\mathcal{I}_1'] & \text{if } \mathcal{I}_1' = \mathcal{I}_1'' \end{cases},$$

$$t[e_1 \downarrow r_1'] \quad \uplus \quad t[e_1 \downarrow r_1''] \quad = \quad \begin{cases} \bot & \text{if } r_1' \neq r_1'' \\ t[e_1 \downarrow r_1'] & \text{if } r_1' = r_1'' \end{cases},$$

$$t[p_1/\mathcal{I}_1'] \quad \uplus \quad t[p_2/\mathcal{I}_2'] \quad = \quad t[p_1/\mathcal{I}_1'][p_2/\mathcal{I}_2'],$$

$$t[e_1 \downarrow r_1'] \quad \uplus \quad t[e_2 \downarrow r_2'] \quad = \quad t[e_1 \downarrow r_1'][e_2 \downarrow r_2'],$$

$$t[p_1/\mathcal{I}_1'] \quad \uplus \quad t[e_1 \downarrow r_1'] \quad = \quad t[p_1/\mathcal{I}_1'][e_1 \downarrow r_1'].$$

Aggregation by weak generalization

Aggregation by weak generalization describes a case, where instances of two distinct model elements are generalized to instances of one model element that carries a part of the information supplied by the aggregated instances. By generalization at least one part of information is removed, namely the distinction between the two generalized types. In this respect, aggregation by generalization is designed for uniformly treating the instances of both aggregated types with respect to reading. To facilitate our subsequent discussions we introduce an abbreviated formalism on arbitrary model elements m_1, m_2, and m_3:

$$m_1 = m_2 \oplus m_3 \Leftrightarrow m_2 \sqsubset:: m_1 \wedge m_3 \sqsubset:: m_1.$$

For universal types t_2 and t_3 with $t_1 = t_2 \sqcup t_3$ this condition is reflected in the rules for aggregation. Any property of the aggregating universal type weakly and predicated subsumes a shared property of both aggregated universal types.

$$\forall p_2 \in t_2.\text{carries}, p_3 \in t_3.\text{carries} : (p_2 \triangleq p_3 \Leftrightarrow \exists p_1 \in t_1.\text{carries} : p_1 = p_2 \oplus p_3).$$

The type for the aggregating properties can be less strict as any of the types supplied by their aggregated counterparts. Two properties p_2 and p_3 that are differently typed are aggregated by weak generalization to a property that supplies a type that weakly subsumes the types of these properties, being at least the primitive type string.

We establish an aggregation by generalization applying on relationships r_2 and r_3 to a relationship $r_1 = r_2 \sqcup r_3$. For any owned relationship end of the aggregating relationship one equivalently predicated relationship end exists in each of the aggregated relationships:

$$\forall e_2 \in r_2.\text{carries}, e_3 \in r_3.\text{carries} : (e_2 \triangleq e_3 \Leftrightarrow \exists e_1 \in r_1.\text{carries} : e_1 = e_2 \oplus e_3).$$

The relationship ends e_2 and e_3 are aggregated by weak generalization to a relationship end that targets a substantial type, which weakly subsumes the corresponding substantial types. Such substantial type can be constructed by aggregating both supplied substantial types by weak generalization $e_1.\text{targets} = e_2.\text{targets} \sqcap e_3.\text{targets}$.

Substantial types s_2 and s_3 with $s_1 = s_2 \sqcup s_3$ are aggregated by weak generalization in a similar way. In particular, the mechanism for aggregating the properties is also applied for aggregating the targeting relationship ends:

$$\begin{aligned}&\forall p_2 \in s_2.\text{carries}, p_3 \in s_3.\text{carries} : (p_2 \triangleq p_3 \Leftrightarrow \exists p_1 \in s_1.\text{carries} : p_1 = p_2 \oplus p_3) \wedge \\ &\forall e_2 \in s_2.\text{targeting}, e_3 \in s_3.\text{targeting} : (e_2 \triangleq e_3 \Leftrightarrow \exists e_1 \in s_1.\text{targeting} : e_1 = e_2 \oplus e_3).\end{aligned}$$

The relationship ends e_2 and e_3 are aggregated by weak generalization to a relationship end that is carried by a relationship (or relationship type), which weakly subsumes the relationships (or relationship types) supplied by the two aggregated ends. This relationship (or relationship type) results from aggregating both relationships (or relationship types) by weak generalization $e_1.\text{owns} = e_2.\text{owns} \sqcap e_3.\text{owns}$.

Finally, we can describe the operator for generalizing two relationship type r_2 and r_3 to one relationship type r_1. For the carried properties of both relationship types carries_1), the rule established with the universal types has to apply, whereas regarding the carried relationship ends carries_2), the corresponding rule established for relationships has to hold. For $r_1 = r_2 \sqcup r_3$ this reads as:

$$\begin{aligned}&\forall p_2 \in r_2.\text{carries}_1, p_3 \in r_3.\text{carries}_1 : (p_2 \triangleq p_3 \Leftrightarrow \exists p_1 \in r_1.\text{carries}_1 : p_1 = p_2 \oplus p_3) \wedge \\ &\forall e_2 \in r_2.\text{carries}_2, e_3 \in r_3.\text{carries}_2 : (e_2 \triangleq e_3 \Leftrightarrow \exists e_1 \in r_1.\text{carries}_2 : e_1 = e_2 \oplus e_3).\end{aligned}$$

Aggregation by weak generalization can be applied to universal types that result from **specialization by constraint**. Let t be a sortal type carrying a property p. Further, the sortal type is target of relationship end e carried by relationship r. Let in addition be $\mathcal{I}', \mathcal{I}'' \subseteq \mathbb{I}[p_1.\text{type}]$. Further $r', r'' <: r$ and $t', t'' <: t$ holds. In this context the following simplification rules on specialized sortal types apply:

$$\begin{aligned}t[p/\mathcal{I}'] \quad &\sqcup \quad t[p/\mathcal{I}''] \quad = \quad t[p/(\mathcal{I}' \cup \mathcal{I}'')], \\ t[e \downarrow r'] \quad &\sqcup \quad t[e \downarrow r''] \quad = \quad t[e \downarrow (r' \sqcup r'')], \\ r[e \downarrow t'] \quad &\sqcup \quad r[e \downarrow t''] \quad = \quad r[e \downarrow (t' \sqcup t'')].\end{aligned}$$

Aggregation by partial composition

Aggregation by partial composition describes a case, where instances of two distinct model elements contribute parts of their corresponding information to an instance of the composed model element. From a general point of view this corresponds to an operation that has both the character of a composition and of a generalization. With the the narrow definition of the composition confined to only sortal types, we nevertheless have to add a definition for the operation of partial composition. Due to its nature, where necessarily parts of the underlying information are lost, a model element resulting from partial composition is confined to reading access on the underlying model elements.

Two universal types t_2 and t_3 are aggregated by partial composition to $t_1 = t_2 \sqcap t_3$, when each property of the aggregating universal type can be read from corresponding properties of the aggregated types. Formally, this condition reads as:

$$\forall p_1 \in t_1.\text{owns} : \ (\exists p_2 \in t_2.\text{owns} : p_2 \sqsubset:: p_1) \vee (\exists p_3 \in t_3.\text{owns} : p_3 \sqsubset:: p_1).$$

Therein, the case, where equivalently predicated properties are supplied by both universal types, deserves special attention. In this case the less restrictive type (primitive type or enumeration) supplied by p_2 or p_3 is used. If neither $p_2.\text{type} \sqsubset: p_3.\text{type}$ nor $p_3.\text{type} \sqsubset: p_2.\text{type}$, it necessarily follows that property p_1 is of type string. Further, the cardinality constraints of p_1 are adapted to the corresponding ones of p_2 and p_3 as follows:

$$\begin{aligned} p_1.\text{lower} &= \begin{cases} 0 & \text{if } p_2.\text{lower} = 0 = p_3.\text{lower} \\ 1 & \text{otherwise} \end{cases} \\ p_1.\text{upper} &= \infty. \end{aligned}$$

Two relationships r_2 and r_3 are aggregated by partial composition to $r_1 = r_2 \sqcap r_3$, when each relationship end owned by the aggregating relationship can be read from a corresponding relationship end of the aggregated relationships. The formalization of this condition resembles the one provided for the properties of universal types and reads as follows:

$$\forall e_1 \in r_1.\text{owns} : \ (\exists e_2 \in r_2.\text{owns} : e_2 \sqsubset:: e_1) \vee (\exists e_3 \in r_3.\text{owns} : e_3 \sqsubset:: e_1).$$

In case two equivalently predicated relationship ends e_2 and e_3 are supplied, the less restrictive of the targeted substantial type is used. If neither of the types weakly subsumes its counterpart, a new type is introduced by weak generalization $e_1.\text{targets} = e_2.\text{targets} \sqcap e_3.\text{targets}$. Further, the cardinality constraints of r_1 are adapted to the corresponding ones of r_2 and r_3 as follows:

$$\begin{aligned} e_1.\text{lower} &= \begin{cases} 0 & \text{if } e_2.\text{lower} = 0 = e_3.\text{lower} \\ 1 & \text{otherwise} \end{cases} \\ e_1.\text{upper} &= \infty. \end{aligned}$$

In a similar vein, two substantial types s_2 and s_3 are aggregated by partial composition to $s_1 = s_2 \sqcap s_3$ both regarding the owned properties and the targeting relationship ends. This yields the following two conditions:

$$\begin{gathered} \forall p_1 \in s_1.\text{owns} : \ (\exists p_2 \in s_2.\text{owns} : p_2 \sqsubset:: p_1) \vee (\exists p_3 \in s_3.\text{owns} : p_3 \sqsubset:: p_1) \ \wedge \\ \forall e_1 \in s_1.\text{targeting} : \ (\exists e_2 \in s_2.\text{targeting} : e_2 \sqsubset:: e_1) \vee (\exists e_3 \in s_3.\text{targeting} : e_3 \sqsubset:: e_1). \end{gathered}$$

The discussion on the type of the aggregating properties as undertaken above on the universal types also applies for the properties of substantial types. For the targeting relationship ends the least restrictive owning relationship (or relationship type) is used. If neither $e_2.\text{owns} \sqsubset:: e_3.\text{owns}$ nor $e_3.\text{owns} \sqsubset:: e_2.\text{owns}$ holds, a new type is introduced by weak generalization as follows: $e_1.\text{owns} = e_2.\text{owns} \sqcup e_3.\text{owns}$. Again above rules for adapting the cardinality constraints hold:

$$\begin{aligned} p_1.\text{lower} &= \begin{cases} 0 & \text{if } p_2.\text{lower} = 0 = p_3.\text{lower} \\ 1 & \text{otherwise} \end{cases} \\ p_1.\text{upper} &= \infty. \end{aligned}$$

$$\begin{aligned} e_1.\text{lower} &= \begin{cases} 0 & \text{if } e_2.\text{lower} = 0 = e_3.\text{lower} \\ 1 & \text{otherwise} \end{cases} \\ e_1.\text{upper} &= \infty. \end{aligned}$$

Finally, we describe how two relationship types are aggregated. For the owned properties (owns_1) of r_2 and r_3 the rule established with the universal types is applied, whereas regarding the owned relationship ends (owns_2), the corresponding rule established for the relationships has to hold. For $r_1 = r_2 \sqcap r_3$ this read as:

$$\begin{aligned} &\forall p_1 \in s_1.\text{owns}_1 : \ (\exists p_2 \in s_2.\text{owns}_1 : p_2 \sqsubset:: p_1) \vee (\exists p_3 \in s_3.\text{owns}_1 : p_3 \sqsubset:: p_1) \wedge \\ &\forall e_1 \in r_1.\text{owns}_2 : \ (\exists e_2 \in r_2.\text{owns}_2 : e_2 \sqsubset:: e_1) \vee (\exists e_3 \in r_3.\text{owns}_2 : e_3 \sqsubset:: e_1). \end{aligned}$$

Again above rules for adapting the cardinality constraints hold:

$$\begin{aligned} p_1.\text{lower} &= \begin{cases} 0 & \text{if } p_2.\text{lower} = 0 = p_3.\text{lower} \\ 1 & \text{otherwise} \end{cases} \\ p_1.\text{upper} &= \infty. \end{aligned}$$

$$\begin{aligned} e_1.\text{lower} &= \begin{cases} 0 & \text{if } e_2.\text{lower} = 0 = e_3.\text{lower} \\ 1 & \text{otherwise} \end{cases} \\ e_1.\text{upper} &= \infty. \end{aligned}$$

Integrating information models

Based on the different types of aggregation described above, two different ways for integrating two information models can be derived:

- **Strict integration** in which only generalization and aggregation by composition are applied.
- **Weak integration** in which all types of aggregation are applied. In particular, aggregation by weak generalization or aggregation by partial composition have to be applied at least for one contained model element.

An information model resulting from the strict integration of two other information models is compatible with respect to both reading and updating operations on the instances of model elements. The information model resulting from a weak integration still support reading operations, but cannot be updated in the same way as the two integrated information models.

6.2.4 Notation of the IM2L

In this section we introduce the graphical notation for the IM2L. Due to the provenience of the UML as a language for describing EA information models, the graphical notation for the IM2L is designed to resemble the notation of the UML. In particular, we decide to utilize *stereotypes* to designate specific qualities of types, relationships, and properties. Thereby, we seek to make the information models accessible to modelers having no specific experience with the IM2L. Additionally, we use color-coding to facilitate the quick recognition of the different kinds of types. The color-coding applies to differentiate between ENUMERATIONS, different kinds of SUBSTANTIAL TYPES, and RELATIONSHIPS or RELATIONSHIP TYPES, respectively. Enumerations are colored green and are equipped with the stereotype «enumeration». Figure 6.11 shows the notation for an enumeration owning a literal denoted in a single row in the second compartment.

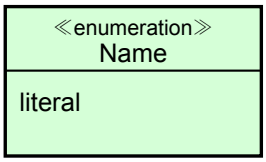


Figure 6.11: Notation of information models: ENUMERATIONS

Both the enumeration and its literals are labeled with the representations of the corresponding **predicators**.

In depicting SUBSTANTIAL TYPES the color-coding is used to distinguish between different kinds of such types with respect to the applicable meta-properties. The background colors of the symbols are used to designate whether the type is **rigid** (white), **anti-rigid** (blue), and **semi-rigid** (red). The border-color of the symbols is used to differentiate between **substantial** (black) and **dispersive** types (blue). With the two meta-properties being independent, this yields in total six different kinds of substantial types, for which the corresponding symbols and stereotypes are depicted in Table 6.2, further indicating **properties** owned by the types.

SUBSTANTIAL TYPE	**rigid**	**anti-rigid**	**semi-rigid**
non-dispersive	Name property:type[l..u]	≪phase≫ Name property:type[l..u]	≪role≫ Name property:type[l..u]
dispersive	≪category≫ Name property:type[l..u]	≪mixin≫ Name property:type[l..u]	≪roleMixin≫ Name property:type[l..u]

Table 6.2: Notation for information models: different kinds of SUBSTANTIAL TYPES

Both the substantial type and its properties are labeled with the representations supplied by the corresponding **predicators**. In respect to the notation for properties the corresponding values for lower and upper bound are supplied in brackets [a,b] after the property's data-type.

Therein, a can take the values 0 or 1, whereas b can take the values 1 or *. Later value indicates an upper bound set to ∞. If a specification of that kind is omitted, we assume per default that lower and upper bound are set to 1. The nominating property of a sortal type is designated via a 'key' symbol in front of the property specification as shown in Figure 6.12.

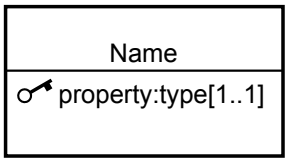


Figure 6.12: Notation for information models: **nominating property**

Another notational element is concerned with specialization and generalization relationships between different types in an information model or IBB. Such relationships are denoted as lines with an empty arrow head, pointing from the specialized type to the general one. Figure 6.13 shows the corresponding notation in action, indicating that the type SUB specializes the type SUPER.

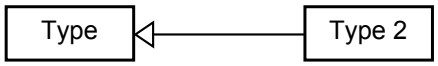


Figure 6.13: Notation for information models: **specialization**

For denoting RELATIONSHIPS and RELATIONSHIP TYPES, the notation of the IM2L goes beyond the one of the UML by introducing a dedicated symbol reflecting the hub-nature of a relationship. The relationship ends a complementary designated via lines connecting the corresponding substantial type with the relationship or relationship type. The cardinality constraints, i.e., lower and upper bound, of the relationship end are depicted at the 'inward' end of the line. The same applies for the corresponding **predicator** of the relationship end, whose representation is used to label the inward end. The symbol representing the 'hub' is labeled with the representation of the predicator assigned to the relationship or relationship type. On the 'outward' end of the line, i.e., the end pointing towards the substantial type, an 'artificial' cardinality 1 is supplied to mimic the way, relationships are designated in the UML. This notational specificity should help users familiar with the UML to understand the relationship semantics of the IM2L. Due to similar reasons, we further supply a notation alternative that applies to **binary relationships**, i.e., RELATIONSHIPS that are targeted by exactly two RELATIONSHIP ENDS. For such relationships, the symbolic representation of the 'hub' can be omitted and the cardinality constraints are moved to the opposite end. Table 6.3 displays the notations and notation alternatives for relationships and relationship types. For

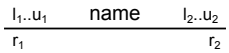
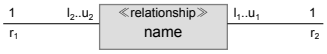
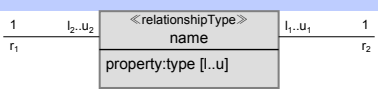
RELATIONSHIP (regular)	RELATIONSHIP (alternative)	RELATIONSHIP TYPE
		

Table 6.3: Notation for information models: RELATIONSHIPS and RELATIONSHIP TYPES

binary relationships the meta-property of being **acyclic** is supplied via a stereotype applied onto the corresponding relationship symbol, i.e., the 'hub' or the line.

Time-dependency of universal types or properties is designated via a 'clock' symbol attached to the corresponding element. Following Table 6.4 shows the notation for a time-dependent substantial type, a time-dependent property, and a time-dependent relationship.

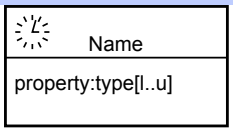
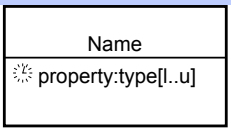
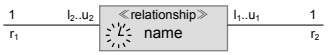
time-dependent SUBSTANTIAL TYPE	time-dependent PROPERTY	time-dependent RELATIONSHIP
		

Table 6.4: Notation for information models: time-dependent UNIVERSAL TYPES and PROPERTY

6.3 Viewpoint definition language (VDL)

The discussions in Section 4.3.5 promote a strongly mathematical understanding of the notation as a function relating syntactical concepts to their graphical counterparts. Each actual notation is realized by a bijective notation function, whereas also non-bijective representation functions exist, which are used to create symbolic EA representations, that cannot be used for modeling. Both functions can in line with our findings from [Er06b] be formalized building on a visualization model containing types (represented in the set Υ) and properties that can be used for describing visualizations.

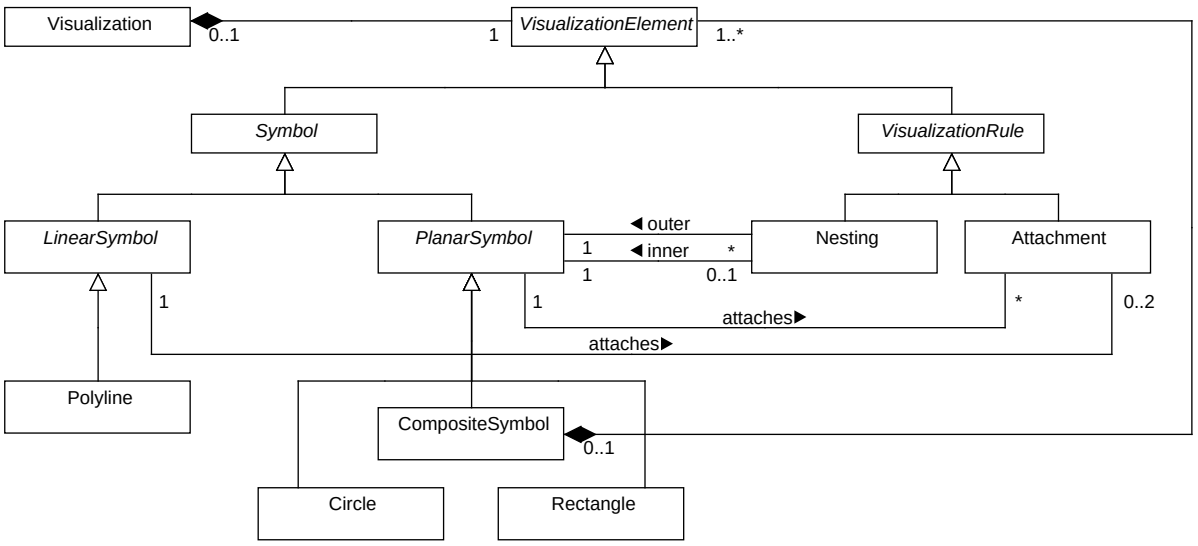


Figure 6.14: Concepts from the visualization model

Figure 6.14 shows a part of the contents of the visualization model. The model requires that any VISUALIZATION consists of one VISUALIZATIONELEMENT which in turn may be a COMPOSITESYMBOL that itself supplies one or more child elements. Different kinds of SYMBOLS may be distinguished: planar ones (Υ_{planar}) that extend over two dimensions and linear ones (Υ_{linear}) that are one-dimensional of nature. The model further introduces VISUALIZATIONRULES which define how different symbols relate to each other. For example the visualization rule NESTING designates that the INNER planar symbol is positioned completely inside the OUTER planar symbol. The second visualization rule exemplified, ATTACHMENT, demands that a linear symbol keeps attached, i.e., linked, to a corresponding planar symbol. In the following we revisit the transformation nature of a viewpoint that translates instances of universal types $\mathcal{U}$ from the viewmodel to instances of visualization elements Υ. We understand that such transformation is constituted from sub-transformations, which instantiate configurable and re-usable viewpoint building blocks. To provide a consistent description of such building blocks, we establish the VDL.

6.3.1 Syntax of the VDL

We introduce a type γ that acts as container for the instances of the universal types, i.e., denotes the semantic model. Based thereon, the viewpoint function can be described as

$$\mathbb{I}_{\mathcal{K}}[\gamma] \rightarrow \mathbb{I}[\Upsilon].$$

This function operates an instance of the viewmodel's container and maps the corresponding child elements to an instance from the visualization model, usually a COMPOSITE SYMBOL. Three different types of transformations constituting a viewpoint can be distinguished along the signatures of the corresponding notation functions, as to say we distinguish three types of **elementary VBBs**. As part of the signature, an arbitrary parameter from the set of parameters, subsequently denoted as $\mathcal{P}$ can occur:

Symbol VBB transforming instances of a universal type to instances of a visual type. This yields a signature, of which we distinguish two facets as follows:

Root VBB transforming instances of the container type γ to instances of COMPOSITE SYMBOL, reading as $\mathbb{I}[\gamma] \rightarrow \mathbb{I}[\Upsilon]$.

General symbol VBB transforming instances of one universal types u_1 to instances of SYMBOL. A general symbol VBB can be equipped with parameters $\mathcal{P}$, e.g. for selecting symbol types. The signature reads as $\mathbb{I}[u_1] \times \mathbb{I}[\mathcal{P}] \rightarrow \mathbb{I}[\Upsilon]$.

Decorating VBB augmenting an instance of a visual type returned by a decorated VBB with additional visual information derived in accordance to another parameter $p \in \mathcal{P}$. The signature reads as $\mathbb{I}[u_1] \times (\mathbb{I}[u_1] \rightarrow \mathbb{I}[\Upsilon]) \times \mathbb{I}[p] \rightarrow \mathbb{I}[\Upsilon]$.

Structural VBB transforming instances of a universal type to instances of a visual type according to a given visualization structure. In this transformation $N \geq 2$ sub-transformations can be employed and parameters $\mathcal{P}$ can exist. The signature reads as $\mathbb{I}[u] \times \prod_{n=1}^{N}(\mathbb{I}[u_n] \rightarrow \mathbb{I}[\Upsilon]) \times \mathbb{I}[\mathcal{P}] \rightarrow \mathbb{I}[\Upsilon]$.

A parameter of a VBB exports, i.e., makes accessible, the value of a specific **variable** on which the operational semantics of the building block depends. The actual quality of the parameter depends on the intended usage context in the VBB's inherent transformation, which binds to the underlying variable. We distinguish four types of variables: **universal type variables**, **property variables**, **relationships variables**, and **symbol variables**. Variables of the former three types bind to corresponding concepts from the viewmodel, whereas symbol variables bind to classes in the visualization model. The variables targeting viewmodel concepts can further be subject to additional constraints, specified as **lower bound** for the value assignments. If the variable is bound to an actual model element, the element specified as lower bound must subsume the supplied value in a structural sense. Let v be a variable and m_1 and m_2 model elements supplied for the value, we distinguish between:

- strictly admissible assignments with $m_1 <: v$.lower bound and
- weakly admissible assignments with $m_2 \sqsubset: v$.lower bound.

Figure 6.15 details the syntax of the VDL with its different types of variables and shows the mechanism for specifying lower bounds. Model elements from the IM2L, used to constrain the valid value assignments, are highlighted in the model.

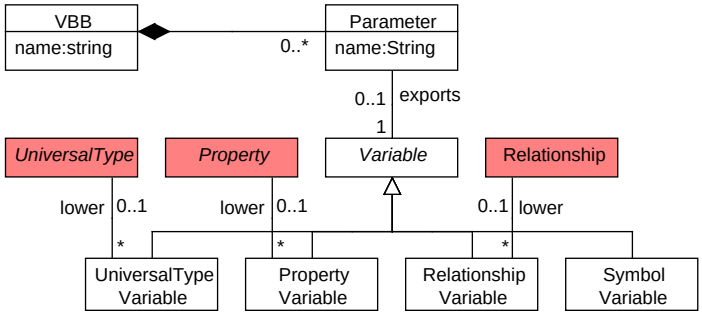


Figure 6.15: Variable concept in the VDL

Each parameter and thereby each underlying variable is bound to one value during the configuration of the viewpoint. The restriction to exactly one value does not limit the expressiveness of the viewpoint definitions. One can assume that whenever more than one value, e.g. universal type, was to be used, we can construct an appropriate type by **weak generalization**. Central to the syntax of VBBs are the concepts of the SINK and SOURCE. A SINK specifies a part of the domain of the VBB's corresponding function. Conversely, a SOURCE designates a part of the range of the VBB's corresponding function. This in particular applies to VBBs that are supplied with other VBBs as sub-transformations. One or more sinks and one or more sources are grouped to a TRANSFORMATION PORT according to logical dependencies, i.e., corresponding sinks and sources for invoking a particular sub-transformation are grouped to one transformation port in order to designate their logical relatedness. In respect to the direction of the transformation port we distinguish INPUT PORTS and OUTPUT PORTS. Figure 6.16 shows the part of the VDL syntax that is used to describe the ports, sinks, and sources of a VBB.

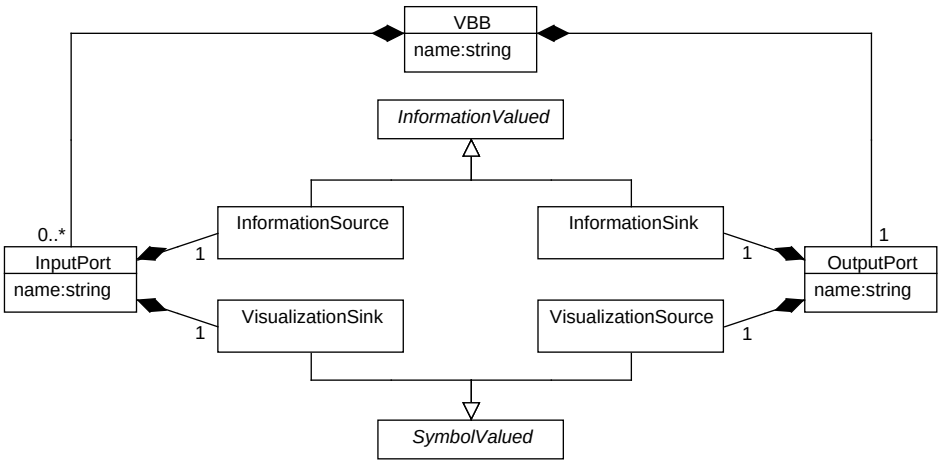


Figure 6.16: Port, sink, and source concept in the VDL

The operational semantics implemented in the corresponding VBB raises a relation between the sinks as well as sources on the one hand and the variables on the other hand. Every SYMBOL VALUED, i.e., every sink or source for symbol instances, is bound to a symbol from the visualization model. In the same sense, every INFORMATION VALUED, i.e., any sink or source for instances from the viewmodel, is bound to a particular universal type. Such universal type is either supplied by the different types of variables in different ways:

- The UNIVERSAL TYPE VARIABLE directly supplies a universal type.
- The PROPERTY VARIABLE allows to derive the universal type under consideration from the supplied property, where the carrying type is taken.
- The RELATIONSHIP VARIABLE supplies a n-ary relationship between n universal types. The actually considered type is specified by the index AT of the type to be taken.

Figure 6.17 shows the corresponding part from the syntax of the VDL. For any information valued concept exactly one of the relationships has to be set. A configured viewpoint composed from instantiated VBBs supplies value-bindings for INFORMATION VALUED and SYMBOL VALUED propagated from the parameters and their values.

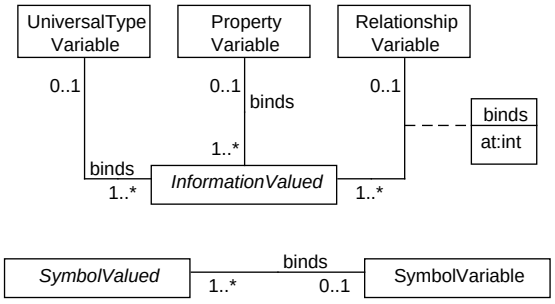


Figure 6.17: Bindings between variables, sinks, and sources in the VDL

6.3.2 Notation of the VDL

The graphical elements shown in Table 6.5 are used to denote SINKS and SOURCES of different types. In addition thereto a VBB supplies TRANSFORMATION PORTS and supply PARAMETERS, i.e., properties, that bind to a specific type. Figure 6.18 shows the symbol used to designate a VBB with a parameter SYMBOLTYPE and an output transformation port.

	Information object	Visualization object
Source		
Sink		

Table 6.5: Notation for VBBs: SOURCES and SINKS

For example, a parameter can expect to be supplied with a particular UNIVERSAL TYPE, PROPERTY, or RELATIONSHIP. In the following exemplary VBBs, we illustrate how transformation ports as well as parameters are depicted, and how they relate to the VBBs' signature.

Create Root Symbol

This VBB provides one output transformation port consuming an instance of the container type γ and creates an instance of COMPOSITE SYMBOL. This yields a signature as

$$\mathsf{cr} : \mathbb{I}[\gamma] \rightarrow \mathbb{I}[\Upsilon].$$

Create Planar Symbol

This VBB provides one transformation port and supplies a parameter for selecting the type of planar symbol to create. The corresponding signature reads as

$$\mathsf{cp} : \mathbb{I}_{\mathcal{K}}[u_1] \times \Upsilon_{planar} \rightarrow \mathbb{I}[\Upsilon_{planar}],$$

whereby Υ_{planar} denotes the set of PLANAR SYMBOLS supplied in the visualization model. Currying the VBB with an actual planar symbol $\upsilon \in \Upsilon_{planar}$, the following condition holds:

$$\forall i \in \mathbb{I}_{\mathcal{K}}[u_1] : \mathsf{cp}(i, \upsilon) \in \mathbb{I}[\upsilon].$$

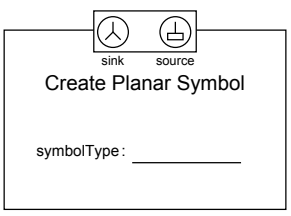


Figure 6.18: Exemplary VBB: create planar symbol (cp)

The VBB cp is denoted as shown in Figure 6.18. A single transformation port with a sink for instances of universal types and a source for instances of visualization elements is provided together with a parameter for selecting the visualization element to create.

The universal type, on whose instances this VBB operates, is designated by the variable u_1. The VBB further specifies a lower bound for this variable via a universal type u that supplies a nominating property of type string. The value of this property is used to label the instances of the symbol created by this VBB. By the lower bound, the viewmodel of the VBB is partially determined, demanding that any valid viewmodel is structurally subsumed by the one shown in Figure 6.19.

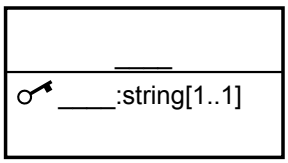


Figure 6.19: Exemplary viewmodel: create planar symbol (cp)

Create Cluster

This VBB provides three transformation ports, two input ports for sub-transformations and one output port along which the VBB can be used externally. In addition, a parameter representing the relationship between the clustered symbols is provided. With u_1 and u_2 being sortal types, this yields the following signature

$$\mathsf{cc} : \mathbb{I}_{\mathcal{K}}[u_1] \times (\mathbb{I}_{\mathcal{K}}[u_1] \to \mathbb{I}[\Upsilon_{\text{planar}}]) \times (\mathbb{I}_{\mathcal{K}}[u_2] \to \mathbb{I}[\Upsilon_{\text{planar}}]) \times (\mathbb{I}_{\mathcal{K}}^{1,1}[u_1] \to \mathbb{I}_{\mathcal{K}}^{0,\infty}[u_2]) \to \mathbb{I}[\Upsilon_{\text{planar}}].$$

Therein $\mathbb{I}_{\mathcal{K}}^{1,1}[u_1] \to \mathbb{I}_{\mathcal{K}}^{0,\infty}[u_2]$ designates a single parameter of the VBB cc, which is denoted as shown in Figure 6.20. One port receives an instance of universal types and provide an instance of visualization elements, thereby constituting the VBB's output port. Two ports each provide an instance of universal types and receive an instance of visualization elements. These ports import the results of sub-transformations. This VBB does not supply lower bounds for the

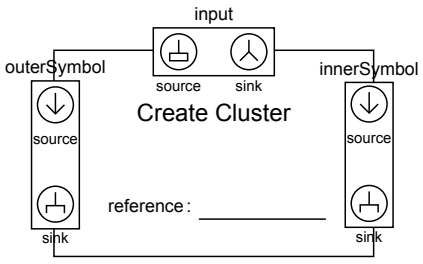


Figure 6.20: Exemplary VBB: create cluster (cc)

corresponding variables. The binding of the parameters to the variables partially determines the viewmodel of the VBB, demanding that any valid viewmodel is subsumed by the one shown in Figure 6.21.

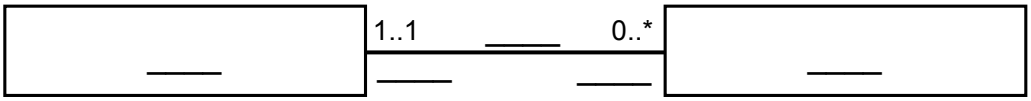


Figure 6.21: Exemplary viewmodel: create cluster (cc)

Decorate with Color-coding

This VBB provides two transformation ports, one output port along which the VBB can be used externally and one input port for sub-transformations. Furthermore, the VBB specifies a parameter representing a property that applies to the universal type whose instances are supplied in the input port. The value of the specified property is used to determine the color of the output visualization element. Color-coding can, as Krogmann et al. [Kr09] showed, be used to convey additional information of a substantial architecture element in a quickly conceivable manner. With d being any arbitrary primitive type or enumeration, the signature of dcc reads as

$$\mathsf{dcc} : \mathbb{I}_{\mathcal{K}}[u_1] \times (\mathbb{I}_{\mathcal{K}}^{1,1}[u_1] \to \mathbb{I}^{1,1}[d]) \times (\mathbb{I}_{\mathcal{K}}[u_1] \to \mathbb{I}[\Upsilon]) \to \mathbb{I}[\Upsilon].$$

Therein $\mathbb{I}_{\mathcal{K}}^{1,1}[u_1] \to \mathbb{I}^{1,1}[d]$ designates the single parameter of the VBB. Figure 6.22 shows how this signature can be described in terms of the graphical notation introduced above. This VBB

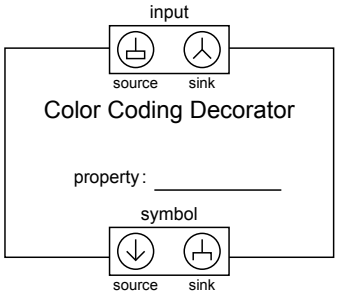


Figure 6.22: Exemplary VBB: decorate with color-coding (dcc)

does not supply lower bounds for the corresponding variables. The binding of the parameters to the variables partially determines the viewmodel of the VBB, demanding that any valid viewmodel is subsumed by the one shown in Figure 6.23.

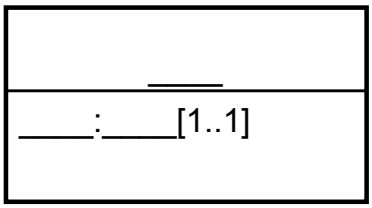


Figure 6.23: Exemplary viewmodel: decorate with color-coding (dcc)

6.3.3 Integrating and configuring viewpoints

The building block-based approach to viewpoint definition centers around the integration of VBBs. This means that different VBBs can be combined to more comprehensive VBBs, called **hybrid VBBs**. A combination of VBBs is described via connections between corresponding sinks and sources. Figure 6.25 illustrates how such connections are graphically represented in the VDL notation. During configuration and integration, any sink for information objects is connected to exactly one source of information objects. Furthermore, any sink for visualization objects is connected to one source of visualization objects. One output port remains unbound. Via this port the symbolic model $\mathbb{I}[\gamma]$ is supplied to the hybrid VBB. Figure 6.24 displays the syntax for the integration and configuration and introduces relationships that are set during these activities. These relationships are designated by the *stereotype* «c».

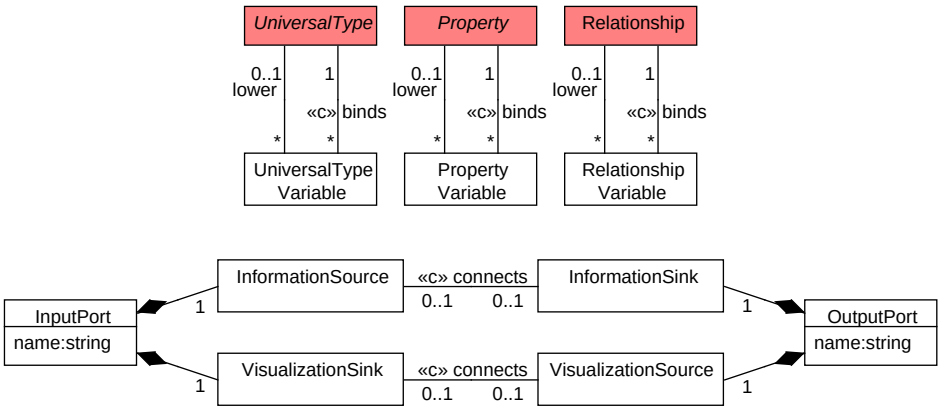


Figure 6.24: Connections between source and sink as well as variable assignment in the VDL

The corresponding rule of combination builds on the principle of currying. Thereby, more complex VBBs, e.g. structural or decorating VBBs, are transformed to functions taking less parameters. During the application of currying as part of composing VBBs, the consistency rules between the different transformation ports of the participating VBBs have to be obeyed. From a syntactic perspective, currying means to bind the together sources and sinks of different VBBs. Such binding is possible, when the corresponding expected and provided values match both regarding the employed types. The connected VBBs thereby establish a set of consistency constraints that apply to the unbound variables, i.e., to the ones not 'curried away'. Figure 6.25, employing the graphical notation, displays an example of a hybrid VBB constituted from the VBBs introduced above. The variables underlying connected sources and sinks are set equivalent in respect to their value. Let in the following A be an information source bound to a universal type variable u_{A}, and B be an information sink bound to a variable u_{B}. If A connects to B, then $u_{\mathsf{A}}.\mathsf{lower} \simeq u_{\mathsf{B}}.\mathsf{lower}$ has to hold. Thereby, u_{A} and u_{B} can be unified to a single variable u, which additionally inherits the lower bound of both original variables. This means that $u.\mathsf{lower} <: u_{\mathsf{A}}.\mathsf{lower}$ and $u.\mathsf{lower} <: u_{\mathsf{A}}.\mathsf{lower}$ have to hold, if values for the lower bound were specified. Similar considerations apply for property variables and relationship variables.

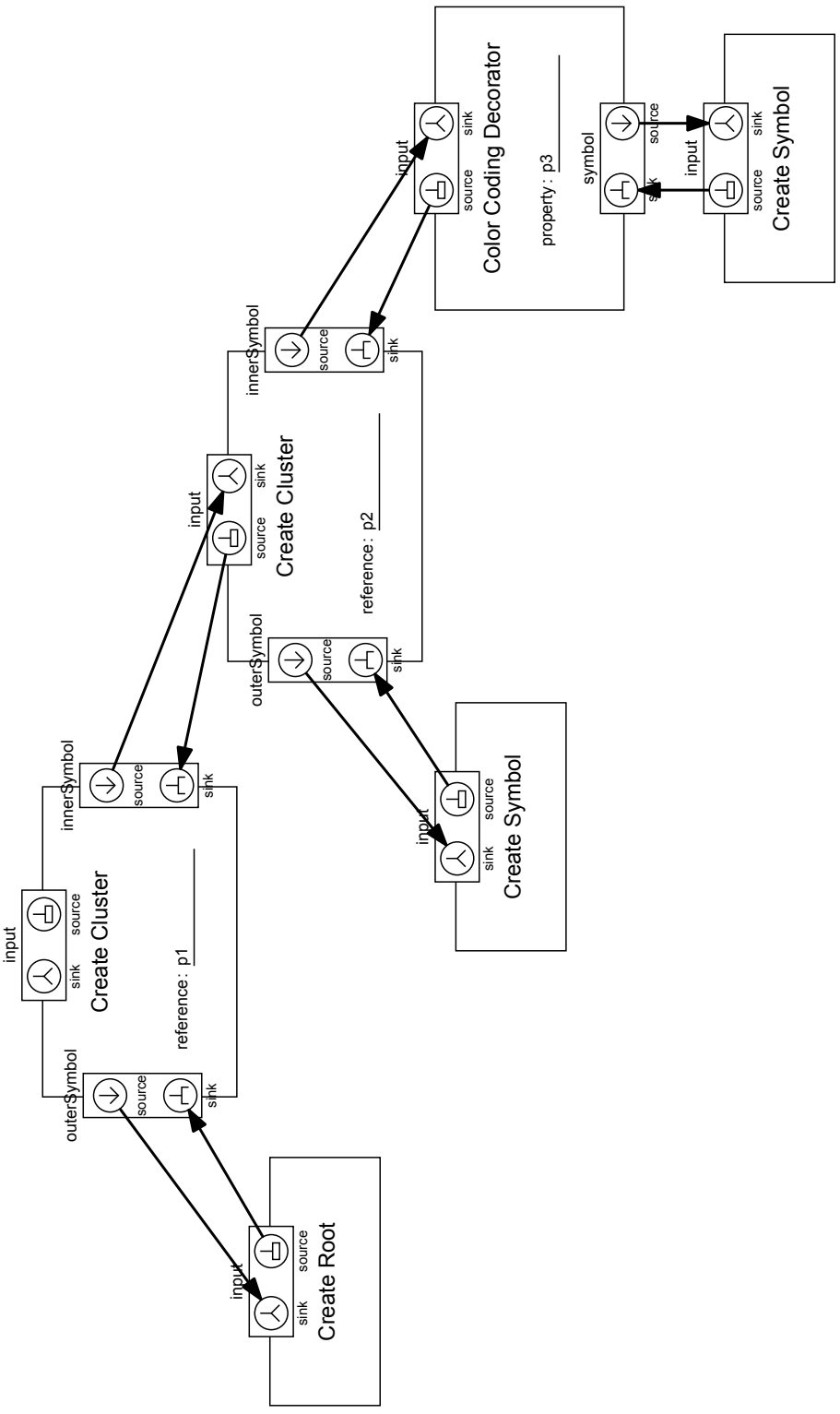


Figure 6.25: Exemplary hybrid VBB

The hybrid VBB from Figure 6.25 can be written as function that is composed of the functions provided by the elementary VBBs. The function of the hybrid VBB thereby retains several parameters p_1, p_2, p_3 and υ_1, υ_2, which have not been curried. To clearly designate the VBB instance for our subsequent discussions, we assign letters to parts of the specification:

$$\underbrace{\mathsf{cc}}_{\mathsf{a}}(\underbrace{\mathsf{cr}}_{\mathsf{b}}, p_1, \underbrace{\mathsf{cc}}_{\mathsf{c}}(\underbrace{\mathsf{cp}}_{\mathsf{d}}(\upsilon_1), p_2, \underbrace{\mathsf{dcc}}_{\mathsf{e}}(p_3, \underbrace{\mathsf{cp}}_{\mathsf{f}}(\upsilon_2)))).$$

Based on the constraints supplied by the elementary VBBs, we can derive the set of constraints that applies to the variables and parameters in the composed function. For υ_1 and υ_2 these constraints are formulated as $\upsilon_1 \in \Upsilon_{\text{planar}}$ and $\upsilon_2 \in \Upsilon_{\text{planar}}$. In the following, the constraints applying to p_1, p_2, and p_3 are derived stepwise. To clearly designate the variables employed in the different functions, we refrain the numbering of types as introduced in the signature of each elementary VBB. The number is further prefixed with a single letter, denoting which function is meant. In these terms, we can write the following bindinds:

1. b to the first parameter of a: we establish the condition $\gamma \simeq \mathsf{b}.u_1 \simeq \mathsf{a}.u_1$.
2. c to the second parameter of a: the following condition is established $\mathsf{a}.u_2 \simeq \mathsf{c}.u_1$.
3. p_1 to the third parameter of a: we establish $\mathsf{a}.u_1 \simeq dom(p_1)$ and $ran(p_1) \simeq \mathsf{a}.u_2$.
4. d to the first parameter of c: we obtain $\mathsf{c}.u_1 \simeq \mathsf{d}.u_1$.
5. e to the second parameter of c: the following condition is established $\mathsf{c}.u_2 \simeq \mathsf{e}.u_1$.
6. p_2 to the third parameter of c: we establish the condition $\mathsf{c}.u_1 \simeq dom(p_2)$ and $ran(p_2) \simeq \mathsf{c}.u_2$.
7. p_3 to the first parameter of e: the following condition is established $\mathsf{e}.u_1 \simeq dom(p_3)$.
8. f to the second parameter of e: we obtain $\mathsf{e}.u_2 \simeq \mathsf{f}.u_2$.

From (1) we derive that $\mathsf{a}.u_1 = \gamma$, which via (3) yields that $dom(p_1) = \gamma$. From (2) and (3) we reason that $ran(p_1) \simeq \mathsf{c}.u_1$ which together with (6) evaluates to $ran(p_1) \simeq dom(p_2)$. In a similar way we can use (6) and (7) to derive that $ran(p_2) \simeq dom(p_3)$. The range of p_1 is a universal type, which participates in a relationship described via p_2 to a universal type, which supplies an arbitrary typed property p_3. From the lower bounds specified in the different VBBs, we derive further constraints for the parameters. VBB instance D of type cp demands that coded.u_1 carries a nominating property. A similar constraint derives via VBB instance f for codef.u_1. VBB instance e of type dcc demands that an additional property for coloring exists in codee.u_1. From there, the viewmodel corresponding to the hybrid VBB can be derived. This model is shown in Figure 6.26, whereby no actual names are assigned to the model elements, but the names of the corresponding variables are provided.

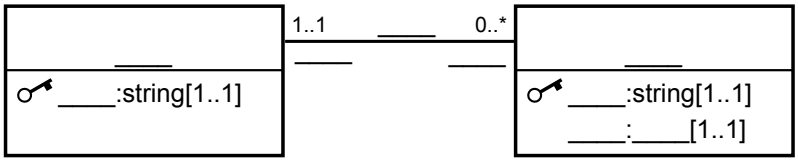


Figure 6.26: Viewmodel of the exemplary hybrid VBB

6.4 Meta-model for EA glossaries

According to our considerations from Section 3.1.2 a predicator serves as textual representation for a model element from an EA information model. Any such element is of its nature intensional, which entails that in order to convey the meaning of the model element or the predicator, respectively, a description therefore is needed. A **glossary** underlying an EA modeling language is a set of **predicators**, each supplied with a consistent textual description providing a definition and an explanation. Both the predicator and the description are represented textually. In this respect a glossary is used to supply the semantics for an EA modeling language via *text-based semantics definitions*, as discussed in Section 4.3.1. As it does so, a glossary can only be sensibly interpreted and used by a member of the corresponding **linguistic community**. The description of a predicator can contain a textual reference to one or more different predicators. These references have to be taken into account, when it comes to rules for understanding what makes a glossary consistent or not. Given one glossary this means that all predicators must not be described using references to predicators contained in a different glossary. Additionally, a glossary can be structured into **namespaces** which act as providers for the identity conditions on predicators. This means that with respect to the owning namespace, a predicator must be unique. This further yields the notion of the **fully qualified predicator**, which is derived from the current predicator and the fully qualified designator of the surrounding namespace. A glossary in turn does not supply a namespace designator. Figure 6.27 depicts the basic interplay between namespaces, glossaries, predicators, and textual descriptions. For the two derived characteristics employed in the interplay of

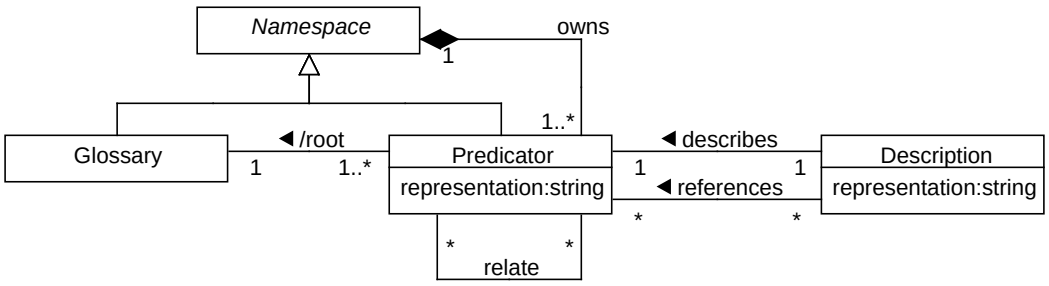


Figure 6.27: Basic structure of a glossary

namespaces and predicators, i.e., for the ROOT-relationship and the FQDESIGNATOR-property, the derivation rules are established. Expressed using the OCL [Ob10b], these rules read as:

```
context: Predicator
derive: root = owner.oclIsKindOf(Glossary)?owner:owner.root
derive: fqDesignator = owner.oclIsKindOf(Glossary)?
          representation:
          owner.fqDesignator + "." + representation
```

Based on these derivation rules, we can further operationalize consistency and uniqueness constraints. Particular interesting is the second invariant for the DESCRIPTION, which demands that the textual representation can be split into different fragments with the number of fragments exceeding the number of referenced predicators by exactly one. This is regarded a prerequisite of compiling the actual description text, as a subsequent derivation shows. The consistency and uniqueness constraints formally read as follows:

```
context: Predicator
inv: owner->collect(p.representation | owns)->count(representation)==1

context: Description
inv: references->forAll(p.root==describes.root)
inv: representation->count()==references->count() + 1
```

The explicit references between descriptions and the predicators used in describing make it possible to ensure consistent usage of textual representations throughout the glossary. For referencing a predicator as part of the complete textual description, a representation of the referenced predicator is used. If both the described and the referenced predicator reside in the same namespace of one acts as namespace for the other, the simple textual representation is used. In other cases, the fully qualified predicator is used. Formally, this derivation rule can be described using helper functions as:

```
context: Description
define: prHelper(p:Predicator) = (p.owner==owner||p.owner==self||owner==p)?
          p.representation:
          p.fqDesignator
define: repHelper(i:int) = (i->mod(2)==0)?
          representation->at(i->div(2)):
          prHelper(references->at(i->div(2)))
define: fullRepresentation(i:int) = (i<2*representation->count())?
          repHelper(i)->concat(fullRepresentation(i+1)):
          ""
derive: fullRepresentation = fullRepresentation(0)
```

One type of relationships between predicators belonging to different glossaries exists. This type is used to indicate, that the related predicators have a nearly equivalent meaning. Such relationships are in particular used to link an organization-specific glossary against the background of the EA management approaches from which the corresponding terms have been borrowed or derived.

A glossary building block of BEAMS is a small-scale glossary, i.e., a very limited namespace, only supplying one distinct predicator and its owned sub-predicators. With respect to the fact that the predicators contained in different GBBs not necessarily are consistent with each other, two additional types of relationships between predicators are introduced, namely SYNONYM and HOMONYM. While former relationship indicates that two predicators supply the same understanding of semantics, later relationship designates that two predicators have a different meaning, albeit they supply the same textual representation. Figure 6.28 introduces

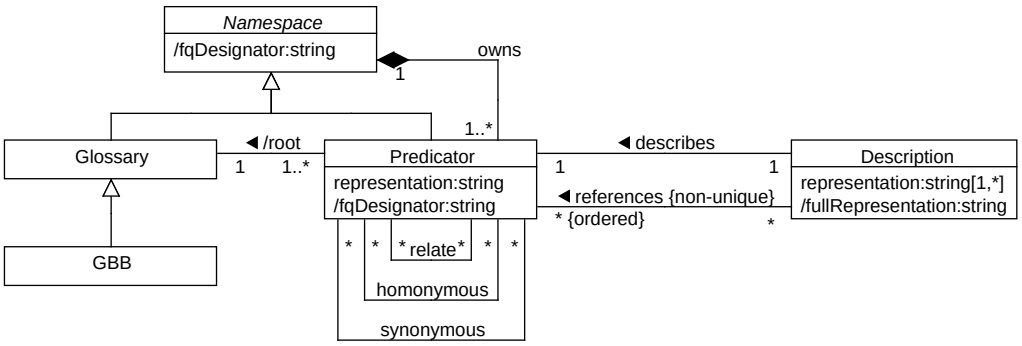


Figure 6.28: Conceptual model relating glossaries and GBBs

the GBB to the basic conceptual model relating glossaries and predicators. The two additional relationships provide a refinement to the notion of being **equivalently predicated** as introduced in Section 6.2. In the case of a single glossary, two model elements are called **equivalently predicated**, if they relate to the same predicator. If GBBs provide the predicators' namespaces, two model elements are equivalently predicated, if they relate to the same predicator or to two synonymous predicators. With respect to the additional relationships between predicators, the following constraints hold:

```
context: Predicator
inv: synonym->forAll(p.root<>root and p.representation<>representation | p)
inv: homonym->forAll(p.root<>root and p.representation==representation | p)
```

Building on these relationships, we can establish a necessary condition for different GBBs supporting the integration into a consistent namespace or glossary, respectively. A consistent namespace must own only consistent namespaces and must not contain two or more predicators, which are mutually homonymous. With this background the consistent integration of GBBs poses a condition to which we have to adhere, when we integrate IBBs into a comprehensive information model. In the analyses of consistency we additionally have to account for the fact that being homonymous is *transitive*[4], i.e., that for any three predicators p_1, p_2, and p_3 the following holds:

$$p_1 \text{ homonym } p_2 \wedge p_2 \text{ homonym } p_3 \Rightarrow p_1 \text{ homonym } p_3.$$

The relationships between GBBs are also used, when a glossary is adapted to the specific terminology of a using linguistic community, e.g. an organization. Using GBBs, we seek to support the organization in establishing a specific terminology, while at the same time the EA modeling language must support being extended by further building blocks. Therefore, especially the **predicated subsume**-relationship introduced in Section 6.2.2 is important. With this relationship building on the **equivalently predicated**-relationship, it is important

[4] Actually, being homonymous is an equivalence relationship, i.e., a *reflexive*, *symmetric*, and *transitive* relationship. This nevertheless is of minor importance here.

that during the adaptations we keep track of the applied **renamings**, i.e., of the changes in the representations of the predicators. This is achieved by initially duplicating the predicator, renaming the duplicate, and establishing a relationship indicating that both predicators are homonymous. Making use of the following extension of the principle of being **equivalently predicated**, the predicate subsume-relationship remains consistent over renaming operations. Two model elements from one or more information models are equivalently predicated, if they share the same predicator or relate to predicators that are homonymous.

6.5 Summary

In this chapter we described the *building blocks for EA management solutions* (BEAMS). BEAMS is a method for developing organization-specific EA management functions. Central to the descriptions in this chapter is the 'language part' of BEAMS, i.e., the building blocks and methods used to develop organization-specific EA modeling languages. Laying the groundwork for BEAMS we develop and explain a language framework that shows how two dedicated meta-languages are employed for describing information models and viewpoints, or corresponding IBBs and VBBs, respectively. In Section 6.1 we stepwise develop the framework, extending the ideas of Buckl et al. [BKS10] towards a comprehensive conceptualization for the domain for EA modeling. Key element in the framework is a formal understanding of information models and their relationships to each other via **embedding**-relationships. These relationship were further decomposed to elementary relationships between the MODEL ELEMENTS of information models. Subsequent Section 6.2 furthers the idea and introduces the IM2L as well as its constituting model elements. Preparing this, we briefly revisit general purpose meta-languages as the UML [Ob10c], ORM [Ha09], or the MML [Fr09], and show which characteristics of EA information modeling are not well reflected in these languages. In particular, we show that **non-rigid typing**, **temporality**, and **specialization by constraint** are not supported well. Responding to these shortcomings, we develop a specialized meta-language for EA information modeling—the IM2L—whose model elements, especially their syntax and semantics, is discussed. Linking back to the concept of embedding, we formally derive rules and conditions for **subsume**-relationships between model elements and refine the general notion of subsume with respect to the usage context. Following Table 6.6 displays the different types of subsume-relationships introduced in Sections 6.2 and 6.3, respectively.

	read & write subsume (**strict**)	read subsume (**weak**)
only structure	strict structural subsume $<:$	weak structural subsume $\sqsubset:$
structure and predicator	updating subsume $<::$	reading subsume $\sqsubset::$

Table 6.6: Different kinds of subsume-relationships

A graphical notation for the modeling elements of EA information modeling concludes the exposition of the IM2L. This notation resembles the one of the UML, which is prevalent in the context of EA information modeling. Proceeding from the IBBs to comprehensive information models, we introduce different operators that can be used to combine IBBs as well as to refactor information models. Later particularly accounts for the fact that different alternatives for representing the same domain qualities can exist, with the refactorings allowing to transform between such representations. Subsequent Section 6.3 revisits the conceptions

notation function as well as **representation function**, which are rewritten in formal terms building on the **visualization model** presented by us in [Er06b]. Building blocks for defining such functions are subsequently presented and a meta-language for describing such VBBs is delineated. This language in particular facilitates the composition of more complex VBBs and ultimatively of viewpoints from other VBBs. From a semantic perspective, this composition is identified with the operation of currying of functions, especially using other functions. Building on this understanding, consistency rules are derived and a technique for deriving the **viewmodel** of a hybrid, i.e., composite, VBB is introduced. Building on this technique and the **updating subsume**-relationship and **reading subsume**-relationship, we delineate how to determine whether a VBB can notate or represent an underlying information model. Completing the exposition of building blocks, we delineate how GBBs can be integrated and adapted to build an organization-specific glossary for the EA modeling language. Therein, we further describe how the GBBs are used to root BEAMS in the EA management approaches, from which the relevant prescriptions and best-practices have been drawn.

We study the mysteries of laser and circuit, crystal and scanner, holographic demons and invocation of equations. These are tools we employ, and we know many things.
» Such as?

Dialog of Elric and John Sheridan, The Geometry of Shadows—Babylon 5

CHAPTER 7

Implementation Aspects of BEAMS

In this chapter we discuss the role of tool-support for BEAMS. Therein, we have to distinguish two different aspects of tool-support: firstly, the support for the building block-based development method, and secondly, the support for the implementation of the configured EA modeling languages. Figure 7.1 gives an overview of the two aspects of tool-support, their interplay, and their stakeholders.

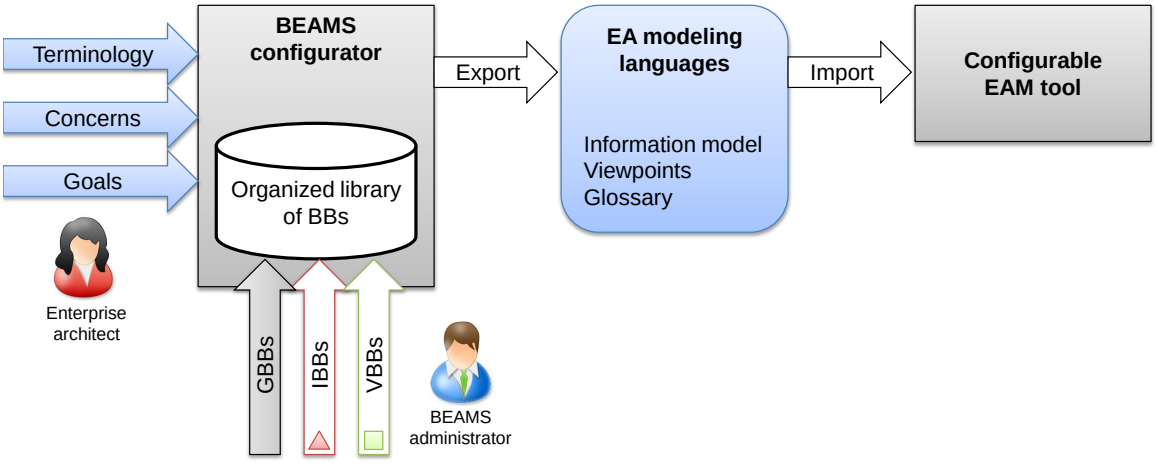


Figure 7.1: Overview about the tool-support for BEAMS

Along the two aspects of support, we distinguish two specific tools:

- The **BEAMS configurator** is used by the enterprise architects to develop a set of organization-specific EA modeling languages. These languages address the EA management-related problems that the stakeholders of the EA management function rise.

- The **configurable EAM tool** implements the modeling languages and is used by the stakeholders as well as other participants to execute the EA management function. The tool in particular supports the graphical modeling as well as representation of EA information.

The two tools are subject to strongly different requirements. While the configurator provides decision support for a small but highly skilled user group, the EA management tool has to provide collaborative access for EA management stakeholders from diverse educational backgrounds. Therefore, we discuss both tools separately, starting with the BEAMS configurator in Section 7.1. We describe the use cases for such configurator and derive a feasible design. The implementation of the design on a web-based wiki-platform *tricia* [In10] and a web-based modeling platform *ORYX* [DOW08a, DOW08b] is further described. Regarding the configurable EA management tool, we delineate in Section 7.2 how characteristics of the building block-based perspective to EA management pertain to requirements for the tool. We discuss to which extent the relevant aspects of a) the language for describing IBBs from Section 6.2, b) the language for describing VBBs from Section 6.3, and c) the techniques for supplying GBB-based glossaries are supported in today's EA management tools and related platforms. Final Section 7.3 summarizes implementation aspects and the tool-support for BEAMS.

7.1 Implementation of the BEAMS configurator

Pries-Heje and Baskerville explain in [PHB08] that a nexus-based development method requires tool-support. A corresponding tool provides decision support for the users of the development method, especially for selecting the appropriate design theories based on the supplied values for the criteria. This decision support is **static**, i.e., the selection of one design theory does not affect the appropriateness of another theory. In the context of BEAMS this assumption does not hold, as we highlighted in Section 5.2. Two building blocks can be incompatible with respect to their prescriptions, such that an admissible design cannot contain both of them. Therefore, the BEAMS tool-set has to provide **dynamic decision support** that adapts to the current organization-specific configuration.

7.1.1 Use cases

The use cases of the BEAMS configurator reflect elementary actions from the development method as outlined in Section 5.3. Central to the development method is the **organization-specific configuration** that determines the outcome of several of the steps taken. Initially, the development method starts with an empty organization-specific configuration, which is extended in a stepwise manner. In order to reflect the interplay between the use cases and the current organizational configuration, we use the UML concept of the *extension point* [Ob10d, pages 609–610] to designate that given a non-empty configuration a use case is extended with additional activities. Two kinds of use cases, namely ones for **browsing the organized library** and ones for **searching the organized library** are provided to the different types of users of the BEAMS configurator. Figure 7.2 gives a corresponding use case diagram.

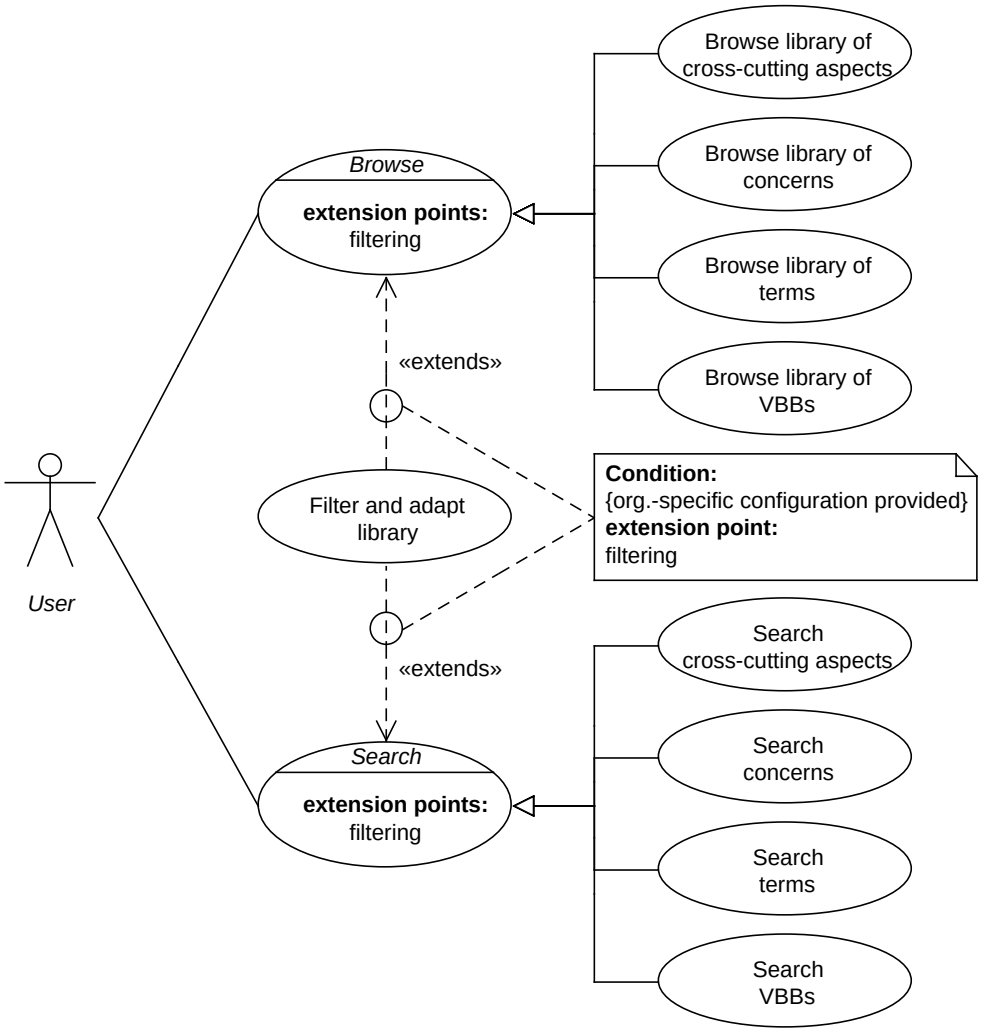


Figure 7.2: BEAMS configurator: general use cases

In the following we give a brief textual description for the use cases and their specializations:

UC1 *Browse* The users can browse through the different parts of the organized library of BEAMS. If the users have already supplied a non-empty organization-specific configuration, the library elements are filtered for admissibility. The following specializations of browsing are available:

- ***UC1a*** *Browse library of cross-cutting aspects* The users can access the list of (admissible) cross-cutting aspects ordered by the (adapted) name of the aspect or the corresponding cross-cutting dimension.
- ***UC1b*** *Browse library of concerns* The users can access the list of (admissible) concerns ordered by the (adapted) name or by the corresponding architectural layer.
- ***UC1c*** *Browse library of terms* The users can access the (adapted) list of terms ordered by name or by the corresponding architectural layer and cross-cutting dimension.

UC1d *Browse library of VBBs* The users can access the (adapted) list of (admissible) VBBs ordered by the name of the VBB.

UC2 *Search* The users can search elements in the organized library of BEAMS by name or by full-text content. In case the users already specified a non-empty organization-specific configuration, the search results are filtered for admissibility. The following specializations of searching are available:

UC2a *Search cross-cutting aspect* The users can find (admissible) cross-cutting aspects based on the name, a textual element in the description or in the complementing IBB, or a particular term.

UC2b *Search concern* The users can find (admissible) concerns based on the name, a textual element in the description or in the complementing IBB, or a particular term.

UC2c *Search term* The users can find (adapted) terms based on textual elements in the description.

UC2d *Search VBB* The users can find (admissible) VBBs based on the name, or a textual element in the description.

UC3 *Filter and adapt library* The BEAMS configurator filters the organized library to the elements that remain admissible in a given context. This means that any area-of-interest expressed in an IBB that conflicts with a single IBB in the organization-specific configuration is filtered out. For the VBBs similar filtering to the usage context (notation vs. representation function) applies. Further, the terminology is adapted to the one in the organization-specific configuration. This means that for any predicator a corresponding synonym from the organization-specific glossary is used.

The development method and its constituting activities are supported by corresponding use cases of the BEAMS configurator. These use cases can be invoked by the **enterprise architects**, who develop the organization-specific EA modeling languages. During development the current organization-specific configuration plays an important role. This configuration is used according to use case ***UC3*** *Filter and adapt library* to filter not admissible concepts and adapt their descriptions to the organization-specific terminology. Figure 7.3 displays the enterprise architect's use cases in the development method, of which we further give a brief textual description in the following:

UC4 *Rename predicator* The enterprise architects change the glossary term that is used to predicate an EA-relevant concept. The adapted term becomes part of the organization-specific configuration and retains a linking SYNONYM-relationship to the original term from the organized library.

UC5 *Specify problem* The enterprise architects specify an EA management-related problem raised by one of the stakeholders of the EA management function. Any problem specification is composed of at least one goal and one concern and optionally encompass one or more cross-cutting aspects. This lists of areas-of-interest are filtered, in case of incompatibilities with the organization-specific configuration. The following use cases are sub-activities of the problem specification:

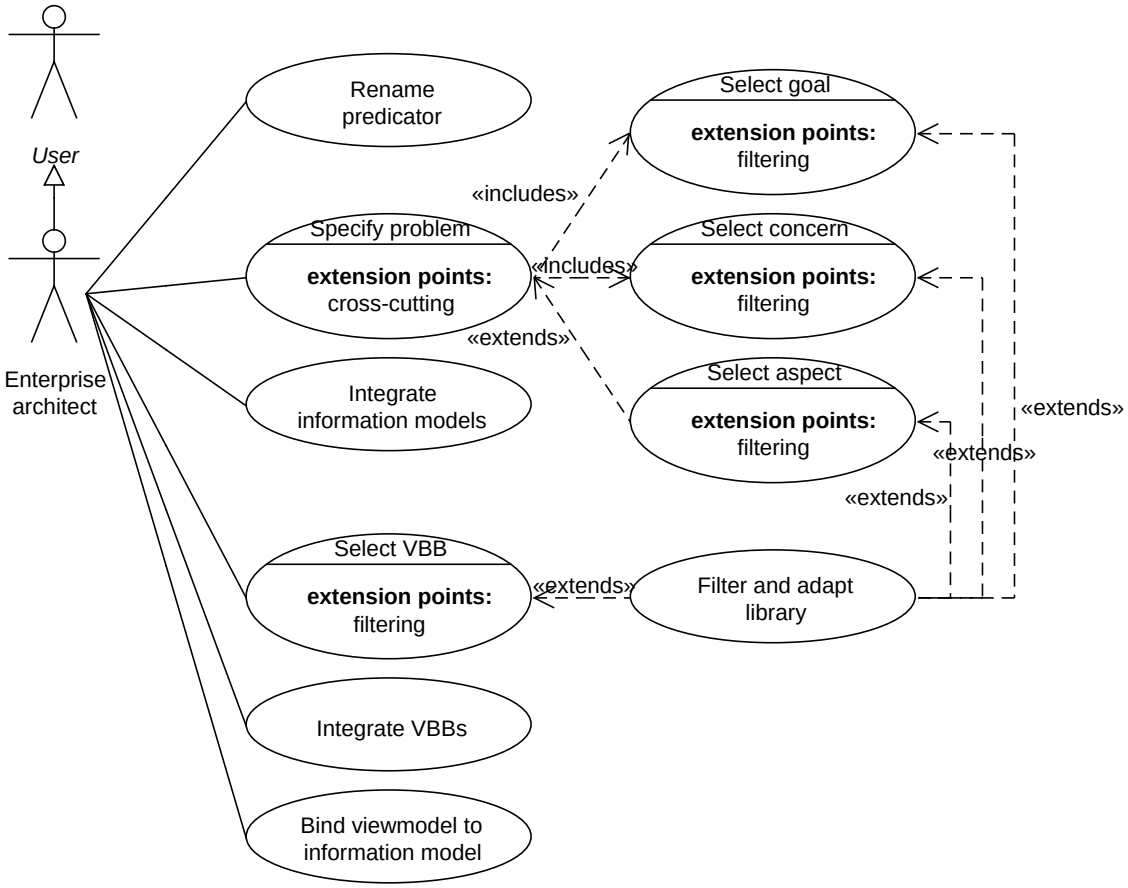


Figure 7.3: BEAMS configurator: use cases of the development method

UC5a *Select goal* The enterprise architects choose one goal from the list of admissible goals, and select a particular operationalization thereof. Thereby, the corresponding IBB is selected.

UC5b *Select concern* The enterprise architects choose a concern from the list of admissible concerns. Thereby, the corresponding IBB is selected.

UC5c *Select aspect* The enterprise architects select one or more cross-cutting aspects from the list of admissible aspects. Thereby, the corresponding IBB is selected.

UC6 *Integrate information models* The enterprise architects integrate the IBB reflecting goal, concern, and optional cross-cutting aspects. Especially, the architects have to specify, which architectural element is subject to the selected goal. The resulting information model is subsequently integrated into the information model of the organization-specific configuration. The BEAMS configurator performs an automatic integration for all concepts, where aggregation by composition or by generalization can be applied (cf. Section 6.2.3).

UC7 *Select VBB* The enterprise architects iteratively select VBBs that contribute to a current viewpoint. This use case is invoked intermittently with the use case ***UC8***, such that any new VBB is integrated into the viewpoint.

UC8 *Integrate VBBs* The enterprise architects integrate the VBBs to a comprehensive viewpoint description. The BEAMS configurator therein ensures syntactic consistency of the integrated viewpoint, which in turn is described as **hybrid VBB**.

UC9 *Bind viewmodel to information model* The enterprise architects assign information model elements to the corresponding parameters of the viewpoint specifying the viewmodel. This viewmodel is iteratively updated, whenever an information model element is bound. The BEAMS configurator additionally provides configuration support by recommending further model elements to be bound.

Above use cases ***UC4–UC9*** are also used in evolving a building block-based EA modeling language. Different use cases are required to support the administration method, in which the **BEAMS administrators** add new language building blocks to the organized library or update existing ones. Central to the administration method are use case ***UC2*** *Search* and its specializations. Thereby, the administrators inform themselves on the already supported EA concepts in order to avoid the introduction of duplicates and synonyms. The search further extends the administration-related use cases outlined below by providing *auto-completion* for names and predicators. Figure 7.4 describes the use cases of the BEAMS administrators, which are subsequently detailed textually.

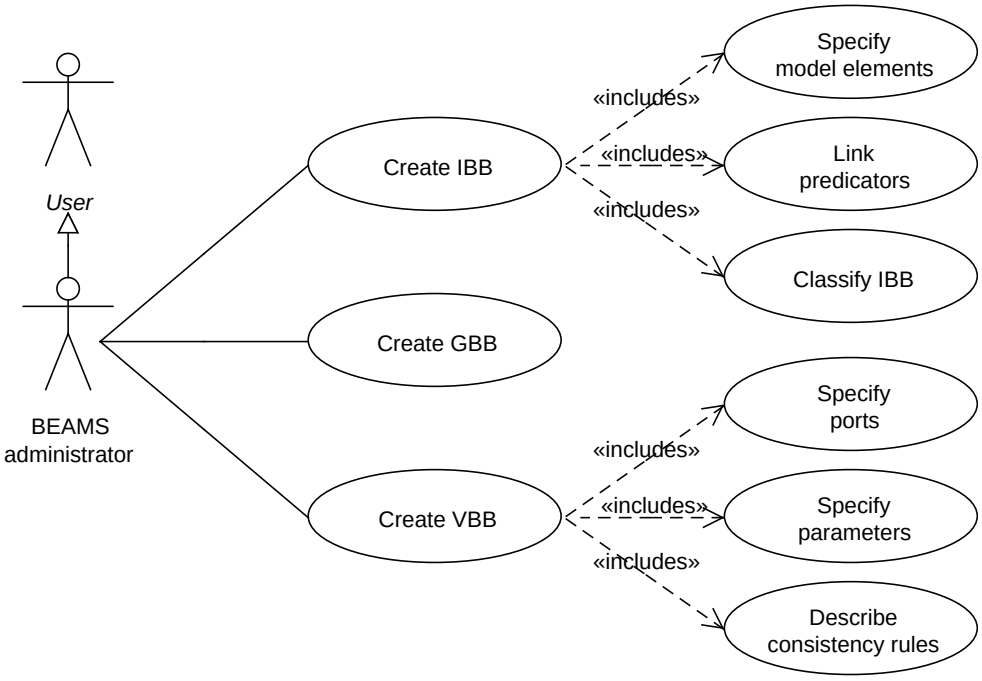


Figure 7.4: BEAMS configurator: use cases of the administration method

UC10 *Create IBB* The BEAMS administrators introduce a new IBB into the organized library of building blocks. Thereto, the following sub-activities have to be performed:

UC10a *Specify model elements* The BEAMS administrators describe the model elements using the graphical notation for IBBs as described in Section 6.2.4.

UC10b *Link predicators* The BEAMS administrators relate the model elements in the IBB to existing or newly defined predicators in the organized library of GBBs.

UC10c *Classify IBB* The BEAMS administrators specify the quality of the **area-of-interest** represented by the IBB. They specify, whether the IBB represents a goal, a cross-cutting aspect, or a concern.

UC11 *Create GBB* The BEAMS administrators introduce a new GBB into the organized library of building blocks and provide a textual description for the concept. Thereby, links to related GBBs are established. The configurator ensures that the new GBB has a unique predicator.

UC12 *Create VBB* The BEAMS administrators introduce a new VBB into the organized library of building blocks and provide an exemplary visualization for the VBB. The administrators further supply a description of the VBB's operational semantics, either textual or using a formal language. The following sub-activities are also employed in creating a new VBB:

UC12a *Specify ports* The BEAMS administrators specify which input and output ports are supplied by the VBB, assigning the ports to names that are unique in the VBB. Any port is further described in terms of its range and domain values.

UC12b *Specify parameters* The BEAMS administrators specify which parameters influence the operational semantics of the VBB. Each parameter is further described in terms of its corresponding value range.

UC12c *Describe consistency rules* The BEAMS administrators describe how the range and domain values of the different ports as well as the value ranges of the parameters of the VBB relate. The administrators identify the different sets of admissible or expected values with each others.

7.1.2 Design

The **BEAMS configurator** is a tool, which provides two distinct operating modes: development and administration. In former mode, the configurator provides decision support for the enterprise architects in developing EA modeling languages. In the latter mode, the configurator works as a knowledge management system to store, retrieve, analyze, and communicate information about best-practice building blocks. Both operating modes build on the same information base—the organized library of building blocks. The hybrid nature of the description consisting both of structured and unstructured content formulates a design constraint for the BEAMS configurator. Against this background and in the light of the twofold nature encompassing knowledge management and decision support aspects, we revisit the functionalities provided by enterprise 2.0 platforms. Büchner et al. describe in [BMN09] the typical services provided by an enterprise 2.0 platform, distinguishing between two major groups of

services, namely *content-centric* and *user-centric* ones. In the following, we briefly introduce the services that are useful for realizing the BEAMS configurator. In addition, we relate the services to the use cases from Section 7.1.1 that they can support:

Authoring is the collaborative and web-based creation and manipulation of content. The content can be textual or structured and can contain tables, images, and other media objects. Further, functionalities to export content are regarded critical.

Link management contains services to reference content of different types. Two key functionalities of link management are automatic propagation of link updates and highlighting of links that are not longer valid.

Tagging allows the users to create a categorization for content of different types. Specific services in this context support to establish private tags, which are only visible to their owners.

Search supports to find contents based on tags or names. Sophisticated search mechanisms allow full-text search and combined searches for different criteria.

The generic services of enterprise 2.0 platforms provide a good starting point for realizing the BEAMS configurator. Nevertheless, the modeling-related use cases as well as the use cases demanding algorithmic functionality for integrating building blocks or ensuring consistency are not natively supported. The same is true for the filtering use cases that build on the organization-specific configuration. Web-based modeling platforms can be used to support the modeling-related use cases, if the platforms can be adapted to the syntax of the meta-languages described in Section 6.2 and 6.3. This motivates a design for the BEAMS configurator, where we integrate an enterprise 2.0 platform with a web-based modeling platform and realize missing algorithmic functionalities by self-developed extensions. Mirrored against the relevant services, this general design entails the following adaptations:

Authoring The authoring service has to be extended to support modeling activities. Therefore, a WISYWIG-editor service supporting both the meta-language for specifying information models and the meta-language for defining viewpoints has to be integrated. This editor incorporates the consistency rules for viewpoint configuration as supplied in Section 6.3.

Link management The information models as well as the viewpoint definitions extend the content model of the platform. Additional functionality for linking between these models or represented model elements to textual descriptions and vice-versa are required. Link-checking and link-propagation mechanisms have to be extended to also operate on the model content.

Tagging Information models and viewpoint definitions are treated as content objects and must be taggable.

Search The contents of the information models as well as of the viewpoint definitions must be accessible via the unified search mechanisms of the platform. This in particular means that models can occur as search results, if a corresponding model element supplies a searched predicator or term. The search is further extended to incorporate the consistency rules supplied in Section 6.2.

An additional service for automatically integrating information models has to be provided by the BEAMS configurator. This service operates on a selection of information models, that can be **consistently aggregated**, and creates an integrated information model, which is handed over to model authoring.

7.1.3 Implementation

Our prototypic **BEAMS configurator** is based on the open-source enterprise 2.0 framework Tricia [In10]. This framework provides a platform core providing the basic abstractions for enterprise 2.0 tools, as user management, including registrations, login, and group creation. Building on the basic abstractions, a plugin-based mechanism can be used to extend Tricia with additional functionality. Tricia out-of-the-box delivers a set of plugins that provide functionalities targeting different content types [BMN09]:

wiki plugin introduces support for the content type *wiki page* and allows authoring, tagging, searching, and link management regarding wikis and wiki pages [BMN10].

hybrid wiki plugin introduces the content type of the semi-structured *hybrid template* that allows to supply key-value pairs in-line in a wiki page.

Graphical modeling is not supported by Tricia, but is of central importance to our configurator. Therefore, we decided to incorporate the open-source web-based modeling framework Oryx[1] as plugin to the Tricia framework [Di10]. The resulting *model editor* plugin enables authoring, tagging, searching, and link management regarding the thereby introduced content type *diagram*. Oryx provides the "stencil technology" [DOW08a] that allows to define different *stencil sets*, which represent the notation and the syntax of a modeling language. We implemented the syntactic rules and the notational aspects for modeling information models, see Section 6.2, and for describing viewpoints, see Section 6.3, in two stencil sets for Oryx.

The consistency rules used during searching and configuring as well as the aggregation techniques for information models can be supported in a dedicated *BEAMS techniques* plugin. The plugin operates on the graphical representations of the corresponding models and can check, whether two information models are compatible, i.e., can be consistently integrated. The results from this analysis are incorporated into the result of searches and used to rule out models, that are not admissible. The aggregation techniques are applied on information models and integrate the model elements to the point, where consistent embedding can unambiguously be performed.

In the following, we describe how the organized library of building blocks is documented using the enterprise 2.0 services provided by Tricia, Oryx, and our extensions. The library is represented by the BEAMS wiki[2], in which each building block is documented by a single wiki page. Depending on the type of building block, the corresponding page is *tagged* with "ibb", "vbb", or "gbb", respectively. Furthermore, the pages representing IBBs and VBBs are equipped with *hybrid templates* that describe the characteristics of the corresponding building block in a structured manner. For an IBB, the template references the coarse grained area-

[1] For more details on the Oryx project see `http://bpt.hpi.uni-potsdam.de/Oryx/`, last accessed 04-05-2011.
[2] The BEAMS wiki is available at `http://wwwmatthes.in.tum.de/wikis/beams/home`, last accessed 04-05-2011.

of-interest, to which the IBB commits, and lists the predicators that are represented in the IBB. In addition, the "references" section of the hybrid template designates which related EA management approaches employ the IBB and which IBBs are subsumed by the current IBB.

Figure 7.5 exemplifies a wiki page representing an IBB.

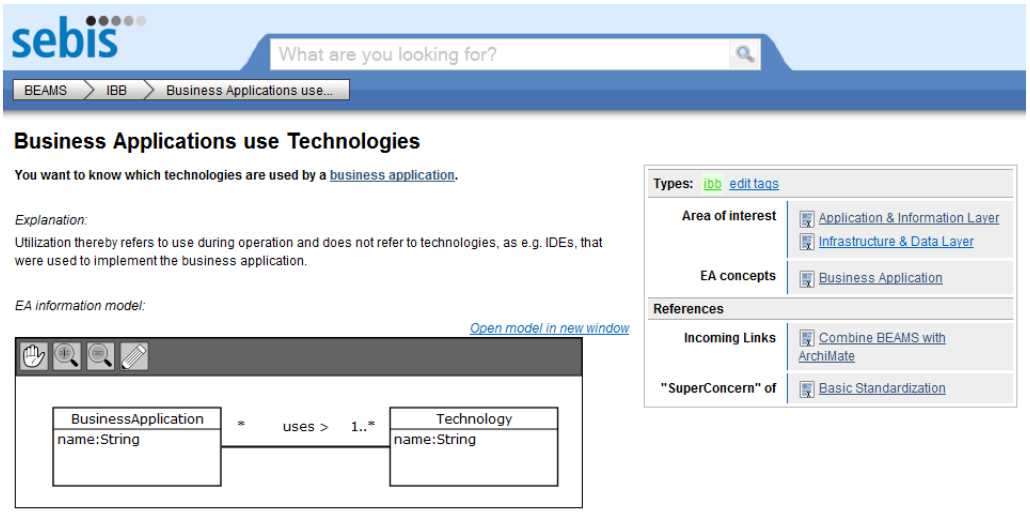


Figure 7.5: BEAMS configurator: wiki page representing an IBB

In a similar sense, we use wiki pages to document VBBs and designate them as **symbol**, **structural**, **decorating**, or **hybrid VBBs**. Due to the graphical nature of the VBBs, we use exemplary views corresponding to the viewpoint to illustrate the nature of the represented perspective. We further supply a graphical model of the building block designating the transformation ports and parameters. A final part of the describing wiki page designates the viewmodel corresponding to the VBB. Figure 7.6 exemplifies a wiki page representing a VBB.

The GBBs are represented in two different ways. Top-level GBBs, i.e., ones directly belonging to the BEAMS glossary, are represented as single wiki pages. The contained GBBs are described in line with the description of the enclosing GBB. Each used term is provided a short description on the corresponding pages and hyper-links are used to represent connections in the descriptions. Linking between the wiki pages representing the different types of building blocks allow to navigate between related concepts and to access related concepts in a convenient manner. We distinguish the two actor roles of the **BEAMS configurator** along their access privileges with respect to the wiki pages and the contained graphical representations. The **BEAMS administrators** can read and edit the wiki pages representing the building blocks, whereas enterprise architects are restricted to reading access and (personal) tagging.

The search mechanisms are used in different ways to support the use cases of the **BEAMS configurator**. We use the functionality to store searches to represent a particular perspective on the overall BEAMS wiki. For instance, one stored search identifies all IBBs that are of type **goal IBB**. These stored searches are embedded into wiki pages that organize the contents of the BEAMS wiki and facilitate user access to the different types of organized libraries. Figure 7.7 shows the central organization page for the IBBs.

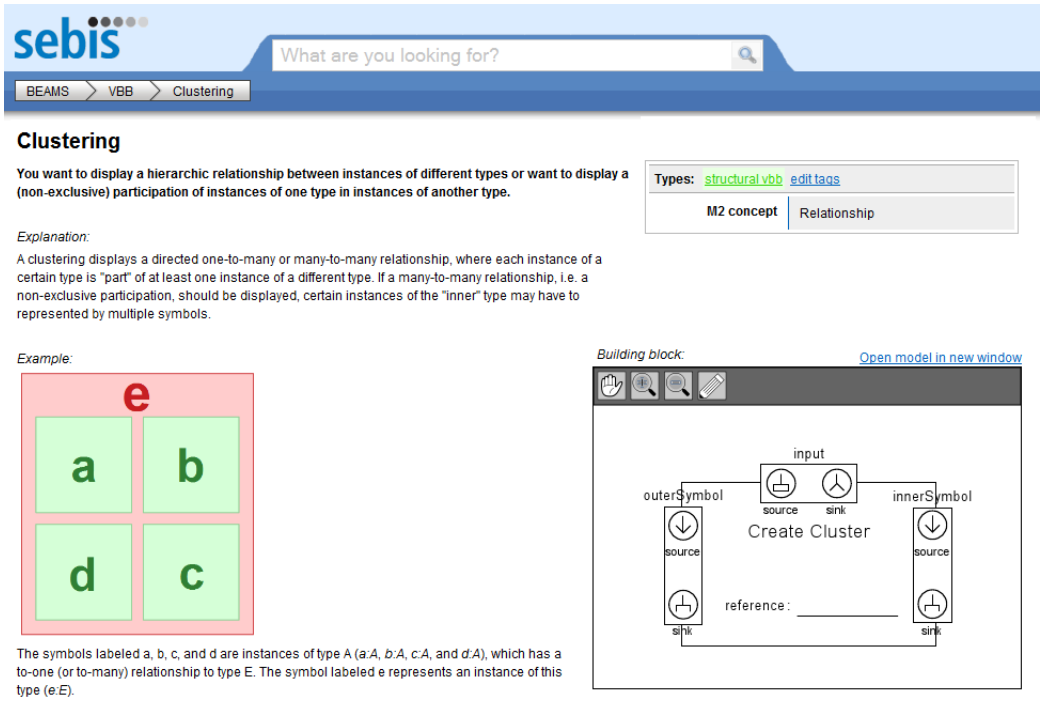


Figure 7.6: BEAMS configurator: wiki page representing a VBB

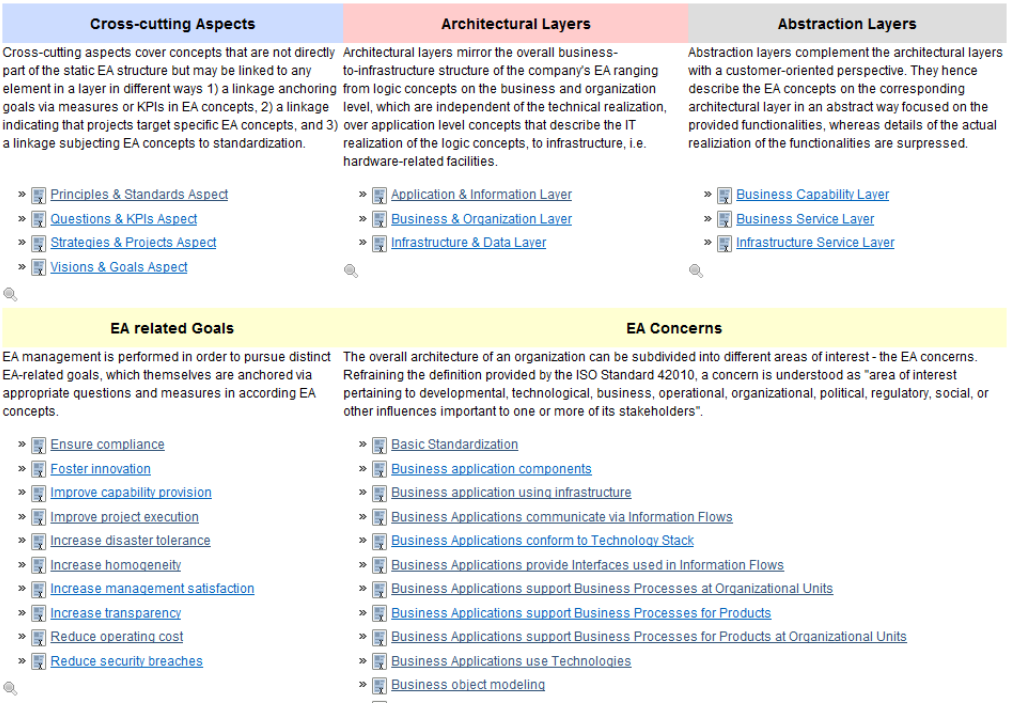


Figure 7.7: BEAMS configurator: browsing the organized library of IBBs

The search mechanisms of the configurator allow the enterprise architects to identify appropriate building blocks. The full-text search can be used to find building blocks containing

arbitrary terms, whereas the referencing between the different types of building blocks can be used to traverse from relevant GBBs to corresponding IBBs and vice versa. Advanced search options as logical conjunction between different search criteria allows the enterprise architects to narrow the search results. Highlighting of the searched keywords in the preview of a search result facilitates quick recognition and access of the relevant information. Figure 7.8 illustrates the search mechanisms for enterprise architects.

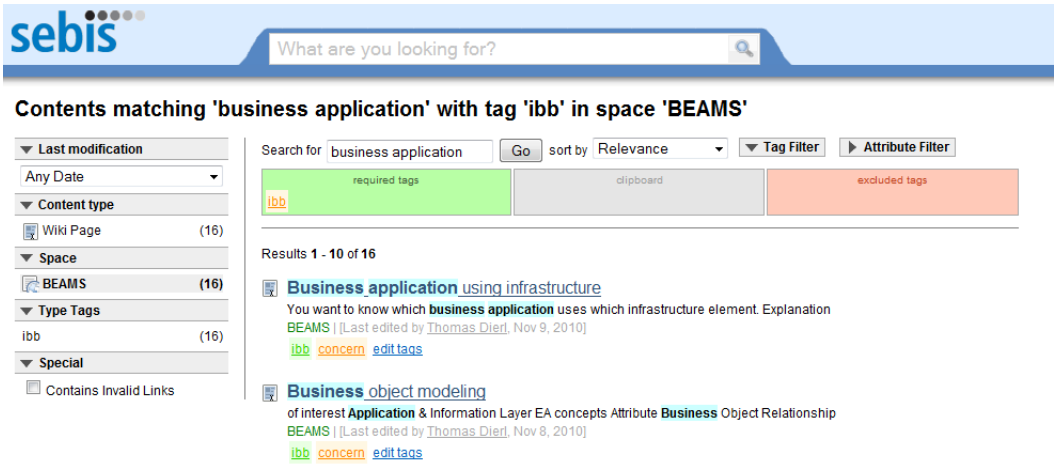


Figure 7.8: BEAMS configurator: searching the organized library of IBBs

The dedicated techniques for integrating information models are realized in a plugin that operates on two information models and performs consistent integration. The resulting configured information model (fragment) is inserted into the organization-specific wiki representing the architects' organization-specific configuration. In this wiki access rights are set differently, such that the enterprise architects can adapt the information model as well as related concepts. From the organization-specific wiki, also the export of the configured EA modeling languages is performed. Therein, the configured information model is represented as metamodel in terms of the *Eclipse Modeling Framework* (EMF)[3], i.e., as *.ecore*-file. The configured viewpoints are represented as *.json*[4]files that encode their composition and binding. This file format is chosen as no standardized format for describing viewpoints in a machine-readable way exists.

7.2 Characteristics for a configurable EA management tool

The building block-based development method for EA modeling languages creates language descriptions consisting of an organization-specific glossary, information model, and viewpoint definitions. These descriptions have to be implemented in an EA management tool to support developing EA modeling languages in the context of an EA management function. The specifics of the syntax, semantics, and notation of the languages developed with BEAMS raise certain requirements with respect to the EA management tool. In the following, we discuss these requirements and show to which extent current EA management tools provide the required functionalities.

[3]For more information see `http://www.eclipse.org/emf`, last accessed 04-05-2011.
[4]For more information see `http://tools.ietf.org/html/rfc4627`, last accessed 04-05-2011.

The elicitation of the required syntactic characteristics in Section 6.2 is the primary source of requirements for the information modeling capabilities of the tool. Especially, the characteristics that are not well-supported by the analyzed meta-languages are of particular interest for our analysis. An EA management tool suitable for supporting BEAMS-based EA modeling languages must support:

- domain-specific data-types DATE, TIME, and MONEY,
- **non-rigid** typing types,
- **bi-temporal** modeling for selected EA model elements,
- **specialization by constraint**, i.e., a mechanism to derive types by selection mechanisms, and
- external **predicators** that are linked to the model elements.

In [Bu08a, Ma08] we discuss that EA management tools can be distinguished along their fundamental approach towards EA management. We identified three dimensions along which EA management tools can be classified, namely "flexibility vs. guidance", "preconfigured vs. customization", and "integration vs. single-point-of-truth". Especially the first dimension is of interest to define the characteristics of an EA management tool that can be used to realize an EA modeling language developed using BEAMS. On this dimension we distinguish the following types of tools:

Meta-model driven tools allow to flexibly adapt the information model based on strong meta-modeling capabilities. These capabilities are often based on standardized meta-modeling facilities, as the *Meta Object Facility* (MOF) [Ob08]. Tools of this type usually provide only limited out-of-the-box visualization capabilities and require their users to specify notations and representations themselves.

Methodology driven tools supply comprehensive predefined information models together with predefined visualizations. Usually, only minor adaptations to the information model, e.g. by adding properties or renaming concepts, can be made. The same applies to the supported visualizations.

Process driven tools do not only predefine the information model and the corresponding visualizations, but also incorporate a predefined EA management process. This process designates roles and responsibilities of the different stakeholders of the EA management function, and prescribe the tasks that the stakeholders have to execute.

The flexibility of the information models, which can be developed using BEAMS, demands for an equal flexibility in the EA management tool. Due to their nature, only *meta-model driven* tools can be used to realize the different EA modeling languages developed with BEAMS. In the following, we briefly analyze typical proponents of this type of tool with respect to the provided fulfillment of the five aforementioned relevant characteristics. Thereby, we concentrate on tools that provide own meta-modeling facilities. Therefore, we do not discuss *adaptive EAM 5.0* as well as *MEGA Modeling Suite 2007*, as these tools provide comprehensive MOF support. In this vein, the analysis results in Table 6.1 applies to them. We also analyze the open-source wiki platform Tricia [In10], as wikis can according to our findings in [Bu09h, Bu11d] also be used to document EAs. The results of the analysis are based on the thesis of Balke [Ba10a]. In the following Table 7.1 we summarize the findings.

	ADOxx	Metastorm Provision 6.0	System Architect 11.0	Troux 7	Tricia
domain-specific data-types	●	●	●	●	○
non-rigid typing	○	○	○	○	●
bi-temporality support	●	○	○	○	○
specialization by constraint	○	○	○	○	◐[1]
external predicators	◐[2]	◐[2]	◐[2]	◐[2]	◐[2]

[1] Stored searches in Tricia can be used to constrain a type to only instances constrained to particular characteristics.
[2] The predicators assigned to the types can be changed on a global, i.e., repository-wide basis.

Table 7.1: Flexible EA management tools and their support for the relevant information model characteristics

The analysis shows that the different characteristics of BEAMS-based information models can partially be reflected in current EA management tools and a wiki-based approach. While this hints towards the implementability of EA information models resulting created by the development method, three characteristics deserve special attention. The 'typical' EA management tools do not support non-rigid typing, i.e., instances of architectural elements are confined to a particular type that cannot be changed over the existence of the instance. Wiki pages in Tricia can be dynamically re-typed by changing the tag assignment. Temporal modeling and bi-temporal modeling are only supported by one tool and remain a critical topic for comprehensive tool support for EA modeling. Specialization by constraint is not supported by 'typical' EA management tools. Wiki-based tools can provide stored searches, which are nevertheless no types in a conceptual sense. From this perspective, more comprehensive tool support, especially for these characteristics is required.

Prevalent EA management tools provide functionalities for visualizing and graphical modeling of EAs. The provided functionalities differ between the types of tools as distinguished along the dimension of "flexibility vs. guidance" [Bu08a, Ma08]. The more flexible tools offer out-of-the-box only basic graph-like visualizations. More complex visualizations, e.g. clustering elements, can be accessed using corresponding programming APIs. As we documented in [Ma08], these APIs are 'low level' and do not provide a concept as a re-usable building block for developing organization-specific viewpoints. The *methodology-driven* and the *process-driven* tools provide richer out-of-the-box visualization functionalities. These functionalities are nevertheless bound to the predefined information model of the tool and do not provide a basis for flexible definition of viewpoints. With this respect, our subsequent analysis takes a different approach and investigates prevalent tools for defining a *domain-specific language* (DSL). The following characteristics are relevant for the technical realization for supporting BEAMS-based viewpoints. The platform for technical realization must

- be able to bind to arbitrary information models.
- abstract from the details of implementing layout algorithms and provide re-usable layout component encapsulated in building blocks.
- support the automated generation of visualizations from other information representations.

- allow to use the visualizations as graphical models, such that a user can update the underlying information or parts thereof by editing the visualization.

In the following, we analyze five approaches from the field of DSLs, namely the *Graphical Modeling Framework* (GMF)[5], the *Graphical Modeling Environment* (GME) [Le01], *Graphiti*[6], *Oryx* [DOW08a, DOW08b], and the *SyCaTool* [Bu07a, Ac10, BGS10]. Both GMF and Graphiti originate from the eclipse ecosystem and build on the EMF as a common meta-modeling facility. A GMF definition for a graphical representation is a mapping from model elements in the information model to graphical primitives. These primitives are nevertheless confined to visible primitives, i.e., **map symbols** in terms of Section 6.3. Only basic **visualization rules**, as clustering are supported via dedicated relationships, as CHILDSYMBOL. Graphiti takes a slightly different approach and augments the information model with graphical information. While this allows to specify visualizations in a more direct manner, it does not affect the expressiveness of the definition language. Therefore, we regard Graphiti and GMF comparable with respect to their contributions as platforms for technical realization. The GME offers a design framework for DSLs that focuses on the syntactic aspects of such languages. In addition to sophisticated mechanisms for defining the syntax, the GME also allows to specify a mapping from the syntactic concepts to corresponding visualization elements. Similar to GMF, the provided technique is confined to **map symbols** and does not supply **visualization rules**. Based on the plugin mechanisms of GME, such rules and layout facilities based thereon can be established but have to employ low level API operations. Oryx provides the notion of the *stencilset* which defines both notational and syntactical aspects of a modeling language, whereby the notational aspects are defined based on the graphical expressiveness of the *Scalable Vector Graphics* (SVG)[7] recommendation. This in turn limits the available visual elements to **map symbols**, which are extended by Oryx with elementary **visualization rules** as clustering.

None of the aforementioned approaches puts strong emphasis on the generation of visualizations, nor on a rich set of visualization rules as concepts used to relate visualization elements with each other. The SyCaTool pursues a different approach and understands the mapping between the syntactic concepts and their graphical representations from an operational perspective as a model-transformation. This understanding is grounded in the basic principles that also promote our distinction between **notation function** and **representation function** as discussed in Section 4.3.5. A central principle behind the SyCaTool approach is a rich **visualization model** that encompasses both **map symbols** and **visualization rules**. The current state of the tool, as described in [Ac10, BGS10] also encompasses building blocks for defining the transformation, although the transformation remains unidirectional, i.e., can create visualizations from the information, but cannot feed changes back,

Table 7.2 shows the fulfillment of the technical realization requirements provided by the aforementioned approaches. This analysis reveals that the requirements can be fulfilled by today's visualization tools and model language platforms. It nevertheless also delineates that reusable layouts, as represented by the VBBs, are yet not fully covered, although the technical realization is possible.

[5] For more information see `http://www.eclipse.org/gmf/`, last accessed 04-05-2011.
[6] For more information see `http://www.eclipse.org/graphiti/`, last accessed 04-05-2011.
[7] For more information see `http://www.w3.org/TR/SVG/`, last accessed 04-05-2011.

	GME	GMF	Graphiti	Oryx	SyCaTool
arbitrary information models	●	●	●	●	●
re-usable layouts	◐[1]	◐[1]	◐[1]	◐[1]	●
automated generation	○	○	○	○	●
graphical modeling	●	●	●	●	○

[1] The tools do not provide visualization rules, but support basic layouts as clustering or line-routing.

Table 7.2: Flexible EA management tools and their support for the relevant viewpoint characteristics

7.3 Summary

This section covers two facets of tool support for BEAMS. On the one hand, we discuss how the development as well as the administration method, their constituting activities, and complementing techniques can benefit from a tool, the **BEAMS configurator**. On the other hand, we describe characteristics of the design of EA modeling languages that pertain to the configurable EA management tool needed to put such languages into action. In a first step, we elicit the use cases for the BEAMS configurator from the perspective of the two different types of users, namely the **enterprise architects** that enact the development method, and the **BEAMS administrators** that enact the administration method. We subsequently understand that the use cases can be realized using services 'typically' supplied by enterprise 2.0 platforms [BMN09]. We discuss this correspondence in more detail and take the design decision to use the open-source enterprise 2.0 platform Tricia to prototypically implement the BEAMS configurator. Regarding the implementation, different aspects are of interest: firstly, we detail how the services of the plugins *wiki* and *hybrid wiki* are used to store, to retrieve, and to edit IBBs, VBBs, and GBBs. We further detail that Dierl developed an additional plugin [Di10] that leverages the open-source modeling platform Oryx [DOW08a, DOW08b] to support editing of graphical information models and viewpoint definitions.

The characteristics of an EA management tool that supports BEAMS-based EA modeling languages are discussed both with respect to the information model and the viewpoint definitions. We revisit our survey [Ma08] on the state-of-the-art in EA management tools and derive that only *meta model-driven*, i.e., highly flexible, EA management tools can be adapted to support the information models that result from the development method. Special characteristics of these models, as **non-rigid types**, **bi-temporality** and **specialization by constraint** are supported by only a minority of today's EA management tools. Therefore, we conclude that tools supporting the BEAMS-based information models exist, but also that future development in this field is needed to provide more comprehensive support for these special characteristics. Regarding the viewpoint aspect of the EA modeling languages developed using BEAMS we build on a result in [Ma08], according to which the graphical modeling capabilities in current EA management tools are 'configured' by programming against low-level visualization APIs. In the light of characteristic, we decide to analyze frameworks and platforms for visualization as well as for realizing a *domain specific language* (DSL). These frameworks actually provide good support for BEAMS-based viewpoints based on arbitrary information models. With respect to the support for re-usable layouts, automated generation of visualizations, and graphical modeling, we have to distinguish between the visualization

and the DSL frameworks. Former provide good support for automated layout calculation, but are confined to visualization capabilities only. Latter frameworks in contrast support graphical modeling, but stay to a low-level understanding of layouts, most often not supported by automated generation mechanisms. From this, we conclude that different tools can provide the relevant characteristics for realizing BEAMS-based viewpoints. Nevertheless, currently no comprehensive coverage of all characteristics can be achieved in a single tool.

A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join it and flow with it.

Paul Artreides – Dune

CHAPTER 8

Evaluating and Applying BEAMS

This chapter decomposes into two main parts. Firstly, we evaluate the development method and the underlying organized library of building blocks. Secondly, we present results from two practice cases. The evaluation is of theoretic nature, following the research method outlined in Section 2.3, as a comprehensive evaluation of the development method would require a time-frame beyond the scope of a PhD-thesis. Therefore, the evaluation in Section 8.1 focuses on quality criteria for an EA modeling language and requirements for the development method. The application in Section 8.2 is no evaluation but provides an expository instantiation of the development method. We report on two cases, where BEAMS and its development method are applied to develop organization-specific EA modeling languages.

8.1 Theoretic evaluation of BEAMS

The fourteen quality criteria for EA modeling languages, as devised in Section 4.1, are organized into six groups along the *appropriatenesses* of Krogstie's language quality framework [Kr02]. Subsequently, we argue on the characteristics of EA modeling languages, that are developed using BEAMS, with respect to each of these appropriatenesses.

By design a BEAMS-based EA modeling language offers several characteristics relating to interpersonal quality, i.e., to *communication appropriateness*. As part of the activity **characterize situation** in the development method, the enterprise architects designate which stakeholder is interested in which **area-of-interest** in the enterprise. These interests reflect the actual information demand of the stakeholder, if such demand is admissible against the background of the organization's information policies. In the steps of the subsequent activity **configure the method** the enterprise architects develop one or more viewpoint that reflects this information demand and graphically represents the corresponding information from the organization's information model. Via its associated **viewmodel**, the viewpoint establishes

both a projection and a selection of the overall available information to the stakeholder-specific part. The viewmodel is further interesting from a different perspective as the chosen kind of relationship between the viewmodel and the information model determines the nature of the viewpoint either committing to a **representation function** or to a **notation function**. Stakeholders assigned to a viewpoint of representation quality can read the information contained in the corresponding viewmodel, whereas stakeholders using a notation viewpoint are also allowed to update the contained information or parts thereof. Based on these mechanisms and distinctions, any EA modeling language based on BEAMS supports an implicit rights and responsibilities management in the sense of quality criterion **Cm1**. Rights and responsibilities are therein approached from the information perspective, i.e., the information model assigned to stakeholders is used to determine their rights and responsibilities put into effect via corresponding viewpoints. The meta-language for defining EA information models, as devised in Section 6.2, explicitly supplies a mechanism for specifying temporality aspects. Temporal modeling can thereby be selected for certain parts of the information model, whereas other parts of the model have no dedicated operational semantics with respect to time. Thereby, the enterprise architects developing the organization-specific information model can exert control over historization and traceability of EA model changes on a fine-grained level, i.e., regarding singular TYPES, PROPERTIES, or RELATIONSHIP ENDS. With this respect, quality criterion **Cm2** is partially fulfilled: for TIMEDEPENDENT elements in the information model it is possible to determine 'what was edited when'. Mechanisms for supplying the 'who' are not defined in the meta-language for EA information modeling. The two EA management tools that according to our analysis from Section 7.2 support temporal modeling, also store the originator of a model change. Based on this tool support, a BEAMS-based EA modeling language is able to completely fulfill the temporal modeling demand.

BEAMS-based EA modeling languages per design have several characteristics that facilitate the comprehension of language statements, i.e., that contribute to *comprehensibility appropriateness*. Enterprise architects iteratively execute the activity **characterize situation** as part of the development method. During each iteration, the **areas-of-interest** for the identified EA stakeholders are described in terms of a concern and corresponding cross-cutting aspects. Each of these areas-of-interest is reified in a corresponding information model fragment. This information model fragment is composed from the IBBs that reflect the concern and the cross-cutting aspects, that the enterprise architects selected. The selection is supported by techniques to determine, if an IBB can be **consistently embedded** into the already specified information model of the organization-specific configuration. Against this background, the enterprise architects cannot add IBBs or information model fragments that would cause inconsistencies. Based on the selected problem, i.e., combination of concern and cross-cutting aspects, the enterprise architects develop the viewpoints that are used by the stakeholders to access and update the corresponding information. Based on the stakeholders' input, the enterprise architects iteratively augment the information model and specify viewpoints taking a particular perspective on this model—in short, they establish a multi-perspective EA modeling in the sense of quality criterion **Cp1**. Every model element in the EA information model is not a purely syntactical concept, but also retains a link to a corresponding predicator in the glossary of the EA management function. The GBBs are used to develop the corresponding glossary and the glossary itself presents an interlinked system of terms that are complemented by textual descriptions of their meaning. The prospective users of the EA modeling language can access this glossary from the corresponding EA models. All EA management tools an-

alyzed in Section 7.2 provide support for global glossaries, i.e., provide means to define the terms at a central point and re-use the predicator in different viewpoints. In this respect the quality criterion **Cp2** is fulfilled by any BEAMS-based EA modeling language. Furthermore, the tool support for such languages enables the EA stakeholders to use the predicators in the glossary.

The organized library supplies a wide range of IBBs targeting different areas-of-interest in the organization. We derived these IBBs from the information models provided by the EA management approaches and frameworks analyzed in Chapter 3 as well as in [BS11]. In Table 3.18 we summarize the coverage of the business and IT aspects as described in the analysis framework. In this respect, the organized library of IBBs inherits the coverage provided by each of these approaches and unifies the coverage to a comprehensive image of the EA domain. Table 8.1 describes how many approaches contributed to the knowledge base of BEAMS with respect to which aspect of the EA according to our understanding of the domain. Above

	Business & organization	Application & information	Infrastructure & data
Black-box perspective	10	7	2
White-box perspective	20	18	16

Table 8.1: Contributions to EA domain coverage

numbers show that a comprehensive coverage of the EA domain ranging from business to IT is incorporated into the library of IBBs. The numbers also show that a service-centric perspective on infrastructure and data is not prevalent in today's EA management approaches. Quality criterion **Do1** is nevertheless fulfilled, at least to the same degree as the state-of-the-art in EA management reflects the corresponding aspects. The organized library of BEAMS supplies different IBBs that relate to cross-cutting aspects of EA management. Specifically, IBBs for planning EA transformations based on DEMANDS and PROJECTS are supplied along IBBs for providing transformation guidance via STANDARDS. The meta-language for EA information modeling as described in Section 6.2 provides the concept of **dispersive types**. Based on these types, cross-cutting aspects can be modeled in isolation, i.e., without being tightly linked to corresponding architectural elements. The corresponding IBBs reflect the nature of the cross-cutting aspects and their related concepts without directly attaching the aspect to a particular concern in the overall EA. Based on the cross-cutting IBBs, a BEAMS-based EA modeling language can be developed to cover transformation aspects and guidelines on different architectural elements, thereby fulfilling quality criteria **Do2** and **Do3**.

The enterprise architects develop EA modeling languages with BEAMS to empower different stakeholders to express the relevant parts of their EA knowledge. This entails that the modeling languages must be appropriate for their prospective users especially with respect to externalizing EA information. Central to this consideration is the notion of *relevance*, which we already discussed in Section 1.1. Different enterprises address different EA management-relevant problems and therefore have a different understanding of relevance. In the activity **characterize situation** the enterprise architects perform two steps, in which they select the EA management goal and the EA concern, respectively. Together the goal and the concern designate an EA management problem that is relevant to the stakeholders identified in the first step of the activity. Each concern corresponds to an IBB that supplies the necessary

and sufficient model elements needed to describe the corresponding part of the EA in the sense of quality criterion **Ex1**. The fine-grained structure of the IBBs thereby supports the concern-specific design of the EA modeling language. The organized library of building blocks further supplies different operationalizing IBBs for a cross-cutting aspect as a goal. These IBBs describe **questions** reflecting the goals in **dispersive types** that supply **metrics** and can be attached to affected architectural elements. Based on the selected goal, the enterprise architects choose the adequate question and thereby an IBB that contains the necessary and sufficient model elements for the particular operationalization. The model elements in this IBB are in the next step of the development method integrated into the elements in the concern IBB and the enterprise architects create the information model fragment that covers the corresponding EA management-relevant problem in an operationalized manner. In this respect quality criterion **Ex2** is fulfilled by any BEAMS-based EA modeling language.

In Section 7.2 we already discussed different characteristics of a BEAMS-based EA modeling language and analyzed to which extent today's EA management tools as well as DSL frameworks can be used to technically realize such modeling languages. We briefly revisit the corresponding arguments. Quality criterion **Te1** is fulfilled by design in the VBBs. Each VBB represents a function that maps elements of viewmodel to corresponding visualization elements. This function is a specialization of the general mechanism of a model transformation, for which technical realizations as QVT [Ob05] or the *Atlas Transformation Language* (ATL) [AT06] exist. The formal meta-language for the EA information models provides a basis for techniques that build on EA information models. The dispersive types in the meta-language further supply a mechanism that can be used to anchor quantitative analyses in the model elements and the thereby represented architectural elements. **Specialization by constraint** is useful for specifying qualitative analyses, e.g. via queries to model elements that have a particular characteristic. In this light quality criterion **Te2** is partially fulfilled, as techniques to supply computation rules on architectural elements and automated quantitative analyses are not supported by the IM2L.

During the activity **configure the method** the enterprise architects identify the viewpoints needed to convey the relevant architectural information. The organized library of VBBs helps to design these viewpoints by providing indications, for which stakeholders the particular VBB is suited better or worse. Based on this information and employing the expressiveness of the visualization model from Section 6.3 especially with respect to the supported symbols, the enterprise architects can define viewpoints that match their target stakeholders. Regarding the symbolic representation of the architectural concepts, the enterprise architects can choose freely. With this respect, quality criterion **Us1** is partially fulfilled. This means that the enterprise architects are empowered to freely configure the viewpoints and symbols, but are not further supported in selecting the most familiar graphical representation in a particular usage context. The development method is an iterative method, which builds on an existing organization-specific configuration for EA modeling languages based on building blocks. While this configuration initially is empty, the method also supports steps to extend an admissible configuration in a consistent and implementable manner. These techniques ensure that BEAMS-based EA modeling languages remain adaptable and extensible. This extensibility nevertheless reaches its limit when no more admissible IBBs are available, i.e., if given the organization-specific configuration no additional IBB can be consistently embedded. The relationships between the IBBs, especially the **subsume**-relationships allow to derive possible paths for evolving an existing information model and the EA modeling languages building

thereon. The aforementioned mechanisms and facilities ensure that EA modeling language developed using BEAMS remain extensible with respect to future changes in goals and concerns. Per design these languages therefore fulfill quality criterion **Us2**. A key step during the activity **characterize situation** is the adaptation of the glossary. The enterprise architects search the glossary of EA predicators as supplied by the organized library of GBBs and establish an organization-specific terminology by providing synonyms for termini. This terminology is consistently used in displaying **areas-of-interest**. As the predicators also serve as designators for the model elements in the information models, the adapted terminology is further propagated to the organization-specific information model as well as to the IBBs, when browsed from the perspective of the corresponding organization. Thereby, quality criterion **Us3** is fulfilled, as BEAMS-based modeling languages are adapted to the terminology of the corresponding linguistic community.

In Section 4.2.1 we elicited six requirements for our development method for EA modeling languages. These requirements are based on the six general principles of modeling, as described by Schütte and Rotthowe in [SR98, pages 245–249]. In the following, we briefly discuss these requirements and their fulfillment by the presented development method. The building blocks constituting the knowledge base of BEAMS are derived from the state-of-the-art literature, as discussed in Section 3.3, and from EA management patterns presented in the EA management pattern catalog [Ch10, Er10]. The pattern in particular and most of the analyzed approaches in general formulate prescriptions that have proven to work in practice. From this perspective, we can state that BEAMS itself is based on practice-proven solutions in the sense of requirement **MU2**. BEAMS can further be extended by such solutions via the **administration method**. This method builds on the concept of the pattern as 'draft' for a building block. The nature of such patterns as observed successful solutions to dedicated problems ensures that future extensions of the organized library of building blocks are also practice-proven in the sense of requirement **MU3**. The building blocks in the organized library of BEAMS are derived from prevalent EA management approaches and frameworks. The blocks' prescriptions are formulated in the unifying terminology of BEAMS. This terminology has been derived by applying the steps of the **administration method** to resolve inconsistencies and homonyms. Based on this terminology and the conceptual framework for the building blocks as well as their relationships, we regard requirement **MU4** to be fulfilled, i.e., semantical and syntactical correctness in BEAMS to be given. The BEAMS terminology does nevertheless not exist in isolation, but links to the termini used by the different incorporated approaches, designates synonyms, and highlights homonyms. The enterprise architects can make use of these linkages between the approaches, when they seek to combine BEAMS with another EA management framework. In the sense of requirement **MU6**, the terminological linkages further provide a starting point for integrating and comparing BEAMS and other EA management approaches. These comparisons are nevertheless confined to the fourteen approaches that have been used as information sources for the development of BEAMS. Requirement **MU1** regarding the ease-of-use of the method cannot be theoretically evaluated. The tool support for the method as described in Section 7.1 facilitates the use of the method, such that requirement **MU5** can be regarded fulfilled. Results regarding its applicability can only be gathered in practice cases as discussed in Section 8.2.

We subsequently reflect our method and its complementing organized library of building blocks against the design research guidelines of Hevner et al. [He04]:

Design as an artifact Our building block-based framewor (see Section 6.1) and the two meta-languages for describing information models (see Section 6.2), and viewpoints (see Section 6.3) supply a vocabulary for describing EA problems, contexts, and solutions. Together they form the *constructs* on which the **organized library** of building blocks is based. The library in turn is the *model*, which is used by the *methods*, namely the **development method** and the **administration method**, see Sections 5.3 and 5.4. The prototypic toolset described in Section 7.1 *instantiates* our design solution.

Problem relevance The plurality of EA management approaches and EA modeling languages, as discussed in Section 3.3, gives an indication towards the relevance of the field. Our analyses in [Bu07b] as well as the work of Aier et al. [Ai08a] showed that EA modeling languages have to organization-specific, in order to be useful for the stakeholders. Todays EA management approaches and frameworks in contrast do not account for this kind of specificity, rendering the problem of how to develop organization-specific EA modeling languages a relevant one.

Design evaluation Due to the nature of the design artifact, compiling practice-proven prescriptions from different sources, the evaluation is confined to target the integrating structures and conceptualizations. These elements cannot be directly evaluated by practical application due to the sheer number of admissible combinations thereof. Further, a practical application would take about five years according to the findings of Ross and Beath [RB06]. Therefore, we subject the development method but also the outcomes of the application of this method to a theoretic and argumentative evaluation.

Research contributions The design artifact presents five contributions to the scientific knowledge base. We contribute two **meta-languages** for describing EA information models and viewpoints, respectively. We present an **organized library** of language building blocks, consolidating and structuring practice-proven prescriptions for developing EA modeling languages according to our **development method**. This method is supported with a **technical toolset**, the BEAMS configurator, that realizes the method's activities. Finally, we present an **administration method** which can be used to extend the organized library. Latter method is, as discussed in Section 2.3, an application of a more general method for pattern-based construction of a design theory nexus instantiation.

Research rigor We developed and researched BEAMS using the research method described in Section 2.3. This method is based in the conceptual foundations provided by Pries-Heje and Baskerville in [PHB08] and rigorously builds on a pattern-based way for developing design theories [BMS10k]. The sources for extracting the patterns in turn are results of rigorous research documented in the state-of-the-art literature of the discipline as analyzed in Section 3.3.

Design as a search process The **organized library** of building blocks is the result of an iterative search process that builds on the findings presented in a previous artifact, the EA management pattern catalog [Ch10]. The **administration method** allows to continue the search process and to integrate novel practice-proven solutions into the knowledge base of BEAMS.

Communication of research We have communicated the contributions of this thesis or preliminary versions thereof in different ways. Firstly, we have submitted the parts of the contribution to scientific conferences, e.g. in [BMS10j, BMS10a, BMS10c, Bu10a, BMS11a, Bu11b, BMS11b]. Secondly, we have taught BEAMS as part of a master level lecture[1] on EA management to students as well as to the industry partners hosting the complementary student mini-projects [Bu09i, BES10]. Thirdly, we have presented the organized library of building blocks as well as the development method to practitioners in the field of EA management at several practice relevant events, e.g. *EAMKON 2010* and *cq-event IT-Industrialisierung*.

8.2 Exemplary applications of BEAMS

In the following, we present two exemplary applications of our development method at partnering organizations. The applications are intended to demonstrate the usability as well as feasibility of the development method. In addition, we discuss lessons learned in the corresponding practical settings. Both practical applications were performed by master-level students in close cooperation with relevant representatives of the partnering organizations and under supervision by us. This means that:

- the students act as **enterprise architects** in the sense of the development method, i.e., perform the steps of the method, whereas
- the representatives of the organization act as **stakeholders** for the EA modeling language.

We present both practical applications in an anonymized way along the following structure:

Organizational environment briefly describes the partnering organization, gives a short overview on the student's involvement in the organization, and describes available tool support for EA modeling.

Identified EA management-related problems introduces the goals and concerns that have been selected and names the corresponding stakeholders.

Results describes the integrated information model of the case, gives explanations to the used glossary, and shows exemplary visualizations.

8.2.1 An application case from an IT service provider

In this exemplary application, alluded to as application case IT in the following, the development method was applied by a student between September 2010 and January 2011 at an IT service provider for the public sector. The method was thereby used to develop organization-specific EA modeling languages.

[1] For more information, see `http://wwwmatthes.in.tum.de/wikis/sebis/vorlesung-eam`, last accessed 04-05-2011.

Organizational environment
The organization is the IT service provider for a large number of federal ministries of an European country. Over the years, the different ministries have sought to establish IT governance initiatives, which remained confined to the local IT portfolio. Only recently, the IT service provider itself decided to establish an EA management function. This management function is intended to support the IT governance endeavors in the different ministries by providing relevant information for decision making. An employee of the IT service provider is responsible for developing the EA management function and in parallel documents the models and viewpoint for his master's thesis in computer science. He therefore takes a double role as enterprise architect applying the development method and as stakeholder to the method's outcome. In addition, he works in close cooperation with an IT responsible from one of the ministries, whose IT infrastructure should be used to pilot the EA management initiative. Performing the literature search for his thesis, the student comes across the building block-based EA management approach and decides to establish contact with us. We conduct two on-site workshops to familiarize the student with the fundamental principles of our approach and provide guidance, where necessary. The student decides to concentrate on the language aspect of the EA management function and proceeds through the steps of the **development method**. The thereby created EA modeling languages are prototypically implemented using the wiki-based approach to EA management together with the visualization capabilities provided by the SyCaTool as described in Section 7.2.

Identified EA management-related problems
The stakeholder identification has already been finished as the student's work starts. IT responsibles from the ministries are the designate stakeholders of the EA management function, which in turn should "increase transparency". Different parts of the overall EA are of interest in this context, namely the concerns "organizational units host business applications", "business application using infrastructure", "business applications use technologies", and "service provisioning" by the business applications. As the IT service provider uses German as working language, the corresponding **predicators** are translated to the organization-specific terminology as shown in Table 8.2. In the following, the case description uses the organization-specific terminology.

BEAMS predicator	Organization-specific predicator
Business application	Anwendung
Infrastructure element	Hardware
Organizational unit	Organisationseinheit
Service	Verfahren
Technology	Technologie
(Organizational unit) hosts (business application)	betreibt
(Business application) provides (service)	unterstützt
(Business application) uses (technology)	nutzt
(Business application) uses (infrastructure element)	läuft auf

Table 8.2: Application case IT: mapping of organization-specific glossary

Results

The abstract goal "increase transparency" does not entail a **cross-cutting IBB**, such that the different **concern IBBs** can be integrated into a comprehensive information model without previous creation of model fragments. The only exception in the context is the IBB "service provision", who introduces the **category** of the SERVICE PROVIDER. This category has to be bound to the corresponding EA concept, which is identified by the enterprise architect as "Anwendung". The resulting information model is shown in Figure 8.1.

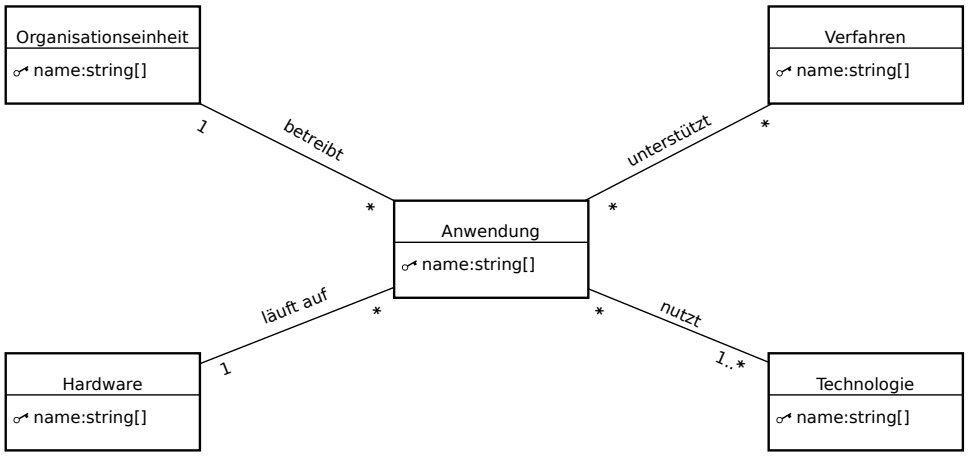


Figure 8.1: Application case IT: organization-specific information model

Based on the identified concerns, the enterprise architect develops two viewpoints. The first viewpoint uses a cluster VBB as **base VBB** to visualize, which *Verfahren* is supported by *Anwendung*. The second viewpoint builds on a matrix VBB to display which *Anwendung* uses which *Technologie*. The organization-specific configuration of the second viewpoint is displayed in Figure 8.2.

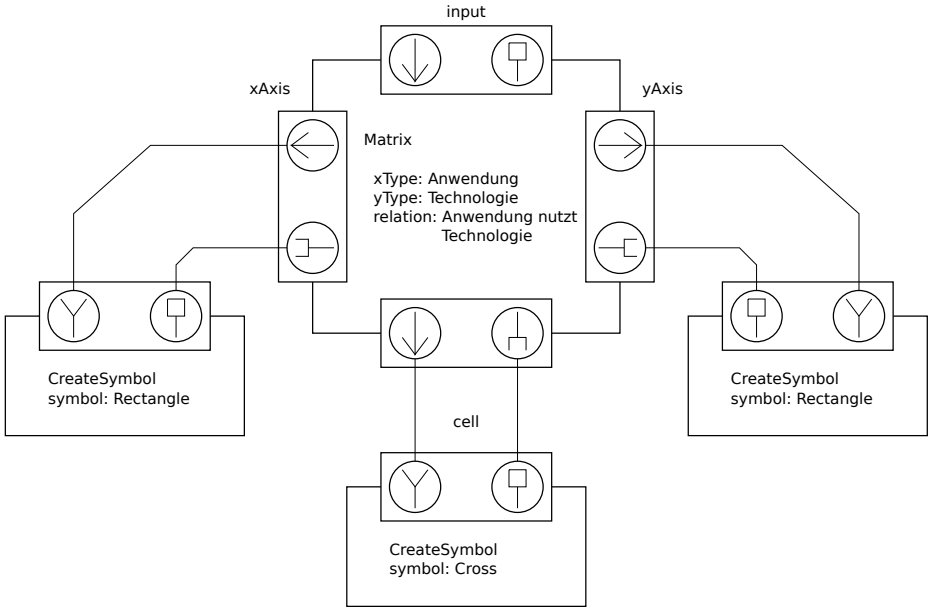


Figure 8.2: Application case IT: definition of matrix viewpoint *Anwendung-Technologie*

Anonymized variants of the visualizations resulting from the two defined viewpoints, which are used representation-only, are shown in Figures 8.3 and 8.4. These views are automatically created based on the EA information stored in the hybrid wiki using the SyCaTool (see Section 7.2).

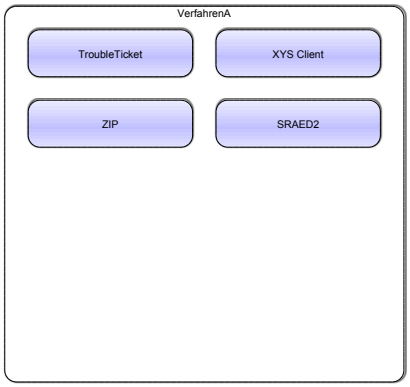


Figure 8.3: Application case IT: clustered view

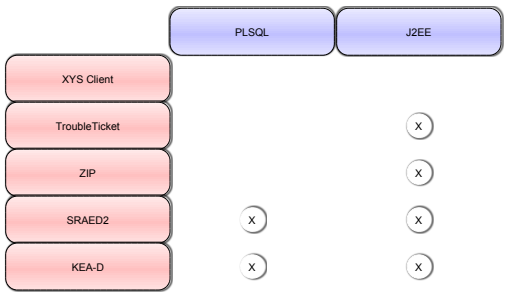


Figure 8.4: Application case IT: matrix view

8.2.2 An application case from the public sector

In this exemplary application, alluded to as application case PS in the following, the development method was applied by a student between October 2010 and January 2011 at the central IT department of an association of research institutes. The method was thereby used to develop an EA management function from scratch. We subsequently report on the language-related aspects of the development and its resulting EA management function.

Organizational environment
The organization is constituted from independent research institutes that perform research in many different areas. The diversity of the areas and the 'academic freedom' of the individual institutes has promoted the development of a diverse EA with a highly heterogeneous IT. The central administrative division of the organization seeks to leverage synergy effects between the different institutes and has thereto recently established the position of an enterprise architect. Synergy effects are especially expected in the field of IT system usage, where reduced diversity in supportive applications will lead to increased cost-efficiency and a more sharp skill-setting for the IT departments. A master-level student of Technische Universität München worked, as part of a "guided research" with the enterprise architect to establish the EA management function. Prior to the guided research, the student had attended our BEAMS-based lecture on EA management[2]. The student and the enterprise architect held a series of workshops going through the different steps of the **development method**. We supervised the student in the workshop preparation and accompanied the student to the workshops.

[2]For more information, see `http://wwwmatthes.in.tum.de/wikis/sebis/vorlesung-eam`, last accessed 04-05-2011.

Identified EA management-related problems
In the activity **characterize situation**, the student identified two stakeholders of the EA management function, namely the "IT project managers" responsible for working with external IT development companies in realizing IT services, and the "upper management" of the central branch. The upper management stated the abstract goal "increase transparency" targeting the concern "business applications support business processes at organizational units", when asked for the EA management-relevant problem that should be addressed. As the student elicited, the different research institutes are understood as the "organizational units" according to former concern. The project managers are interested in "increasing homogeneity" (abstract goal) with respect to the "technologies used by business applications". The student and the enterprise architect elicit two different perspectives on homogeneity, depending on the vendor of the business application. Applications developed by SAP are considered standard, and not targeted by the homogenization endeavor, whereas applications of other vendors are standardized based on the technologies, they use.

Results
The abstract goal "increase transparency" does not contribute a **cross-cutting IBB**, such that the **concern IBB** is incorporated in the information model without adaptations. The abstract goal "increase homogeneity" is operationalized via the cross-cutting IBB "book of standards", which is added to the corresponding concern IBB, relating technologies and business applications. Both IBBs are integrated into an information model fragment as shown in Figure 8.5. Thereby, the **categories** STANDARDIZABLE and STANDARD are identified with the **substantial types** BUSINESS APPLICATION and TECHNOLOGY, respectively.

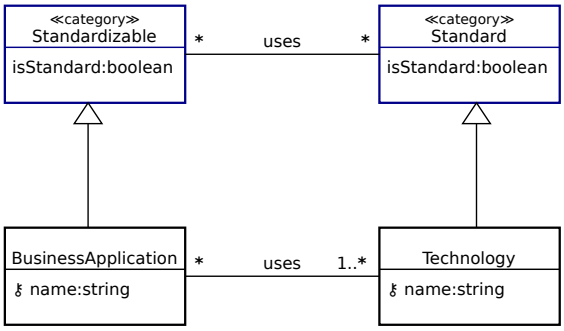


Figure 8.5: Application case PS: information model fragment for technology-based standardization

The information model fragment from Figure 8.5 is further refined to reflect the fact that the vendor of an application is relevant with respect to homogeneity. Firstly, the property VENDOR is added and secondly, **specialization by constraint** is applied to exclude applications from vendor SAP from technology-based standardization. The integration of this adapted information model fragment with the fragment, discussed above, yields the organization-specific information model as shown in Figure 8.6.

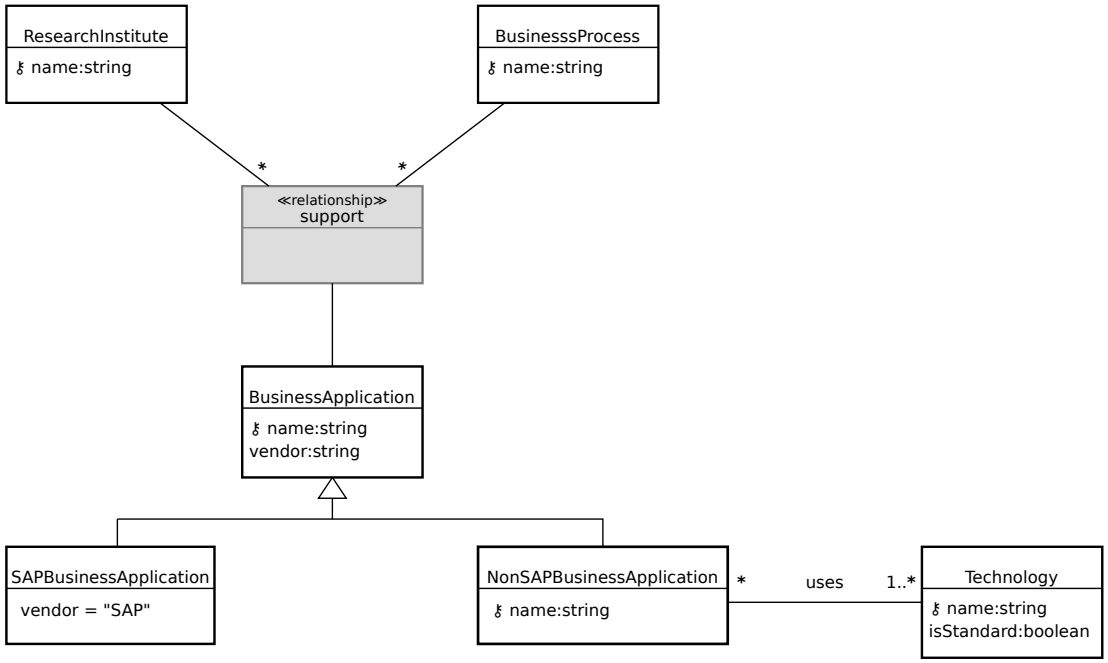


Figure 8.6: Application case PS: organization-specific information model

8.3 Summary

The theoretic evaluation targets the development method from an outcome-oriented, a method-oriented, and an artifact-oriented perspective. From the perspective of the outcome, we argument how the development method and its underlying organized library of building blocks are useful to construct organization-specific EA modeling languages that fulfill the quality criteria raised in Section 4.1. Regarding the method perspective, we discuss, if the development method itself satisfies the domain-specific requirements for such method as described in Section 4.2.1. The evaluation from these two perspectives yields the evaluation results as summarized in Table 8.3. Therein (●) denotes that a characteristic is completely provided and (◐) designates a partial fulfillment.

Quality criteria				Method requirements	
(Cm1) Right & responsibility	●	(Ex1) Concern coverage	●	(MU1) Easy to use	-[1]
(Cm2) Traceability	●	(Ex2) Goal coverage	●	(MU2) Practice-proven	●
(Cp1) Multi-perspectivity	●	(Te1) View generation	●	(MU3) Extensible	●
(Cp2) Glossary support	●	(Te2) Analysis coverage	◐	(MU4) Correct	●
(Do1) Business & IT coverage	●	(Us1) Representation fit	◐	(MU5) Tool-supported	◐[2]
(Do2) Transformation cover.	●	(Us2) Sustainability	●	(MU6) Integration	◐
(Do3) Guidelines coverage	●	(Us3) Terminological fit	●		

[1] The ease of use cannot be evaluated theoretically. The practical applications nevertheless indicate, that the development is easy to use after an initial familiarization phase.

[2] The prototypic tool described in Section 7.1 demonstrates that tool support for the method is feasible, but further several mechanisms remain to be added.

Table 8.3: Fulfillment of language and method requirements

Finally, we understand the development method as artifact of design research and revisit this artifact as well as the research process that has lead to its creation from the perspective of the general design guidelines discussed in Section 4.2.2. Therein, we show a good fulfillment of the design guidelines of Hevner et al. [He04].

In addition, we present findings gathered during the BEAMS-based development of EA modeling languages. The two application cases conducted from September 2010 to January 2011 apply the development method to establish a new EA management function. Particularly, we focus on the language-related aspects of the function and apply the steps of the development method in the sense of Section 8.2 for:

- **nexus-driven problem elicitation**, in which the particular EA management-related problems of the using organization were identified,
- **nexus-driven artifact design**, in which the development method was applied to construct organization-specific EA modeling languages, and
- **nexus-driven artifact evaluation**, in which the industry partners of the cases were asked for their feedback regarding the practicability of the approach and the usability of the EA modeling languages.

The artifact evaluation showed the stakeholder's initial satisfaction with the modeling languages, although these statements remain punctiform. A comprehensive evaluation required a time-frame of at least one year, which was beyond the scope of this thesis. The subjective feedback of the method's users in addition showed that BEAMS in general is useful for creating EA modeling languages, although the rudimentary tool-support especially regarding the integration of building blocks posed a challenge. Furthermore, the demand to familiarize oneself with the dedicated meta-languages for EA information modeling and viewpoint definition, respectively, was mentioned. Here again, a stronger tool-support is expected to help prospective users of the development method.

This... is yours now. And you have an obligation to do as we have done. To teach [those] that will follow you and, when your time comes, as ours has, to step aside and allow them to grow into their own destiny

Lorien, Into the fire – Babylon 5

CHAPTER 9

Summary, Critical Reflection, and Outlook

Today's organizations find themselves confronted with a changing environment, whose challenges demand for an embracing management approach. EA management responds to this need, but has to be performed in an organization-specific way to effectively leverage benefits while being efficient regarding the investments taken. The ensuing need to develop organization-specific EA modeling languages tailored to the stakeholders' particular information requirements motivates this thesis and its research objective.

> ***Research objective:*** Develop a method for designing and re-designing organization-specific EA modeling languages to support multi-viewpoint EA management based on redundancy-free best-practice knowledge documented in current EA management approaches.

In this thesis we approach above objective and present BEAMS, a building block-based development method for EA modeling languages. In Section 9.1 we summarize and reflect the thesis' contributions that constitute the method. Section 9.2 furthers our discussions from [BMS10l] and gives an outlook on research topics that are made possible by but go beyond the core ideas of BEAMS.

9.1 Summary and critical reflection

The building block-based development method established in this thesis realizes an "integration approach" [Bu07b] for designing EA management functions. According to our perspective, an EA management function constitutes of a method and a language part. The method part describes which participating actors perform which tasks using which EA modeling languages as described in the language part. The development method of BEAMS provides re-usable

solutions, so called **building blocks** for both the method and the language part. These building blocks are presented in two theses, of which the current one focuses on the language aspect. Figure 9.1 gives an overview of the different contributions of this thesis focussing on the language aspect.

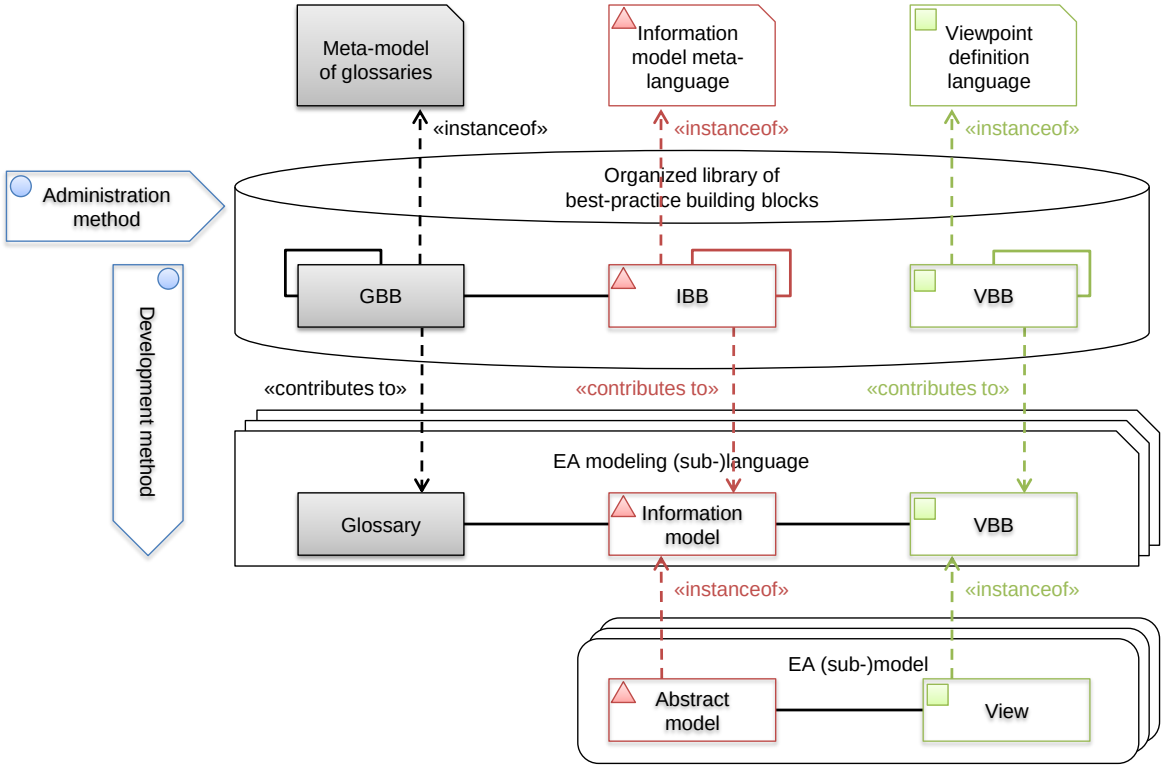


Figure 9.1: Reflecting the contributions of the thesis

Central is the notion of the **language building block**, of which this thesis introduces three particular specializations:

The **information model building block**s describe re-usable fragments for the syntax of an EA modeling language. They cover a specific **area-of-interest** in the EA, which can either be an architectural concern or a cross-cutting aspect pertaining to different concerns.

The **viewpoint building block**s describe re-usable fragments for the notation of an EA modeling language. They present mappings from syntactic primitives to visualization elements, both symbols and visualization rules.

The **glossary building block**s describe re-usable fragments for semantics definitions of an EA modeling language. They textually describe the semantics of syntactic primitives in context.

The building blocks are based on two dedicated meta-languages—the *information model meta-language* (IM2L) and the *viewpoint definition language* (VDL)—and a meta-model for describing glossaries. This meta-model supports the utilization of glossary building blocks and com-

plements the formal meta-languages for syntax and notation. This meta-model specifically accounts for the communicative nature of EA management. It provides relationships that allow to interlink terms originating in diverse **linguistic communities**. These relationships are also used to facilitate the evolution of an organization-specific glossary, by keeping the adapted terms rooted in the glossary building blocks of BEAMS.

The IM2L follows the paradigm of object-oriented meta-modeling and presents **model elements** tailored to the domain of building block-based modeling of evolving universes. The IM2L introduces specific model elements for modeling change on the instance level (**bi-temporal modeling**) as well as for modeling change on the conceptual level (**non-rigid typing**). It further supplies a sound model for the concept of identity, which allows to distinguish between **substantial properties** of the architecture and cross-cutting aspects reflected in **dispersive types**. The native support for **specialization by constraint** provides a basis to use the IM2L in environments, where architecture conceptualizations are applied differently in different parts of the using organization. The formal underpinnings of the IM2L give rise to a set of techniques centering around **embedding**. This relationship designates that one information model fragment can supply and/or store at least the information corresponding to another information model fragment. Based on this knowledge, the enterprise architects can ensure that an information model building block is consistently, i.e., without loss of expressiveness, embedded into the organization-specific information model. Further, embedding-relationships are used to derive possible evolution paths for a configured information model.

The VDL provides a formal basis for describing viewpoint building blocks. In particular, the language allows to specify the signature of such a building block, which in turn realizes a specific mapping, i.e., either a bijective **notation function** or a surjective **representation function**. The VDL ensures that an enterprise architect can only configure syntactically correct mappings from the mappings provided by the viewpoint building blocks. The VDL further allows to derive the **viewmodel** that corresponds to such configured mapping or **hybrid viewpoint building block** in terms of the thesis. The viewmodel abstractly designates the minimum set of information that is necessary to create a corresponding visualization. It further designates the model elements that are sufficient to reflect visual changes in informational ones. In the development method the enterprise architect binds the abstract viewmodel to actual model elements from the selected information model and thereby establishes an organization-specific viewpoint. The quality of the viewpoint, either notating or representing, is conversely determined by the actual binding of the concepts chosen.

The building blocks of all three types are interlinked via formal relationships that act as organizing principle on the knowledge base of BEAMS. The **organized library** of language building blocks is used by the **development method** to construct organization-specific EA modeling languages. It is constituted of two major activities, in which the situation of the target organization is characterized and subsequently admissible building blocks are selected as well as integrated. Both selection and integration of building blocks are only possible based on the meta-languages and their formalizations, which support the notion of consistent embedding. Two exemplary applications show the applicability of the development method and its underlying organized library. Together with the theoretic evaluation of the method in Section 8.1, they give indications on the validity as well as the utility of the contribution. In

these exemplary cases not all building blocks or combinations thereof could be covered. This nevertheless ascribes to an inherent property of the field of EA management. EA management is a management function, whose impact on the overall performance of the enterprise is not immediately evident.

In addition to a method for using BEAMS, this thesis also supplies a method for administrating the organized library of building blocks—the **administration method**. This method institutionalizes the research method that was applied in developing BEAMS during the thesis. The method builds on a critical linguistic perspective towards conceptual modeling and can be used to elicit practice proven prescriptions in the form of design theories. These prescriptions are identified as pattern candidates for EA modeling, proven by repeated observation to be patterns, and are consolidated to design theories based on a consistent terminology for EA modeling. The different design theories are further interlinked in the organized library, which in turn instantiates a design theory nexus [PHB08].

Discussions on implementation aspects of BEAMS indicate how both the development method and the administration method can be supported by a corresponding tool. This thesis establishes the conceptual basis for BEAMS and discusses how the organized library can be implemented to facilitate the development method. The wiki-based approach chosen for representing the organized library allows a maximum of flexibility regarding the documentation of the building blocks. We could further show how current EA management tools are capable to support an EA management function developed with BEAMS.

Future research can target the applicability of the development method described in this thesis. Further application cases can provide more tangible results about the method's usability. In the light of the ensuing need for comprehensive guidance in this field, we nevertheless decided to create an innovative artifact that enables to integrate prescriptions for EA management on a sound conceptual basis. We did this, although we are aware that it is impossible to show the usability of the artifact on the same level of generality as the artifact applies. Due to the sheer amount of combinations of building blocks, it would have been a research stream exceeding a single thesis in both duration and extent to evaluate all possible prescriptions in practice cases. As we have simplified the development of EA modeling languages, application cases could be executed by students on master level in different organizations as part of their curriculum.

The question of implementing BEAMS remains an open issue for future research. The sketched implementation remains prototypic and does not relief the user of manual tasks, that could be automated. This may ascribe to the nature of the implementation platform chosen. The choice of enterprise 2.0 platforms can be contested and other tools, as decision support systems can be of interest. In the light of the discussions undertaken by Pries-Heje and Baskerville [PHB08] an open platform for supporting the development method nevertheless seemed to us most advisable and we opted for semi-structured wiki-based information storage and retrieval. Future development of this platform in close cooperation with a community of users can help to refine the elicited use-cases and to provide more sophisticated mechanisms for guiding enterprise architects.

9.2 Outlook on future research topics and directions

BEAMS consolidates practice proven knowledge from a broad variety of sources. Its building block-based nature provides a conceptual basis, on which future research can build—a basis that is open for extension in respect to new EA management-relevant problems and a broader understanding of the subject of EA management. Possible directions to investigate here, are EA modeling challenges arising in the context of mergers and acquisitions, but also in hybrid value networks of dynamically binding and unbinding enterprises. The patterns of Lau et al. [La09b] pose an interesting starting point here. The understanding of EA management-relevant problems as combination of an abstract goal and a concern provides a more concise way of expressing cross-cutting aspects in EA modeling. The list of such aspects as covered by this thesis and by the building blocks of BEAMS, namely goals, projects, and standards is most likely subject to future extensions. In the thesis we already discussed the importance of modeling responsibilities for EA information as well as access rights thereto. Future research may continue these discussions and establish a comprehensive cross-cutting aspect for rights & responsibilities [Bu10c].

EA models represent the EA or a relevant part thereof according to the particular knowledge of the model creator. The ways in which such models are developed and maintained greatly influences the quality and accuracy of the models' statements. These statements supply the basis on which decisions are taken by the enterprise architects and other involved stakeholders of the organization. Uncertainty in respect to the statements made is another cross-cutting aspect of EA modeling, that deserves in-depth investigations. EA models that allow to express uncertainty about their statements would require richer EA information models and potentially extensions to the meta-language for EA information modeling. A richer meta-language would also be required for comprehensive support for quantitative analyses of the EA. For the definition of an organization-specific indicator system on the EA level, it is necessary to have dedicated meta-language concepts for designating that one indicator is defined based on the values of related architectural properties. Closely related are concepts to describe that two or more architectural properties influence each other. Such causal dependencies, as already discussed by us in [BES08], can be used to understand the interplay between different architectural aspects, especially behavioral ones. Regarding the predictive quality of the analyses also meta-information on the reliability of the architectural models can be supplied. Appropriate meta-language concepts could allow to model uncertainty of architecture properties as well as architectural structures. A possible approach in this direction is outlined by Buckl et al. in [Bu11a].

Furthering the idea of causal dependencies, future research building on the groundwork provided by BEAMS can use these dependencies to simulate behavioral aspects of planned states of the EA. Thereby, not only the building blocks of BEAMS can be employed, but also the theoretical basis, which offers a key conceptualization for separating different facets of EA information modeling. As we discuss in [Bu08d], powerful simulation techniques require certain characteristics of their models, which in turn are often orthogonal to the part of reality, which is targeted by the simulation. The conceptualization of BEAMS supports to separate language aspects from each other, while allowing their flexible combination and consistent integration. In a similar sense, future research can investigate, whether and how re-usable fragments of quantitative and simulation-based analysis can be formulated on the one hand

in a way independent from the actual application scenario and on the other hand in a manner that consistency of integration can be evaluated and ensured. The dependency structures can be a first step into this direction. Based on such structures, it would also be possible to describe socio-technic dependency structures, that reflect dependencies between architectural properties against the motivational dependencies between the different actor roles in an enterprise [BMS11c].

This thesis contributes to the field of EA modeling languages, but especially the research design outlined, can have implications beyond the chosen field. The general method to research practice-proven prescriptions for solving wicked problems based on patterns and a pattern-based understanding of design theories may be applicable to a wide range of other problems in IS research. Its pragmatic nature aiming at early results that can nevertheless be refined in a consistent and systematic way, should be especially appealing in research areas with high industry impact and involvement. Future research on a meta-theoretic level has to refine this method over different applications and should investigate a more formal perspective on both the research steps and outcomes in the sense of design process and design artifact.

Bibliography

[1002] 107th Congress: *Sarbanes-Oxley Act of 2002 (Public Law 107–204)*. `http://www.sec.gov/about/laws/soa2002.pdf` (last accessed 04-16-2011). 2002.

[Ac10] Achenbach, P.: *Eine domänenspezifische Sprache zur Generierung von Sichten der Unternehmensarchitektur*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2010.

[ACB89] Alavi, M.; Carlson, P.; Brooke, G.: *The ecology of MIS research: a twenty year status review*. In *ICIS '89: Proceedings of the tenth international conference on Information Systems*. pages 363–375. New York, NY, USA. 1989. ACM.

[AG10] Aier, S.; Gleichauf, B.: *Application of Enterprise Models for Engineering Enterprise Transformation. Enterprise Modelling and Information System Architectures*. 5:56–72. 2010.

[Ai08a] Aier, S.; Kurpjuweit, S.; Riege, C.; Saat, J.: *Stakeholderorientierte Dokumentation und Analyse der Unternehmensarchitektur*. In (Hegering, H.-G.; Lehmann, A.; Ohlbach, H. J.; Scheideler, C., Ed.): *GI Jahrestagung (2)*. volume 134 of *LNI*. pages 559–565. Bonn, Germany. 2008. Gesellschaft für Informatik.

[Ai08b] Aier, S.; Kurpjuweit, S.; Schmitz, O.; Schulz, J.; Thomas, A.; Winter, R.: *An Engineering Approach to Enterprise Architecture Design and its Application at a Financial Service Provider*. In (Loos, P.; Nüttgens, M.; Turowski, K.; Werth, D., Ed.): *Modellierung betrieblicher Informationssysteme (MobIS 2008)*. pages 115–130. 2008.

[Ai09a] Aier, S.; Buckl, S.; Franke, U.; Gleichauf, B.; Johnson, P.; Närman, P.; Schweda, C. M. et al.: *A Survival Analysis of Application Life Spans based on Enterprise Architecture Models*. In *3rd International Workshop on Enterprise Modelling and Information Systems Architectures*. pages 141–154. Ulm, Germany. 2009.

[Ai09b] Aier, S.; Kurpjuweit, S.; Saat, J.; Winter, R.: *Business Engineering Navigator – A "Business to IT" Approach to Enterprise Architecture Management.* In (Bernard, S.; Doucet, G.; Gøtze, J.; Saha, P., Ed.): *Coherency Management – Architecting the Enterprise for Alignment, Agility, and Assurance.* pages 77–98. Bloomington. 2009. Author House.

[Ai09c] Aier, S.; Kurpjuweit, S.; Saat, J.; Winter, R.: *Enterprise Architecture Design as an Engineering Discipline. AIS Transactions on Enterprise Systems.* 1:36–43. 2009.

[Ai10] Aitken, C.: *EA Management Patterns for Future State Design.* In (Engels, G.; Luckey, M.; Pretschner, A.; Reussner, R., Ed.): *2nd European Workshop on Patterns for Enterprise Architecture Management (PEAM2010).* pages 267–280. Paderborn, Germany. 2010.

[Ak04] van Aken, J. E.: *Management Research Based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules. Journal of Management Studies.* 41(2):219–246. 2004.

[Al72] Alexander, C.: *Notes on the Synthesis of Form.* Harvard University Press. Cambridge, MA, USA. 1972.

[Al77] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I.; Angel, S.: *A Pattern Language.* Oxford University Press. New York, NY, USA. 1977.

[AMT08] Aveiro, D.; Mendes, J. a.; Tribolet, J.: *Organizational modeling with a semantic wiki.* In *Proceedings of the 2008 ACM symposium on Applied computing.* SAC '08. pages 592–593. New York, NY, USA. 2008. ACM.

[An99] Anderson, F.: *A Collection of History Patterns.* In (Harrison, N. B.; Foote, B.; Hans, R., Ed.): *Pattern Languages of Program Design.* Boston, MA, USA. 1999. Addison Wesley.

[An04] Andrade, J.; Ares, J.; García, R.; Pazos, J.; Rodríguez, S.; Silva, A.: *A Methodological Framework for Viewpoint-Oriented Conceptual Modeling. IEEE Trans. Softw. Eng.* 30:282–294. 2004.

[Ar07] Arbab, F.; Boer, F. S. d.; Bonsangue, M. M.; Lankhorst, M. M.; Proper, E. H.; Torre, L. W. N. v. d.: *Integrating Architectural Models - Symbolic, Semantic and Subjective Models in Enterprise Architecture. Enterprise Modelling and Information Systems Architectures.* 2(1):40–57. 2007.

[ARW08a] Aier, S.; Riege, C.; Winter, R.: *Classification of Enterprise Architecture Scenarios – An Exploratory Analysis. Enterprise Modelling and Information Systems Architectures.* 3:14–23. 2008.

[ARW08b] Aier, S.; Riege, C.; Winter, R.: *Unternehmensarchitektur – Literaturüberblick Stand der Praxis. Wirtschaftsinformatik.* 50(4):292–304. 2008.

[AS07] Aier, S.; Schönherr, M.: *Integrating an Enterprise Architecture Using Domain Clustering.* In (Lankhorst, M. M.; Johnson, P., Ed.): *Second Workshop on Trends in Enterprise Architecture Research.* pages 23–30. 2007.

[AS09] Aier, S.; Schelp, J.: *A Reassessment of Enterprise Architecture Implementation.* In *Trends in Enterprise Architecture Research.* pages 53–68. 2009.

[AST10a] Aveiro, D.; Silva, A. R.; Tribolet, J. M.: *Extending the Design and Engineering Methodology for Organizations with the Generation Operationalization and Discontinuation Organization.* In *DESRIST 2010.* pages 226–241. 2010.

[AST10b] Aveiro, D.; Silva, A. R.; Tribolet, J. M.: *Towards a GOD-theory for organizational engineering: continuously modeling the continuous (re)generation, operation and deletion of the enterprise.* In *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing.* pages 150–157. New York, NY, USA. 2010. ACM.

[AT06] ATLAS group at LINA & INRIA: *ATL: Atlas Transformation Language.* `http://www.eclipse.org/gmt/atl/doc/ATL_User_Manual[v0.7].pdf` (last accessed 04-16-2011). 2006.

[Av10] Aveiro, D.: *G.O.D. (Generation, Operationalization & Discontinuation) and Control (sub)organizations: a DEMO-based approache for continuous real-time management of organizational change caused by exceptions.* PhD thesis. Instituto Superior Técnico, Universidade Técnica de Lisboa. Lisbon, PT. 2010.

[AW09] Aier, S.; Winter, R.: *Virtual Decoupling for IT/Business Alignment – Conceptual Foundations, Architecture Design and Implementation Example. Business & Information Systems Engineering.* 1(2):150–163. 2009.

[Ba90] Barley, S. R.: *The Alignment of Technology and Structure through Roles and Networks. Administrative Science Quarterly.* 35(1):61–103. 1990.

[Ba10a] Balke, F.: *Specifying and Classifying Generic Enterprise Model Repository Services.* Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2010.

[Ba10b] Bauer, M. S.: *Ex-post analysis of customizations of an EA management tool.* Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2010.

[BA10c] Burton, B.; Allega, P.: *Hype Cycle for Enterprise Architecture, 2010.* (July):63. 2010.

[BCR94] Basili, V. R.; Caldiera, G.; Rombach, H. D.: *The Goal Question Metric Approach.* Wiley. New York. 1994.

[Be95] Bem, D. J.: *Writing a Review Article for Psychological Bulletin. Psychological Bulletin.* 118(2):172–177. 1995.

[Be03] Becker, J.; Holten, R.; Knackstedt, R.; Niehaves, B.: *Forschungsmethodische Positionierung in der Wirtschaftsinformatik – epistemologische, ontologische und linguistische Leitfragen –.* Technical report. Münster University, Germany. 2003.

[Be05] Bernard, S. A.: *An Introduction to Enterprise Architecture.* Authorhouse. Bloomington, USA. 2nd edition. 2005.

[Be09] Bender, G.: *Designing a Stakeholder-Specific Enterprise Architecture Management based on Patterns.* Master thesis. Fakultät für Informatik, Technische Universität München. 2009.

[BES08] Buckl, S.; Ernst, A. M.; Schweda, C. M.: *An Extension to the Essential Meta-Object Facility (EMOF) for Specifying and Indicating Dependencies between Properties.* Technical report. Chair for Informatics 19 (sebis), Technische Universität München. 2008.

[BES10] Buckl, S.; Ernst, A. M.; Schweda, C. M.: *Teaching Enterprise Architecture Management with student mini-projects.* In (Engels, G.; Luckey, M.; Pretschner, A.; Reussner, R., Ed.): *2nd European Workshop on Patterns for Enterprise Architecture Management (PEAM2010).* pages 309–320. Paderborn, Germany. 2010.

[BG09] Bernard, S. A.; Grasso, J.: *Enterprise Architecture Formalization and Auditing.* In (Doucet, G.; Gøtze, J.; Saha, P.; Bernard, S., Ed.): *Coherency Management – Architecting the Enterprise for Alignment, Agility, and Assurance.* Bloomington, USA. 2009. AuthorHouse.

[BG10] Burger, E.; Gruschko, B.: *A Change Metamodel for the Evolution of MOF-Based Metamodels.* In (Engels, G.; Karagiannis, D.; Mayr, H. C., Ed.): *Modellierung 2010.* pages 285–300. 2010.

[BGS10] Buckl, S.; Gulden, J.; Schweda, C. M.: *Supporting ad hoc Analyses on Enterprise Models.* In *4th International Workshop on Enterprise Modelling and Information Systems Architectures.* 2010.

[BJ87] Brooks Jr., F. P.: *No Silver Bullet – Essence and Accidents of Software Engineering. Computer.* 20:10–19. April 1987.

[BKS10] Buckl, S.; Krell, S.; Schweda, C. M.: *A formal approach to Architectural Descriptions – Refining the ISO Standard 42010.* In (Aalst, W.; Mylopoulos, J.; Sadeh, N. M.; Shaw, M. J.; Szyperski, C.; Albani, A.; Dietz, J. L. G., Ed.): *Advances in Enterprise Engineering IV.* volume 49 of *Lecture Notes in Business Information Processing.* pages 77–91. Springer Berlin Heidelberg. 2010.

[BMN09] Büchner, T.; Matthes, F.; Neubert, C.: *A Concept and Service based Analysis of Commercial and Open Source Enterprise 2.0 Tools.* In (Fred, A.; Dietz, J.; Liu, K.; Filipe, J., Ed.): *KMIS.* pages 37–45. 2009.

[BMN10] Büchner, T.; Matthes, F.; Neubert, C.: *Data Model Driven Implementation of Web Cooperation Systems with Tricia.* In (Dearle, A.; Zicari, R., Ed.): *Objects and Databases.* Lecture Notes in Computer Science. pages 70–84. Springer. Berlin, Heidelberg, Germany. 2010.

[BMS09a] Buckl, S.; Matthes, F.; Schweda, C. M.: *Classifying Enterprise Architecture Analysis Approaches.* In (Poler, R.; van Sinderen, M.; Sanchis, R., Ed.): *Enterprise Interoperability – Second IFIP WG 5.8 International Workshop, IWEI 2009, Valencia, Spain, October 13-14, 2009. Proceedings.* pages 66–79. Berlin, Heidelberg, Germany. 2009. Springer.

[BMS09b] Buckl, S.; Matthes, F.; Schweda, C. M.: *Future Research Topics in Enterprise Architecture Management – A Knowledge Management Perspective.* In (Aier, S.; Schelp, J.; Schönherr, M., Ed.): *4^{th} Workshop on Trends in Enterprise Architecture Research (TEAR 2009).* pages 5–20. 2009.

[BMS09c] Buckl, S.; Matthes, F.; Schweda, C. M.: *A Viable System Perspective on Enterprise Architecture Management.* In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA, 11-14 October 2009.* pages 1483–1488. IEEE. 2009.

[BMS10a] Buckl, S.; Matthes, F.; Schweda, C.: *A situated approach to enterprise architecture management.* In *IEEE International Conference on Systems, Man and Cybernetics (SMC 2010).* pages 587–592. IEEE. 2010.

[BMS10b] Buckl, S.; Matthes, F.; Schweda, C. M.: *Conceptual Models for Cross-cutting Aspects in Enterprise Architecture Modeling.* In *5th International Workshop on Vocabularies, Ontologies, and Rules for the Enterprise (VORTE 2010).* 2010.

[BMS10c] Buckl, S.; Matthes, F.; Schweda, C. M.: *A Design Theory Nexus for Situational Enterprise Architecture Management – Approach and Application Example.* In *14th IEEE International EDOC Conference (EDOC 2010).* Vitoria, Brazil. 2010.

[BMS10d] Buckl, S.; Matthes, F.; Schweda, C. M.: *From EA management patterns towards a prescriptive theory for designing enterprise-specific EA management functions – Outline of a research stream.* In (Schumann, M.; Kolbe, L. M.; Breitner, M. H.; Freirichs, A., Ed.): *Multikonferenz Wirtschaftsinformatik (MKWI2010).* pages 67–78. Göttingen, Germany. 2010.

[BMS10e] Buckl, S.; Matthes, F.; Schweda, C. M.: *Future Research Topic in Enterprise Architecture Management – A Knowledge Management Perspective. Journal of Enterprise Architecture (JEA).* 6(3):16–27. 2010.

[BMS10f] Buckl, S.; Matthes, F.; Schweda, C. M.: *Interrelating Concerns in EA Documentation – Towards a Conceptual Framework of Relationships.* In (Engels, G.; Luckey, M.; Pretschner, A.; Reussner, R., Ed.): *2nd European Workshop on Patterns for Enterprise Architecture Management (PEAM2010).* pages 243–252. Paderborn, Germany. 2010.

[BMS10g] Buckl, S.; Matthes, F.; Schweda, C. M.: *A Meta-Language for EA Information Modeling – State-of-the-art and Requirements Elicitation.* In (Bider, I.; Halpin, T.; Krogstie, J.; Nurcan, S.; Proper, E.; Schmidt, R.; Ukor, R., Ed.): *Enterprise, Business-Process and Information Systems Modeling.* Lecture Notes in Business Information Systems. pages 169–181. Springer. 2010.

[BMS10h] Buckl, S.; Matthes, F.; Schweda, C. M.: *A Modeling Language for Describing EA Management Methods.* In (Esswein, W.; Turowski, K.; Juhrisch, M., Ed.): *Modellierung betrieblicher Informationssysteme (MobIS2010) – Modellgestütztes Management, 15.-17.September 2010, Dresden.* Bonn, Germany. 2010. Gesellschaft für Informatik (GI).

[BMS10i] Buckl, S.; Matthes, F.; Schweda, C. M.: *A technique for Annotating EA Information Models.* In (Barjis, J., Ed.): *Enterprise and Organizational Modeling and Simulation: 6th International Workshop, EOMAS 2010 held at CAiSE 2010, Hammamet, Tunisia, June 2010 Selected Papers.* Lecture Notes in Business Information Systems. pages 113–127. Berlin, Heidelberg, Germany. 2010. Springer.

[BMS10j] Buckl, S.; Matthes, F.; Schweda, C. M.: *Towards a Method Framework for Enterprise Architecture Management – A Literature Analysis from a Viable System Perspective.* In *5th International Workshop on Business/IT Alignment and Interoperability (BUSITAL 2010).* 2010.

[BMS10k] Buckl, S.; Matthes, F.; Schweda, C. M.: *Utilizing Patterns in Developing Design Theories.* In *2010 International Conference on Information Systems (ICIS 2010).* 2010.

[BMS10l] Buckl, S.; Matthes, F.; Schweda, C. M.: *Vergangenheit, Gegenwart und Zukunft von EAM.* In (Keuntje, J. H.; Barkow, R., Ed.): *Enterprise Architecture Management in der Praxis – Wandel, Komplexität und IT-Kosten im Unternehmen beherrschen.* pages 377–415. Düsseldorf, Germany. 2010. Symposium.

[BMS11a] Buckl, S.; Matthes, F.; Schweda, C. M.: *A Method Base for Enterprise Architecture Management.* In (Ralyte, J.; Mirbel, I.; Deneckere, R., Ed.): *IFIP Working Conference on Method Engineering (ME 2011).* Paris, France. 2011. Springer.

[BMS11b] Buckl, S.; Matthes, F.; Schweda, C. M.: *A method to develop EA modeling languages using practice-proven solutions.* In *1st Enterprise Engineering Working Conference, EEWC2011.* 2011.

[BMS11c] Buckl, S.; Matthes, F.; Schweda, C. M.: *Socio-technic Dependency and Rationale Models for the Enterprise Architecture Management Function.* In (Pastor, O.; Salinesi, C., Ed.): *Ontology, Conceptualization and Epistemology for Information Systems, Software Engineering and Service Science: 5th International Workshop, ONTOSE 2011.* Lecture Notes in Business Information Systems. Berlin, Heidelberg, Germany. 2011. Springer.

[BN94] Bernus, P.; Nemes, L.: *A Framework to Define Generic Enterprise Reference Architecture and Methodology.* In *Proceedings of the International Conference on Automation, Robotics and Computer Vision (ICARCV'94), Singapore, November 10–12.* pages 88–92. 1994.

[BN96] Bernus, P.; Nemes, L.: *A Framework to Define Generic Enterprise Reference Architecture and Methodology. Proceedings of the International Conference on Automation, Robotics and Computer Vision (ICARCV'94), Singapore, November 10–12.* 9:179–191. 1996.

[BN07] Becker, J.; Niehaves, B.: *Epistemological perspectives on IS research: a framework for analysing and systematizing epistemological assumptions. Information Systems Journal.* 17:197–214. 2007.

[BNS03] Bernus, P.; Nemes, L.; Schmidt, G.: *Handbook on Enterprise Architecture.* Springer. Berlin, Heidelberg, Germany. 2003.

[Br07] Braun, C.: *Modellierung der Unternehmensarchitektur – Weiterentwicklung einer bestehenden Methode und deren Abbildung in einem Meta-Modellierungswerkzeug.* PhD thesis. Universität St.Gallen. 2007.

[Br10] Bruls, W. A. G.; Steenbergen, M. v.; Foorthuis, R. M.; Boos, R.; Brinkkemper, S.: *Domain Architectures as an Instrument to Refine Enterprise Architecture. Communications of the Associations for Information Systems.* 27:517–540. 2010.

[BRS95] Becker, J.; Rosemann, M.; Schütte, R.: *Grundsätze ordnungsmäßiger Modellierung. Wirtschaftsinformatik.* 37(5):435–445. 1995.

[BS06] van Berg, M. d.; van Steenbergen, M.: *Building an Enterprise Architecture Practice – Tools, Tips, Best Practices, Ready-to-Use Insights.* Springer. Dordrecht, The Netherlands. 2006.

[BS11] Buckl, S.; Schweda, C. M.: *On the state-of-the-art in EA management literature.* Technical report. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2011.

[Bu96] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.: *Pattern-oriented software architecture: a system of patterns.* John Wiley & Sons, Inc. New York, NY, USA. 1996.

[Bu07a] Buckl, S.; Ernst, A. M.; Lankes, J.; Matthes, F.; Schweda, C. M.; Wittenburg, A.: *Generating Visualizations of Enterprise Architectures using Model Transformation (extended version). Enterprise Modelling and Information Systems Architectures – An International Journal.* 2(2):3–13. 2007.

[Bu07b] Buckl, S.; Ernst, A. M.; Lankes, J.; Schneider, K.; Schweda, C. M.: *A pattern based Approach for constructing Enterprise Architecture Management Information Models.* In (Oberweis, A.; Weinhardt, C.; Gimpel, H.; Koschmider, A.; Pankratius, V.; Schnizler, Ed.): *Wirtschaftsinformatik 2007.* pages 145–162. Karlsruhe, Germany. 2007. Universitätsverlag Karlsruhe.

[Bu08a] Buckl, S.; Dierl, T.; Matthes, F.; Ramacher, R.; Schweda, C. M.: *Current and Future Tool Support for EA Management.* In (Steffens, U.; Addicks, J.; Streekmann, N., Ed.): *MDD, SOA und IT-Management (MSI 2008).* Berlin, Germany. 2008. GITO-Verlag.

[Bu08b] Buckl, S.; Ernst, A. M.; Lankes, J.; Matthes, F.: *Enterprise Architecture Management Pattern Catalog (Version 1.0, February 2008).* Technical report. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2008.

[Bu08c] Buckl, S.; Ernst, A. M.; Matthes, F.; Schweda, C. M.: *An Information Model for Landscape Management – Discussing Temporality Aspects.* In (Johnson, P.; Schelp, J.; Aier, S., Ed.): *Pre-Proceedings of the 3rd Workshop on Trends in Enterprise Architecture Research.* pages 63–77. Sydney, Australia. 2008.

[Bu08d] Buckl, S.; Matthes, F.; Renz, W.; Schweda, C. M.; Sudeikat, J.: *Towards simulation-supported enterprise architecture management.* In (Loos, P.; Nüttgens, M.; Turowski, K.; Werth, D., Ed.): *Modellierung betrieblicher Informationssysteme (MobIS2008).* pages 131–145. 2008.

[Bu09a] Buckl, S.; Ernst, A. M.; Lankes, J.; Matthes, F.; Schweda, C. M.: *State of the Art in Enterprise Architecture Management 2009.* Technical report. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2009.

[Bu09b] Buckl, S.; Ernst, A. M.; Matthes, F.; Ramacher, R.; Schweda, C. M.: *Using Enterprise Architecture Management Patterns to complement TOGAF.* In *13th IEEE International EDOC Conference (EDOC 2009).* Auckland, New Zealand. 2009. IEEE Computer Society.

[Bu09c] Buckl, S.; Ernst, A. M.; Matthes, F.; Schweda, C. M.: *Enterprise Architecture Management Pattern for EA Visioning.* In *Proceedings of EuroPLoP 2009.* 2009.

[Bu09d] Buckl, S.; Ernst, A. M.; Matthes, F.; Schweda, C. M.: *How to make your enterprise architecture management endeavor fail!* In *Pattern Languages of Programs 2009 (PLoP 2009), Chicago.* 2009.

[Bu09e] Buckl, S.; Ernst, A. M.; Matthes, F.; Schweda, C. M.: *An Information Model for Managed Application Landscape Evolution. Journal of Enterprise Architecture (JEA).* 5(1):12–26. 2009.

[Bu09f] Buckl, S.; Ernst, A. M.; Matthes, F.; Schweda, C. M.: *Visual Roadmaps for Enterprise Architecture Evolution.* In *The 1st International Workshop on Enterprise Architecture Challenges and Responses.* Korea. 2009.

[Bu09g] Buckl, S.; Franke, U.; Holschke, O.; Matthes, F.; Schweda, C. M.; Sommestad, T.; Ullberg, J.: *A Pattern-based Approach to Quantitative Enterprise Architecture Analysis.* In *15th Americas Conference on Information Systems (AMCIS).* San Francisco, CA, USA. 2009.

[Bu09h] Buckl, S.; Matthes, F.; Neubert, C.; Schweda, C. M.: *A Wiki-based Approach to Enterprise Architecture Documentation and Analysis.* In *The 17th European Conference on Information Systems (ECIS) – Information Systems in a Globalizing World: Challenges, Ethics and Practices, 8.–10. June 2009, Verona, Italy.* pages 2192–2204. Verona, Italy. 2009.

[Bu09i] Buckl, S.; Matthes, F.; Schulz, C.; Schweda, C. M.: *Teaching Enterprise Architecture Management – A Practical Experience.* Technical report. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2009.

[Bu10a] Buckl, S.; Dierl, T.; Matthes, F.; Schweda, C. M.: *Building Blocks for Enterprise Architecture Management Solutions.* In (Harmsen, F. e. a., Ed.): *Practice-Driven Research on Enterprise Transformation, second working conference, PRET 2010, Delft.* pages 17–46. Berlin, Heidelberg, Germany. 2010. Springer.

[Bu10b] Buckl, S.; Matthes, F.; Roth, S.; Schulz, C.; Schweda, C. M.: *A Conceptual Framework for Enterprise Architecture Design.* In (Aalst, W.; Mylopoulos, J.; Sadeh, N. M.; Shaw, M. J.; Szyperski, C.; Proper, E.; Lankhorst, M. M. et al., Ed.): *Trends in Enterprise Architecture Research.* volume 70 of *Lecture Notes in Business Information Processing.* pages 44–56. Springer Berlin Heidelberg. 2010.

[Bu10c] Buckl, S.; Matthes, F.; Roth, S.; Schulz, C.; Schweda, C. M.: *A Method for Constructing Enterprise-wide Access Views on Business Objects.* In (Fähnrich, K.-P.; Franczyk, B., Ed.): *GI Jahrestagung (2).* volume 176 of *LNI.* pages 279–284. GI. 2010.

[Bu10d] Buckl, S.; Matthes, F.; Schulz, C.; Schweda, C. M.: *Exemplifying a Framework for Interrelating Enterprise Architecture Concerns.* In (Sicilia, M.-A.; Kop, C.; Sartori, F., Ed.): *Ontology, Conceptualization and Epistemology for Information Systems, Software Engineering and Service Science: 4^{th} International Workshop, ONTOSE 2010, held at CAiSE 2010 Hammamet, Tunisia, June 2010, Revised Selected Papers.* Lecture Notes in Business Information Systems. pages 33–46. Berlin, Heidelberg, Germany. 2010. Springer.

[Bu11a] Buckl, S.; Buschle, M.; Johnson, P.; Matthes, F.; Schweda, C. M.: *A meta-language for enterprise architecture analysis.* In (Bider, I.; Halpin, T.; Krogstie, J.; Nurcan, S.; Proper, E.; Schmidt, R.; Soffer, P., Ed.): *16th International Conference on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD 2011).* Lecture Notes in Business Information Systems. Berlin, Heidelberg, Germany. 2011. Springer.

[Bu11b] Buckl, S.; Dierl, T.; Matthes, F.; Schweda, C. M.: *Complementing The Open Group Architecture Framework with Best Practice Solution Building Blocks.* In *44^{th} Hawaii International Conference on System Sciences (HICSS 2011).* Kauai, Hawaii, USA. 2011.

[Bu11c] Buckl, S.; Matthes, F.; Monahov, I.; Schweda, C. M.: *Modeling Enterprise Architecture Transformations.* In *3^{rd} International IFIP WG5.8 Working Conference on Enterprise Interoperability (IWEI 2011).* Stockholm, Sweden. 2011.

[Bu11d] Buckl, S.; Matthes, F.; Neubert, C.; Schweda, C. M.: *A Lightweight Approach to Enterprise Architecture Modeling and Documentation.* In (Soffer, P.; Proper, E., Ed.): *Information Systems Evolution – CAiSE Forum 2010, Hammamet, Tunisia, June 2010, Selected Extended Papers.* pages 136–149. Berlin, Heidelberg, Germany. 2011. Springer.

[Bu11e] Buckl, S. M.: *Developing Organization-Specific Enterprise Architecture Management Functions Using a Method Base.* PhD thesis. Technische Universität München. München, Germany. 2011.

[BW06] Balabko, P.; Wegmann, A.: *Systemic classification of concern-based design methods in the context of enterprise architecture. Information Systems Frontiers.* 8(2):115–131. 2006.

[Ca07] Carlsson, S. A.: *Developing Knowledge Through IS Design Science Research: For Whom, What Type of Knowledge, and How? Scandinavian J. Inf. Systems.* 19(2):75 – 86. 2007.

[CEF99] Carlson, A.; Estepp, S.; Fowler, M.: *Temporal Patterns.* In (Harrison, N. B.; Foote, B.; Hans, R., Ed.): *Pattern Languages of Program Design.* Boston, MA, USA. 1999. Addison Wesley.

[Ch76] Chen, P. P.-S.: *The Entity-Relationship Model – Toward a Unified View of Data. ACM Transactions on Database Systems (TODS).* 1(1):9–36. 1976.

[CH04] Coplien, J. O.; Harrison, N. B.: *Organizational Patterns of Agile Software Development.* Prentice Hall PTR. Indianapolis, IN, USA. 2004.

[Ch10] Chair for Informatics 19 (sebis),Technische Universität München: *EAM Pattern Catalog Wiki.* `http://eampc-wiki.systemcartography.info` (last accessed 04-16-2011). 2010.

[CI04] CIMOSA Association: *CIM – Open System Architecture.* `http://www.pera.net/Methodologies/Cimosa/CIMOSA.html` (last accessed 04-16-2011). 2004.

[Co96] Coplien, J. O.: *Software Patterns: Management Briefs.* Cambridge University Press. Cambridge, UK. 1996. 188484250X.

[CST09] Caetano, A.; Silva, A. R.; Tribolet, J.: *A role-based enterprise architecture framework.* In *Proceedings of the 2009 ACM symposium on Applied Computing.* SAC '09. pages 253–258. New York, NY, USA. 2009. ACM.

[CST10] Caetano, A.; Silva, A. R.; Tribolet, J.: *Business Process Decomposition – An Approach Based on the Principle of Separation of Concerns. Enterprise Modeling and Information Systems Architectures.* 5(1):44–58. 2010.

[Cu42] Curry, H. B.: *The Inconsistency of Certain Formal Logic. The Journal of Symbolic Logic.* 7(3):115–117. 1942.

[Da00] Date, C. J.: *An introduction to database systems.* Addison-Wesley-Longman. Reading, Menlo Park, New York. 7$^{\text{th}}$ edition. 2000.

[DD00] Date, C.; Darwen, H.: *Foundation for Future Database Systems.* Addison Wesley. Reading, MA, USA. 2$^{\text{nd}}$ edition. 2000.

[De82] Deming, E. W.: *Out of the Crisis.* Massachusetts Institute of Technology. Cambridge, MA, USA. 1982.

[De06] Dern, G.: *Management von IT-Architekturen (Edition CIO).* Vieweg. Wiesbaden, Germany. 2006.

[De09a] Department of Defense (DoD) USA: *DoD Architecture Framework Version 2.0: Volume 1: Introduction, Overview, and Concepts – Manager's Guide.* `http://www.defenselink.mil/cio-nii/docs/DoDAF%20V2%20-%20Volume%201.pdf` (last accessed 04-16-2011). 2009.

[De09b] Department of Defense (DoD) USA: *DoD Architecture Framework Version 2.0: Volume 2: Architectural Data and Models – Architect's Guide.* `http://www.defenselink.mil/cio-nii/docs/DoDAF%20V2%20-%20Volume%202.pdf` (last accessed 04-16-2011). 2009.

[Di05] Dietz, J. L.: *A World Ontology Specification Language.* In *On the Move to Meaningful Internet Systems 2005: OTM Workshops.* pages 688–699. 2005.

[Di06a] Dietz, J. L.: *Enterprise Ontology.* Springer. Heidelberg, Germany. 2006.

[Di06b] Dijkman, R. M.: *Consistency in multi-viewpoint architectural design.* PhD thesis. Enschede. 2006.

[Di10] Dierl, T.: *Design and prototypic implementation of a web-based data and process model configurator.* Master's thesis. Fakultät für Informatik, Technische Universität München. 2010.

[Do08] Doucet, G.; Gøtze, J.; Saha, P.; Bernard, S. A.: *Coherency Management: Using Enterprise Architecture for Alignment, Agility, and Assurance. Journal on Enterprise Architecture.* 4(2). 2008.

[Do09a] Doucet, G.; Gøtze, J.; Saha, P.; Bernard, S. A.: *Coherency Management – Architecting the Enterprise for Alignment, Agility, and Assurance.* AuthorHouse. Bloomington, USA. 2009.

[Do09b] Doucet, G.; Gøtze, J.; Saha, P.; Bernard, S. A.: *Commencing the Journey: Realizing Coherency Management.* In (Doucet, G.; Gøtze, J.; Saha, P.; Bernard, S., Ed.): *Coherency Management – Architecting the Enterprise for Alignment, Agility, and Assurance.* Bloomington, USA. 2009. AuthorHouse.

[Do09c] Doucet, G.; Gøtze, J.; Saha, P.; Bernard, S. A.: *Introduction to Coherency Management: The Transformation of Enterprise Architecture.* In (Doucet, G.; Gøtze, J.; Saha, P.; Bernard, S., Ed.): *Coherency Management – Architecting the Enterprise for Alignment, Agility, and Assurance.* Bloomington, USA. 2009. AuthorHouse.

[DOW08a] Decker, G.; Overdick, H.; Weske, M.: *Oryx – An Open Modeling Platform for the BPM Community.* In (Dumas, M.; Reichert, M.; Shan, M.-C., Ed.): *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.

[DOW08b] Decker, G.; Overdick, H.; Weske, M.: *Oryx – Sharing Conceptual Models on the Web.* In *Demo Session of the 27th International Conference on Conceptual Modeling (ER).* pages 536–537. Barcelona, Spain. 2008. LNCS 5231.

[DQS08] Dijkman, R. M.; Quartel, D. A.; van Sinderen, M. J.: *Consistency in multiviewpoint design of enterprise information systems. Information and Software Technology.* 50(7–8):737 – 752. 2008.

[DV02] Domokos, P.; Varró,Dániel: *An Open Visualization Framework for Metamodel-Based Modeling Languages. Electronic Notes in Theoretical Computer Science.* 72(2). 2002.

[ED09] Ettema, R.; Dietz, J. L.: *ArchiMate and DEMO – Mates to Date?* In *Advances in Enterprise Engineering III.* pages 172–186. Albani, Antonia and Barjis, Joseph and Dietz, Jan L.G. 2009.

[EDO06] *1st Workshop on Trends in Enterprise Architecture Research (TEAR 2006).* Washington, DC, USA. 2006. IEEE Computer Society.

[EG07] Eisenhardt, K. M.; Graebner, M. E.: *Theory building from cases: Opportunities and challenges. Academy of Management Journal.* 50(1):25–32. 2007.

[Ek04] Ekstedt, M.; Johnson, P.; Lindström, Å.; Gammelgård, M.; Johansson, E.; Plazaola, L.; Silva, E. et al.: *Consistent Enterprise Software System Architecture for the CIO – A Utility-Cost Based Approach.* In *37th Hawaii International Conference on System Sciences (HICSS 2004).* 2004.

[Ek09] Ekstedt, M.; Franke, U.; Johnson, P.; Lagerström, R.; Sommestad, T.; Ullberg, J.; Buschle, M.: *A Tool for Enterprise Architecture Analysis of Maintainability.* In (Winter, A.; Ferenc, R.; Knodel, J., Ed.): *13th European Conference on Software Maintenance and Reengineering, CSMR 2009.* pages 327–328. Kaiserslautern, Germany. 2009.

[Er06a] Ernst, A. M.; Lankes, J.; Schweda, C. M.; Wittenburg, A.: *Tool Support for Enterprise Architecture Management – Strengths and Weaknesses.* In *10^{th} IEEE International EDOC Conference (EDOC 2006).* pages 13–22. Washington, DC, USA. 2006. IEEE Computer Society.

[Er06b] Ernst, A. M.; Lankes, J.; Schweda, C. M.; Wittenburg, A.: *Using Model Transformation for Generating Visualizations from Repository Contents – An Application to Software Cartography.* Technical report. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2006.

[Er08] Ernst, A. M.: *Enterprise Architecture Management Patterns.* In *PLoP 08: Proceedings of the Pattern Languages of Programs Conference 2008.* Nashville, USA. 2008.

[Er10] Ernst, A. M.: *A Pattern-Based Approach to Enterprise Architecture Management.* PhD thesis. Technische Universität München. München, Germany. 2010.

[Fa98] Falkenberg, E. D.; Hesse, W.; Lindgreen, P.; Nilsson, B. E.; Oei, J. L. H.; Rolland, C.; Stamper, R. K. et al.: *A Framework of Information Systems Concepts.* 1998.

[Fe94] Ferstl, O. K.; Sinz, E. J.; Amberg, M.; Hagemann, U.; Malischewski, C.: *Tool-Based Business Process Modeling Using the SOM Approach.* In *GI Jahrestagung.* pages 430–436. 1994.

[Fe06] Fettke, P.: *State-of-the-Art des State-of-the-Art – Eine Untersuchung der Forschungsmethode "Review" innerhalb der Wirtschaftsinformatik. WIRTSCHAFTSINFORMATIK.* 48(74):257–266. 2006.

[Fi08] Fischer, R.: *Organisation der Unternehmensarchitektur – Entwicklung der aufbau- und ablauforganisatorischen Strukturen unter besonderer Berücksichtigung des Gestaltungsziels Konsistenzerhaltung.* PhD thesis. Universität St.Gallen. 2008.

[Fo96] von Foerster, H.: *Wissen und Gewissen – Versuch einer Brücke.* Suhrkamp. Frankfurt a. M., Germany. 1996.

[Fr94] Frank, U.: *MEMO: A Tool Supported Methodology for Analyzing and (Re-)Designing Business Information Systems.* In (Ege, R.; Singh, M.; Meyer, B., Ed.): *Technology of Object-Oriented Languages and Systems.* pages 367–380. 1994.

[Fr98a] Frank, U.: *The MEMO Meta-Metamodel.* Technical Report 9. Arbeitsberichte des Instituts für Wirtschaftsinformatik Koblenz. 1998.

[Fr98b] Frank, U.: *The MEMO Object Modelling Language (MEMO-OML).* Technical Report 10. Arbeitsberichte des Instituts für Wirtschaftsinformatik Koblenz. 1998.

[Fr99] Frank, U.: *Memo: Visual Languages for Enterprise Modelling.* Technical Report 18. Arbeitsberichte des Instituts für Wirtschaftsinformatik Koblenz. 1999.

[Fr08] Frank, U.; Heise, D.; Kattenstroth, H.; Schauer, H.: *Designing and Utilising Business Indicator Systems within Enterprise Models – Outline of a Method.* In *Modellierung betrieblicher Informationssysteme (MobIS 2008) – Modellierung zwischen SOA und Compliance Management 27.-28. November 2008.* Saarbrücken, Germany. 2008.

[Fr09] Frank, U.: *The MEMO Meta Modelling Language (MML) and Language Architecture (ICB-Research Report).* Technical report. Institut für Informatik und Wirtschaftsinformatik. Duisburg-Essen, Germany. 2009.

[FS95] Ferstl, O. K.; Sinz, E. J.: *Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. Wirtschaftsinformatik.* 37(3):209–220. 1995.

[FS97] Ferstl, O. K.; Sinz, E. J.: *Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework.* pages 339–358. Springer. 1997.

[FS98] Ferstl, O. K.; Sinz, E. J.: *Grundlagen der Wirtschaftsinformatik.* Oldenbourg. 5th edition. 1998.

[Ga75] Gadamer, H.-G.: *Wahrheit und Methode – Grundzüge einer philosophischen Hermeneutik.* J.C.B. Mohr. Tübingen, Germany. 3rd edition. 1975.

[Ga92] Gamma, E.: *Object-Oriented Software Development based on ET++.* PhD thesis. Institut für Informatik, University of Zurich. 1992.

[Ga94] Gamma, E.; Helm, R.; Johnson, R. E.; Vlissides, J. M.: *Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series).* Addison-Wesley Professional. Munich, Germany. 1994.

[Ge07] Getoor, L.; Friedman, N.; Koller, D.; Pfeffer, A.; Taskar, B.: *Probabilistic Relational Models.* In (Getoor, L.; Taskar, B., Ed.): *An Introduction to Statistical Relational Learning.* MIT Press. Cambridge, MA, USA. 2007.

[Ge09] Gehlert, A.; Schermann, M.; Pohl, K.; Krcmar, H.: *Towards a Research Method for Theory-driven Design Research.* In (Hansen, H. R.; Karagiannis, D.; Fill, H. G., Ed.): *Business Services: Konzepte, Technologien, Anwendungen, 9. Internationale Tagung Wirtschaftsinformatik.* volume 1. pages 441–450. Wien, Austria. 2009. Österreichische Computer Gesellschaft.

[GJ07] Gregor, S.; Jones, D.: *The anatomy of a design theory. Journal of the Association of Information Systems.* 8(5):312–335. 2007.

[GLS06] Gammelgård, M.; Lindström, Å.; Simonsson, M.: *A reference model for IT management responsibilities.* In *10th IEEE International EDOC Conference Workshops (EDOCW).* pages 26–33. Washington, DC, USA. 2006. IEEE Computer Society.

[Gr06] Gregor, S.: *The nature of theory in information systems. MIS Quarterly.* 30(3):491–506. 2006.

[Gu05] Guizzardi, G.: *Ontological foundations for structural conceptual models.* PhD thesis. CTIT, Centre for Telematics and Information Technology. Enschede, The Netherlands. 2005.

[GW00a] Guarino, N.; Welty, C. A.: *A Formal Ontology of Properties.* In *12th International Conference on Knowledge Acquisition, Modeling and Management.* pages 97–112. Juan-les-Pins, France. 2000. Springer.

[GW00b] Guarino, N.; Welty, C. A.: *Identity, Unity, and Individuality: Towards a Formal Toolkit for Ontological Analysis.* In (Horn, W., Ed.): *Proceedings of the 14th European Conference on Artificial Intelligence.* pages 219–223. Berlin, Germany. 2000. IOS Press.

[Ha05] Halpin, T. A.: *ORM 2.* In *OTM Workshops.* pages 676–687. 2005.

[Ha09] Halpin, T.: *Object-Role modeling.* pages Liu,Ling AND öszu,Tamer M. Springer. 2009.

[Ha10] Hanschke, I.: *Strategic IT Management – A Toolkit for Enterprise Architecture Management.* Springer. Berlin, Germany. 2010.

[He04] Hevner, A. R.; March, S. T.; Park, J.; Ram, S.: *Design Science in Information Systems Research. MIS Quarterly.* 28(1):75–105. 2004.

[He08] Heise, D.; Strecker, S.; Frank, U.; Jung, J.: *Erweiterung einer Unternehmensmodellierungsmethode zur Unterstützung des IT-Controllings.* In (Bichler, M.; Hess, T.; Krcmar, H.; Lechner, U.; Matthes, F.; Picot, A.; Speitkamp, B. et al., Ed.): *Multikonferenz Wirtschaftsinformatik.* GITO-Verlag, Berlin. 2008.

[HKL95] Hirschheim, R.; Klein, H. K.; Lyytinen, K.: *Information Systems Development and Data Modeling – Conceptual and Philosophical Foundations.* Cambridge University Press. Cambridge, UK. 1995.

[HP04] Hirvonen, A. P.; Pulkkinen, M.: *A Practical Approach to EA Planning and Development: the EA Management Grid.* In (Abramowicz, W., Ed.): *Proceedings of 7th International Conference on Business Information Systems.* pages 284–302. Poznan, PL. 2004.

[HR00] Harel, D.; Rumpe, B.: *Modeling Languages: Syntax, Semantics and All That Stuff.* Technical report. The Weizmann Institute of Science. Rehovot, Isreal. 2000.

[HV93] Henderson, J. C.; Venkatraman, N.: *Strategic alignment: leveraging information technology for transforming organizations. IBM Systems Journal.* 32(1):472–484. 1993.

[HW08] Hafner, M.; Winter, R.: *Processes for Enterprise Application Architecture Management.* In *41st Hawaii International Conference on System Sciences (HICSS 2008).* page 396. Los Alamitos, CA, USA. 2008. IEEE Computer Society.

[IE00] IEEE: *IEEE Std 1471-2000 for Recommended Practice for Architectural Description of Software-Intensive Systems.* 2000.

[IF99] IFIP-IFAC Task Force on Architecture for Enterprise Integration: *Geram: Generalised Enterprise Reference Architecture and Methodology – version 1.6.3.* 1999.

[IF03] IFIP-IFAC Task Force on Architecture for Enterprise Integration: *GERAM: The Generalised Enterprise Reference Architecture and Methodology.* In (Bernus, P.; Nemes, L.; Schmidt, G., Ed.): *Handbook on Enterprise Architecture.* pages 21–63. Berlin, Heidelberg, Germany. 2003. Springer.

[IL08] Isomäki, H.; Liimatainen, K.: *Challenges of Government Enterprise Architecture Work – Stakeholders' Views.* In (Wimmer, M.; Scholl, H. J.; Ferro, E., Ed.): *Electronic Government, 7th International Conference.* pages 364–374. Turin, Italy. 2008. Springer.

[In96] International Organization for Standardization: *ISO/IEC 10746 Reference Model of Open Distributed Processing (RM-ODP).* 1996.

[In99] International Organization for Standardization: *ISO 15704 (1999) ISO/DIS 15704: Industrial automation systems – Requirements for enterprise reference architectures and methodologies. ISO/TC 184/SC5/WG1.* 1999.

[In06] International Organization for Standardization: *ISO 19439:2006 Enterprise integration – Framework for enterprise modelling.* 2006.

[In07] International Organization for Standardization: *ISO/IEC 42010:2007 Systems and software engineering – Recommended practice for architectural description of software-intensive systems.* 2007.

[In10] InfoAsset: *Tricia – Open Source Web Collaboration and Knowledge Management Software.* `http://www.infoasset.de/wikis/infoasset/home` (last accessed 04-16-2011). 2010.

[JE07] Johnson, P.; Ekstedt, M.: *Enterprise Architecture – Models and Analyses for Information Systems Decision Making.* Studentlitteratur. Pozkal, Poland. 2007.

[JNL06] Johnson, P.; Nordström, L.; Lagerström, R.: *Formalizing Analysis of Enterprise Architecture.* In *Enterprise Interoperability.* pages 35–44. Bordeaux, France. 2006. Springer.

[Jo03] Jonkers, H.; van Burren, R.; Arbab, F.; de Boer, F.; Bonsangue, M. M.; Bosma, H.; ter Doest, H. et al.: *Towards a language for coherent enterprise architecture descriptions.* In *7^{th} IEEE International EDOC Conference (EDOC 2003).* Brisbane, Australia. 2003. IEEE Computer Society.

[Jo04a] Johnson, P.; Ekstedt, M.; Silva, E.; Plazola, L.: *Enterprise Architecture for CIO Decision-Making: On the Importance of Theory.* In *Proceedings of the Second Annual Conference on Systems Engineering Research.* pages 1–10. 2004.

[Jo04b] Jonkers, H.; Lankhorst, M. M.; Buuren, R. v.; Bonsangue, M. M.; Torre, L. W. N. v. d.: *Concepts for Modelling Enterprise Architectures. International Journal of Cooperative Information Systems.* 13:257–287. 2004.

[Jo06] Jonkers, H.; Lankhorst, M. M.; ter Doest, H.; Arbab, F.; Bosma, H.; Wieringa, R.: *Enterprise architecture: Management tool and blueprint for the organisation. Information Systems Frontiers.* 8:63–66. 2006.

[Jo07] Johnson, P.; Lagerström, R.; Närman, P.; Simonsson, M.: *Enterprise architecture analysis with extended influence diagrams. Information Systems Frontiers.* 9:163–180. 2007.

[Jo09] Josey, A. e. a.: *TOGAF Version 9 – A Pocket Guide.* Van Haren Publishing. Wilco, Amersfoort, Netherlands. 2^{nd} edition. 2009.

[Jo10] Jonkers, H.; van den Berg, H.; Iacob, M.-E.; Quartel, D.: *ArchiMate Extension for Modeling TOGAF's Implementation and Migration phases.* `http://www.bizzdesign.com/index.php/component/docman/doc_download/18-archimate-extension-for-modeling-togafs-implementation-and-migration-phases` (last accessed 04-16-2011). 2010.

[Ju07] Jung, J.: *Entwurf einer Sprache für die Modellierung von Ressourcen im Kontext der Geschäftsprozessmodellierung.* PhD thesis. Universität Duisburg-Essen. Berlin, Germany. 2007.

[Ka08] Kant, I.: *Critique of Pure Reason.* Penguin Classics. New York, NY, USA. 2008.

[Ke07] Keller, W.: *IT-Unternehmensarchitektur.* dpunkt.verlag. Heidelberg, Germany. 2007.

[Ki99] Kirch, J.: *Prozessorientiertes Management von Client-Server-Systemen.* PhD thesis. University of Wiesbaden. 1999.

[Ki08] Kirchner, L.: *Eine Methode zur Unterstützung des IT-Managements im Rahmen der Unternehmensmodellierung.* PhD thesis. Universität Duisburg-Essen. Berlin, Germany. 2008.

[KL67] Kamlah, W.; Lorenzen, P.: *Logische Propädeutik: Vorschule des vernünftigen Redens.* Metzler. Stuttgart, Germany. 2nd edition. 1967.

[Kl99] Klar, M.: *A Semantical Framework for the Integration of Object-oriented Modeling Languages.* PhD thesis. Technische Universität Berlin. 1999.

[KNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: *Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)".* Technical Report Heft 89. Institut für Wirtschaftsinformatik (IWi), Universität des Saarlandes. 1992.

[Kr02] Krogstie, J.: *A Semiotic Approach to Quality in Requirements Specifications.* In *Proceedings of the IFIP TC8 / WG8.1 Working Conference on Organizational Semiotics: Evolving a Science of Information Systems.* pages 231–249. Deventer, The Netherlands, The Netherlands. 2002. Kluwer, B.V.

[Kr09] Krogmann, K.; Schweda, C. M.; Buckl, S.; Kuperberg, M.; Martens, A.; Matthes, F.: *Improved Feedback for Architectural Performance Prediction Using Software Cartography Visualizations.* In *Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems.* QoSA '09. pages 52–69. Berlin, Germany. 2009. Springer-Verlag.

[Kü04] Kühn, H.: *Methodenintegration im Business Engineering.* PhD thesis. Universität Wien. 2004.

[Ku09] Kurpjuweit, S.: *Stakeholder-orientierte Modellierung und Analyse der Unternehmensarchitektur.* PhD thesis. Universität St.Gallen. 2009.

[KUJ09] Källgren, A.; Ullberg, J.; Johnson, P.: *A Method for Constructing a Company Specific Enterprise Architecture Model Framework.* In (Kim, H.-K.; Lee, R. Y., Ed.): *10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, SNPD 2009.* pages 346–351. Daegu, Korea. 2009.

[KW07] Kurpjuweit, S.; Winter, R.: *Viewpoint-based Meta Model Engineering.* In (Reichert, M.; Strecker, S.; Turowski, K., Ed.): *2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2007).* LNI. pages 143–161. Bonn, Germany. 2007. Gesellschaft für Informatik.

[KW09] Kurpjuweit, S.; Winter, R.: *Concern-oriented business architecture engineering.* In *SAC'09: Proceedings of the 2009 ACM symposium on Applied Computing.* pages 265–272. New York, NY, USA. 2009. ACM.

[La04] Lankhorst, M. M.; van Buuren, R.; van Leeuwen, D.; Jonkers, H.; ter Doest, H.: *Enterprise architecture modelling-the issue of integration. Adv. Eng. Inform.* 18(4):205–216. 2004.

[La05] Lankhorst, M. M.: *Enterprise Architecture at Work: Modelling, Communication and Analysis.* Springer. Berlin, Heidelberg, Germany. 2005.

[La07] Lageström, R.: *Analyzing System Maintainability using Enterprise Architecture Models. Journal of Enterprise Architecture.* 3:33–42. 2007.

[La08a] Lagerström, R.; Chenine, M.; Johnson, P.; Franke, U.: *Probabilistic Metamodel Merging.* In (Bellahsene, Z.; Woo, C.; Hunt, E.; Franch, X.; Coletta, R., Ed.): *CAiSE Forum.* pages 25–28. Montpellier, France. 2008.

[La08b] Lankes, J.: *Metrics for Application Landscapes – Status Quo, Development, and a Case Study.* PhD thesis. Technische Universität München, Fakultät für Informatik. Munich, Germany. 2008.

[La09a] Lankhorst, M. M.: *Enterprise Architecture at Work: Modelling, Communication and Analysis.* Springer. Berlin, Heidelberg, Germany. 2nd edition. 2009.

[La09b] Lau, A.; Fischer, T.; Buckl, S.; Ernst, A. M.; Matthes, F.; Schweda, C. M.: *EA Management Patterns for Smart Networks.* In *SE 2009 – Workshopband.* pages 79–90. 2009.

[La09c] Op 't Land, M.; Proper, E.; Waage, M.; Cloo, J.; Steghuis, C.: *Enterprise Architecture – Creating Value by Informed Governance.* Springer. Berlin, Heidelberg. 2009.

[LD08] Land, M. O.; Dietz, J. L.: *Enterprise ontology based splitting and contracting of organizations.* In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing.* pages 524–531. New York, NY, USA. 2008. ACM.

[Le01] Ledeczi, A.; Maroti, M.; Bakay, A.; Karsai, G.; Garrett, J.; Thomason, C.; Nordstrom, G. et al.: *The Generic Modeling Environment.* In *Workshop on Intelligent Signal Processing.* Budapest, Hungary. 2001. IEEE.

[LHS08] Liimatainen, K.; Heikkilä, J.; Seppänen, V.: *A Framework for Evaluating Compliance of Public Service Development Programs with Government Enterprise Architecture.* In *Proceedings of the 2nd European Conference on Information Management and Evaluation.* pages 269–276. London, UK. 2008.

[LJ08] Lagerström, R.; Johnson, P.: *Using Architectural Models to Predict the Maintainability of Enterprise Systems.* In *12th European Conference on Software Maintenance and Reengineering.* pages 248–252. Athens, Greece. 2008. IEEE.

[LKL10] Lucke, C.; Krell, S.; Lechner, U.: *Critical Issues in Enterprise Architecting — A Literature Review.* In *Proceedings of the Sixteenth Americas Conference on Information Systems (AMCIS 2010).* Lima, Peru. 2010.

[Lo87] Lorenzen, P.: *Constructive Philosophy.* University of Massachusetts Press. Amherst, MA, USA. 1987.

[Lo01] Losee, J.: *A Historical Introduction to the Philosophy of Science.* Oxford University Press. New York, NY, USA. 4th edition. 2001.

[LS07] Lankes, J.; Schweda, C. M.: *Constructing Application Landscape Metrics: Why & How.* Technical Report TB0701. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2007.

[LS08] Lankes, J.; Schweda, C. M.: *Using Metrics to Evaluate Failure Propagation and Failure Impacts in Application Landscapes.* In (Bichler, M.; Hess, T.; Krcmar, H.; Lechner, U.; Matthes, F.; Picot, A.; Speitkamp, B. et al., Ed.): *Multikonferenz Wirtschaftsinformatik.* Berlin, Germany. 2008. GITO-Verlag.

[Lu03] Luftman, J. N.: *Competing in the Information Age – Align in the Sand.* Oxford University Press. New York, NY, USA. 2nd edition. 2003.

[Lu09] Luijpers, J.: *White Paper: Project Start Architecture – Version 1.0.* Technical report. Sogeti Nederland B.V. Diemen, The Netherlands. 2009.

[LW04a] Langenberg, K.; Wegmann, A.: *Enterprise Architecture: What Aspect is Current Research Targeting?* Technical report. Laboratory of Systemic Modeling, Ecole Polytechnique Fédérale de Lausanne. Lausanne, Switzerland. 2004.

[LW04b] Lê, L.-S.; Wegmann, A.: *Meta-model for Object-Oriented Hierarchical Systems.* Technical report. Laboratory of Systemic Modeling, Ecole Polytechnique Fédérale de Lausanne. Lausanne, Switzerland. 2004.

[LW05] Lê, L.-S.; Wegmann, A.: *Definition of an Object-Oriented Modeling Language for Enterprise Architecture.* In *38th Hawaii International Conference on System Sciences (HICSS 2005).* pages 179–188. 2005.

[LW06] Lê, L.-S.; Wegmann, A.: *SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture.* In *39th Hawaii International Conference on System Sciences (HICSS 2006).* 2006.

[Ma08] Matthes, F.; Buckl, S.; Leitel, J.; Schweda, C. M.: *Enterprise Architecture Management Tool Survey 2008.* Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2008.

[MaZT07] Magalhães, R.; Zacharias, M.; Tribolet, J. M.: *Making Sense of Enterprise Architectures as Tools of Organizational Self-Awareness (OSA).* In *2nd Workshop on Trends in Enterprise Architecture Research (TEAR 2007).* 2007.

[MD97] Meszaros, G.; Doble, J.: *A Pattern Language for Pattern Writing.* pages 529–574. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA. 1997.

[Mi95] Miller, J.: *Living Systems.* University of Colorado Press. 1995.

[Mo03] Monod, E.: *A Copernican Revolution in IS: using Kant's critique of pure reason for describing epistemological trends in IS.* In *AMCIS 2003: Proceedings of the 9^{th} Americas Conference on Information Systems.* Tampa, FL, USA. 2003.

[MS95] March, S. T.; Smith, G. F.: *Design and natural science research on information technology. Decis. Support Syst.* 15(4):251–266. 1995.

[MWF08] Murer, S.; Worms, C.; Furrer, F. J.: *Managed Evolution. Informatik Spektrum.* 31(6):537–547. 2008.

[My92] Mylopoulos, J.: *Conceptual modeling and Telos.* pages 49–68. Wiley. New York, USA. 1992.

[My11] Mykhashchuk, M.; Buckl, S.; Dierl, T.; Schweda, C. M.: *Charting the landscape of enterprise architecture management – An extensive literature analysis.* In *Wirtschaftsinformatik.* Zürich, Swiss. 2011.

[MZT10] Marques, J. a.; Zacarias, M.; Tribolet, J.: *A Bottom-Up Competency Modeling Approach.* In (Aalst, W.; Mylopoulos, J.; Sadeh, N. M.; Shaw, M. J.; Szyperski, C.; Albani, A.; Dietz, J. L. G., Ed.): *Advances in Enterprise Engineering IV.* volume 49 of *Lecture Notes in Business Information Processing.* pages 50–64. Springer Berlin Heidelberg. 2010.

[Nä08] Närman, P.; Schönherr, M.; Johnson, P.; Ekstedt, M.; Chenine, M.: *Using Enterprise Architecture Models for System Quality Analysis.* In *12^{th} IEEE International EDOC Conference (EDOC 2008).* pages 14–23. Washington, DC, USA. 2008. IEEE Computer Society.

[NCP91] Nunamaker, J. F.; Chen, M.; Purdin, T. D. M.: *Systems development in information systems research. J. Manage. Inf. Syst.* 7(3):89–106. 91.

[Ni06] Niemann, K. D.: *From Enterprise Architecture to IT Governance – Elements of Effective IT Management.* Vieweg+Teubner. Wiesbaden, Germany. 2006.

[NKF94] Nuseibeh, B.; Kramer, J.; Finkelstein, A.: *A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. IEEE Trans. Softw. Eng.* 20:760–773. 1994.

[No98] Noble, J.: *Classifying Relationships between Object-Oriented Design Patterns.* In *Australian Software Engineering Conference (ASWEC).* pages 98–107. Los Alamitos, CA, USA. 1998. IEEE Computer Society.

[No03] Noran, O.: *A Mapping of Individual Architecture Frameworks (GRAI, PERA, C4ISR, CIMOSA, Zachman, ARIS) onto GERAM.* In (Bernus, P.; Nemes, L.; Schmidt, G., Ed.): *Handbook on Enterprise Architecture.* pages 65–210. Berlin, Heidelberg, Germany. 2003. Springer.

[Ob05] Object Management Group (OMG): *Revised Submission for MOF 2.0 Query/View/Transformation (ptc/05-11-01).* 2005.

[Ob06a] Object Management Group (OMG): *Meta Object Facility (MOF) Core Specification, version 2.0 (formal/06-01-01).* 2006.

[Ob06b] Object Management Group (OMG): *UML Diagram Interchange.* `http://www.omg.org/spec/UMLDI/1.0/PDF` (last accessed 04-16-2011). 2006.

[Ob08] Object Management Group (OMG): *MOF 2.0 Facility and Object Lifecycle Specification, ptc/2008-02-20.* 2008.

[Ob10a] Object Management Group (OMG): *Business Motivation Model 1.1.* `http://www.omg.org/spec/BMM/1.1/` (last accessed 04-16-2011). 2010.

[Ob10b] Object Management Group (OMG): *Object Constraint Language (OCL) Available Specification, version 2.2 (formal/2010-02-01).* 2010.

[Ob10c] Object Management Group (OMG): *OMG Unified Modeling Language^{TM}(OMG UML), Infrastructure – Version 2.3 (formal/2010-05-03).* 2010.

[Ob10d] Object Management Group (OMG): *OMG Unified Modeling Language^{TM}(OMG UML), Superstructure – Version 2.3 (formal/2010-05-05).* 2010.

[Of09] Offermann, P.; Levina, O.; Schönherr, M.; Bub, U.: *Outline of a design science research process.* In *DESRIST '09: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology.* pages 1–11. New York, NY, USA. 2009. ACM.

[Ös07] Österle, H.; Winter, R.; Hoening, F.; Kurpjuweit, S.; Osl, P.: *Business Engineering: Core-Business-Metamodell. Wisu – Das Wirtschaftsstudium.* 36(2):191–194. 2007.

[ÖW03] Österle, H.; Winter, R.: *Business Engineering – Auf dem Weg zum Unternehmen des Informationszeitalters.* Springer. Berlin, Germany. 2nd edition. 2003.

[Pe09a] Peyret, H.: *Enterprise Architecture Tool Trends: Slow Adoption But Expanding Usage.* 2009.

[Pe09b] Peyret, H.: *The Forrester Wave: Business Process Analysis, EA Tools, And IT Planning, Q1 2009.* 2009.

[PH05] Pulkkinen, M.; Hirvonen, A. P.: *EA Planning, Development and Management Process for Agile Enterprise Development.* In *38th Hawaii International Conference on System Sciences.* Big Island, HI, USA. 2005. IEEE Computer Society.

[PHB08] Pries-Heje, J.; Baskerville, R.: *The Design Theory Nexus. MIS Quarterly.* 32(4):731–755. 2008.

[Pi02] Pierce, B. C.: *Types and Programming Languages.* The MIT Press. Cambridge, MA, USA. 2002.

[PNL07] Pulkkinen, M.; Naumenko, A.; Luostarinen, K.: *Managing information security in a business network of machinery maintenance services business - Enterprise architecture as a coordination tool. J. Syst. Softw.* 80(10):1607–1620. 2007.

[Pu06] Pulkkinen, M.: *Systemic Management of Architectural Decisions in Enterprise Architecture Planning. Four Dimensions and Three Abstraction Levels.* In *39th Hawaii International Conference on System Sciences (HICSS 2006).* volume 8. page 179c. 2006.

[PVSH05] Proper, E. H.; Verrijn-Stuart, A. A.; Hoppenbrouwers, S. J. B. A.: *On utility-based selection of architecture-modelling concepts.* In *APCCM '05: Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling.* pages 25–34. Darlinghurst, Australia, Australia. 2005. Australian Computer Society, Inc.

[QEJ10] Quartel, D.; Engelsman, W.; Jonkers, H.: *ArchiMate Extension for Modeling and Managing Motivation, Principles and Requirements in TOGAF.* `http://www.bizzdesign.com/index.php/component/docman/doc_download/16-extending-ea-modelling-with-business-goals-and-requirements` (last accessed 04-16-2011). 2010.

[RA09] Riege, C.; Aier, S.: *A Contingency Approach to Enterprise Architecture Method Engineering.* In *Service-Oriented Computing – ICSOC 2008 Workshops.* pages 388–399. 2009.

[RB06] Ross, J. W.; Beath, C. M.: *Sustainable IT Outsourcing Success: Let Enterprise Architecture be your Guide. MIS Quarterly Executive.* 5(4):181–192. 12 2006.

[RBW03] Rychkova, I.; Balabko, P.; Wegmann, A.: *Operational ASM Semantics behind Graphical SEAM Notation.* In *DAIS/FMOODS Ph.D. workshop.* pages 10–19. 2003.

[Re09] Regev, G.; Hayard, O.; Gause, D. C.; Wegmann, A.: *Modeling Service-Level Requirements: A Constancy Perspective.* In *RE 2009, 17th IEEE International Requirements Engineering Conference.* pages 231–236. 2009.

[RNE09] Raderius, J.; Närman, P.; Ekstedt, M.: *Assessing System Availability Using an Enterprise Architecture Analysis Approach.* In *Service-Oriented Computing — ICSOC 2008 Workshops: ICSOC 2008 International Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers.* pages 351–362. Berlin, Heidelberg. 2009. Springer-Verlag.

[Ro03] Ross, J. W.: *Creating a Strategic IT Architecture Competency: Learning in Stages. MIS Quarterly Executive.* 2(1). 2003.

[RRN09] Rhodes, D. H.; Ross, A. M.; Nightingale, D. J.: *Architecting the system of systems enterprise: Enabling constructs and methods from the field of engineering systems.* In *Systems Conference, 2009 3rd Annual IEEE.* pages 190–195. Vancouver, Canada. 2009.

[RV08] van Raadt, B. d.; van Vliet, H.: *Designing the Enterprise Architecture Function.* In (Steffen Becker, F. P.; Reussner, R., Ed.): *4th International Conference on the Quality of Software Architectures (QoSA2008).* volume 5281 of *Lecture Notes in Computer Science.* pages 103–118. Karlsruhe, Germany. 2008. Springer.

[RW73] Rittel, H. W. J.; Webber, M. M.: *Dilemmas in a general theory of planning. Policy Sciences.* 4(2):155–169. June 1973.

[RW06] Rychkova, I.; Wegmann, A.: *A Method for Functional Alignment Verification in Hierarchical Enterprise Models.* In *BUSITAL.* 2006.

[RW07] Rychkova, I.; Wegmann, A.: *Formal Semantics for Property-Property Relations in SEAM Visual Language: Towards Simulation and Analysis of Visual Specifications.* In *MSVVEIS.* pages 138–147. 2007.

[RWR06] Ross, J. W.; Weill, P.; Robertson, D. C.: *Enterprise Architecture as Strategy.* Harvard Business School Press. Boston, MA, USA. 2006.

[SB09] Steenbergen, M. v.; Brinkkemper, S.: *The Architectural Dilemma: Division of Work vs. Knowledge Integration.* In *Proceedings of the 4th International Workshop on Business/IT Alignment and Interoperability (BUSITAL'09).* Amsterdam, The Netherlands. 2009. Springer.

[SBB07a] Steenbergen, M. v.; Berg, M. v. d.; Brinkkemper, S.: *An Instrument for the Development of the Enterprise Architecture Practice.* In (Cardoso, J.; Cordeiro, J.; Filipe, J., Ed.): *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems, Volume EIS, Funchal, Madeira, Portugal, June 12-16, 2007.* pages 14–22. 2007.

[SBB07b] van Steenbergen, M.; van den Berg, M.; Brinkkemper, S.: *An Instrument for the Development of the Enterprise Architecture Practice.* In (Cardoso, J.; Cordeiro, J.; Filipe, J., Ed.): *ICEIS 2007 – Proceedings of the Ninth International Conference on Enterprise Information Systems, Volume EIS, Funchal, Madeira, Portugal, June 12-16, 2007 (3).* pages 14–22. 2007.

[SBK07a] Schermann, M.; Böhmann, T.; Krcmar, H.: *Fostering the Evaluation of Reference Models: Application and Extension of the Concept of IS Design Theories.* In (Oberweis, A.; Weinhardt, C.; Gimpel, H.; Koschmider, A.; Pankratius, V.; Schnizler, B., Ed.): *Wirtschaftsinformatik (2).* pages 181–198. Karlsruhe, Germany. 2007. Universitaetsverlag Karlsruhe.

[SBK07b] Schermann, M.; Böhmann, T.; Krcmar, H.: *A Pattern-based Approach for Constructing Design Theories with Conceptual Models.* In *ECIS2007: Processding of the European Conference on Information Systems.* pages 1368–1379. 2007.

[Sc01] Scheer, A.-W.: *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen.* Springer. Berlin, Germany. 4th edition. 2001.

[Sc02] Scheer, A.-W.: *ARIS – Vom Geschäftsprozess zum Anwendungssystem.* Springer. Berlin, Germany. 4th edition. 2002.

[Sc06a] Schekkerman, J.: *Enterprise Architecture Deliverables Guide.* Technical report. Institute For Enterprise Architecture Development. 2006.

[Sc06b] Schekkerman, J.: *Extended Enterprise Architecture Framework Essential Guide.* Technical report. Institute For Enterprise Architecture Development. 2006.

[Sc06c] Schekkerman, J.: *Extended Enterprise Architecture Maturity Model Suppot Guide.* Technical report. Institute For Enterprise Architecture Development. 2006.

[Sc06d] Schekkerman, J.: *How to Survive in the Jungle of Enterprise Architecture Frameworks: Creating or Choosing an Enterprise Architecture Framework.* Trafford Publishing. Victoria, BC, Canada. 2006.

[Sc08a] Schekkerman, J.: *Enterprise Architecture Good Practices Guide – How to Manage the Enterprise Architecture Practice.* Trafford Publishing. Victoria, BC, Canada. 2008.

[Sc08b] Schönherr, M.: *Towards a common terminology in the discipline of Enterprise Architecture.* In (Aier, S.; Johnson, P.; Schelp, J., Ed.): *Pre-Proceedings of the 3rd Workshop on Trends in Enterprise Architecture Research.* pages 107–123. Sydney, Australia. 2008.

[se05] sebis: *Enterprise Architecture Management Tool Survey 2005.* Technical report. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2005.

[SEJ08] Sommestad, T.; Ekstedt, M.; Johnson, P.: *Combining Defense Graphs and Enterprise Architecture Models for Security Analysis.* In *12th International IEEE Enterprise Distributed Object Computing Conference.* pages 349–355. Munich, Germany. 2008. IEEE Computer Society.

[SG89] Star, S. L.; Griesemer, J. R.: *Institutional Ecology: "Translations" and Coherence: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology. Social Studies of Science.* Vol. 19(3):pages 387–420. 1989.

[Sh86] Shewhart, W. A.: *Statistical Method from the Viewpoint of Quality Control.* Dover Publication. New York, NY, USA. 1986.

[SH93] Spewak, S. H.; Hill, S. C.: *Enterprise Architecture Planning – Developing a Blueprint for Data, Applications, and Technology.* John Wiley & Sons. New York, USA. 1993.

[SHL09] Seppänen, V.; Heikkilä, J.; Liimatainen, K.: *Key Issues in EA-Implementation: Case Study of Two Finnish Government Agencies.* In (Hofreiter, B.; Werthner, H., Ed.): *2009 IEEE Conference on Commerce and Enterprise Computing.* pages 114–120. Vienna, Austria. 2009. IEEE Computer Society.

[Si96] Simon, H. A.: *The Sciences of the Artificial.* MIT Press. Cambridge, Massachusetts, USA. 3rd edition. 1996.

[SK04] Schwarzer, B.; Krcmar, H.: *Wirtschaftsinformatik. Grundzüge der betrieblichen Datenverarbeitung.* Schäffer - Poeschel Verlag Stuttgart. 3rd edition. 2004. 3791021710.

[SR98] Schütte, R.; Rotthowe, T.: *The Guidelines of Modeling – An Approach to Enhance the Quality in Information Models.* In (Ling, T. W.; Ram, S.; Lee, M. L., Ed.): *Conceptual Modeling – ER'98.* pages 240–254. Berlin, Germany. 1998. Springer.

[SS07] Schelp, J.; Stutz, M.: *A Balanced Scorecard Approach to Measure the Value of Enterprise Architecture.* In (Lankhorst, M. M.; Johnson, P., Ed.): *Second Workshop on Trends in Enterprise Architecture Research.* pages 5–11. 2007.

[St73] Stachowiak, H.: *Allgemeine Modelltheorie.* Springer-Verlag. Wien, Austria. 1973.

[St98] Stack, G. J.: *Materialism.* In (Craig, E., Ed.): *Routledge Encyclopedia of Philosophy.* pages 170–171. New York, USA. 1998. Routledge.

[St05] Steenbergen, M. v.: *Architecture Maturity Matrix DYA – Version 1.0.* Technical report. Sogeti Nederland B.V. Diemen, The Netherlands. 2005.

[St09] Stelzer, D.: *Enterprise Architecture Principles: Literature Review and Research Directions.* In *4th Workshop on Trends in Enterprise Architecture Research (TEAR 2009).* Stockholm, Sweden. 2009.

[St10a] Steenbergen, M. v.; Bos, R.; Brinkkemper, S.; Weerd, I. v. d.; Bekkers, W.: *The Design of Focus Area Maturity Models.* In (Winter, R.; Zhao, J.; Aier, S., Ed.): *Global Perspectives on Design Science Research.* volume 6105 of *Lecture Notes in Computer Science.* pages 317–332. Springer Berlin / Heidelberg. 2010.

[St10b] Struck, V.; Buckl, S.; Matthes, F.; Schweda, C. M.: *Enterprise Architecture Management from a knowledge management perspective – Results from an empirical study.* In *5th Mediterranean Conference on Information Systems (MCIS2010), Tel Aviv.* 2010.

[SW08] Schelp, J.; Winter, R.: *On the Interplay of Design Research and Behavioral Research – A Language Community Perspective.* In (Vaishanvi, V.; Baskerville, R., Ed.): *Proceedings of the Third International Conference on Design Science Research in Information Systems and Technology (DESRIST2008), May 7-9, 2008, Westin, Buckhead, Atlanta, Georgia, USA,.* pages 79–92. Georgia State University, Atlanta, Georgia, USA. 2008.

[SW09] Schelp, J.; Winter, R.: *Language communities in enterprise architecture research.* In *DESRIST '09: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology.* pages 1–10. New York, NY, USA. 2009. ACM.

[SZ92] Sowa, J. F.; Zachman, J. A.: *Extending and Formalizing the Framework for Information Systems Architecture. IBM Systems Journal.* 31(3):590–616. 1992.

[Ta44] Tarski, A.: *The Semantic Conception of Truth: and the Foundations of Semantics. Philosophy and Phenomenological Research.* 4(3):341–376. 1944.

[Ta90] Takeda, H.; Veerkamp, P.; Tomiyama, T.; Yoshikawa, H.: *Modeling design processes. AI Mag.* 11(4):37–48. 1990.

[Th09a] The Open Group: *TOGAF "Enterprise Edition" Version 9.* `http://www.togaf.org` (last accessed 04-16-2011). 2009.

[Th09b] The Open Group: *TOGAF Version 9 – A Manual.* Van Haren Publishing. 9th edition. 2009. 978-90-8753-230-7.

[TT91] Tanaka, J. W.; Taylor, M.: *Object categories and expertise: Is the basic level in the eye of the beholder? Cognitve Psychology.* 23:457–482. 1991.

[Va01] Vasconcelos, A.; Caetano, A.; Neves, J.; Sinogas, P.; Mendes, R.; Tribolet, J. M.: *A Framework for Modeling Strategy, Business Processes and Information Systems.* In *5th IEEE International EDOC Conference (EDOC 2001).* pages 69–81. IEEE Computer Society. 2001.

[Va04] Vasconcelos, A.; Pereira, C. M.; Sousa, P.; Tribolet, J. M.: *Open Issues On Information System Architecture.* In *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS 2004).* 2004.

[Ve06] Venable, J. R.: *The Role of Theory and Theorising in Design Science Research.* In *Design Science Research in Information Systems and Technology.* 2006.

[VH05] Verschuren, P.; Hartog, R.: *Evaluation in Design-Oriented Research. Quality and Quantity.* 39(6):733–762. December 2005.

[VK04] Vaishnavi, V. K.; Kuechler, W. J.: *Design Research in Information Systems.* -. 2004.

[VS08] Valtonen, K.; Seppänen, V.: *Adaptation and adoption of Finnish government EA method – Preliminary guidelines for method users.* Ministry of Finance. 2008.

[VSL09] Valtonen, K.; Seppänen, V.; Leppänen, M.: *Government Enterprise Architecture Grid Adaptation in Finnland.* In *Hawaii International Conference on System Sciences.* Los Alamitos, CA, USA. 2009. IEEE Computer Society.

[VST03] Vasconcelos, A.; Sousa, P.; Tribolet, J. M.: *Information System Architectures: Representation, Planning and Evaluation. Journal of Systemics, Cybernetics and Informatics.* 1(6):78–84. 2003.

[VST05] Vasconcelos, A.; Sousa, P.; Tribolet, J. M.: *Information System Architecture Evaluation: From Software to Enterprise Level Approaches.* In *12th European Conference On Information Technology Evaluation (ECITE 2005).* 2005.

[VST07] Vasconcelos, A.; Sousa, P.; Tribolet, J. M.: *Information System Architecture Metrics: an Enterprise Engineering Evaluation Approach. The Electronic Journal Information Systems Evaluation.* 10(1):91–122. 2007.

[VST08] Vasconcelos, A.; Sousa, P.; Tribolet, J. M.: *Enterprise Architecture Analysis - An Information System Evaluation Approach. Enterprise Modelling and Information Systems Architectures.* 3(2):31–53. 2008.

[Wa05] Wagter, R.; van den Berg, M.; Luijpers, J.; van Steenbergen, M.: *Dynamic Enterprise Architecture: How to Make IT Work.* John Wiley. 2005.

[We03] Wegmann, A.: *On the Systemic Enterprise Architecture Methodology (SEAM).* In *SEAM). Published at the International Conference on Enterprise Information Systems 2003 (ICEIS 2003.* pages 483–490. 2003.

[We05] Wegmann, A.; Balabko, P.; Lê; Regev, G.; Rychkova, I.: *A Method and Tool for Business-IT Alignment in Enterprise Architecture.* In (Belo, O.; Eder, J.; ao e Cunha, F.; Pastor, O., Ed.): *Proceedings of the CAiSE'05 Forum.* pages 113–118. Porto, Portugal. 2005.

[We07a] Wegmann, A.; Lê, L.-S.; Regev, G.; Wood, B.: *Enterprise modeling using the foundation concepts of the RM-ODP ISO/ITU standard. Inf. Syst. E-Business Management.* 5(4):397–413. 2007.

[We07b] Wegmann, A.; Regev, G.; Rychkova, I.; Lê, L.-S.; Julia, P.: *Business and IT Alignment with SEAM for Enterprise Architecture.* In *11th IEEE International EDOC Conference (EDOC 2007).* pages 111–121. 2007.

[We08] Wegmann, A.; Kotsalainen, A.; Matthey, L.; Regev, G.; Giannattasio, A.: *Augmenting the Zachman Enterprise Architecture Framework with a Systemic Conceptualization.* In *12th IEEE International EDOC Conference (EDOC 2008).* pages 3–13. 2008.

[WF06] Winter, R.; Fischer, R.: *Essential Layers, Artifacts, and Dependencies of Enterprise Architecture.* In *1st Workshop on Trends in Enterprise Architecture Research (TEAR 2006)* [EDO06]. page 30.

[Wi53] Wittgenstein, L.: *Philosophical investigations.* Macmillan. New York, NY, USA. 1953.

[Wi07] Wittenburg, A.: *Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften.* PhD thesis. Fakultät für Informatik, Technische Universität München, Germany. 2007.

[Wi09] Winter, R.; vom Brocke, J.; Fettke, P.; Loos, P.; Junginger, S.; Moser, C.; Keller, W. et al.: *Patterns in Business and Information Systems Engineering. Business & Information Systems Engineering.* pages 468–474. 2009.

[Wi10] Winter, K.; Buckl, S.; Matthes, F.; Schweda, C. M.: *Investigating the state-of-the-art in enterprise architecture management method in literature and practice.* In *Proceedings of the 5th Mediterranean Conference on Information Systems.* 2010.

[Wo10] Wout, J. v.; Waage, M.; Hartman, H.; Stahlecker, M.; Hofman, A.: *The Integrated Architecture Framework Explained.* Springer. 2010.

[WR04] Weill, P.; Ross, J. W.: *IT Governance – How Top Performers Manage IT Decision Rights for Superior Results.* Harvard Business School Press. Boston, Massachusetts, USA. 2004.

[WRL05] Wegmann, A.; Regev, G.; Loison, B.: *Business and IT Alignment with SEAM.* In *REBNITA Workshop at 14th International Requirements Engineering Conference.* 2005.

[WS08] Winter, R.; Schelp, J.: *Enterprise architecture governance: the need for a business-to-IT approach.* In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing.* pages 548–552. New York, NY, USA. 2008. ACM.

[WW02] Webster, J.; Watson, R. T.: *Analyzing the Past to Prepare for the Future: Writing a Literature Review. MIS Quarterly.* 26(2):xiii—-xxiii. 2002.

[WWES92] Walls, J. G.; Widmeyer, G. R.; El Sawy, O. A.: *Building an Information System Design Theory for Vigilant EIS. INFORMATION SYSTEMS RESEARCH.* 3(1):36–59. 1992.

[WWES04] Walls, J. G.; Widmeyer, G. R.; El Sawy, O. A.: *Assessing Information System Design Theory in Perspective: How Useful Was Our 1992 Rendition? Journal of Information Technology Theory and Practice.* 6(2):43–58. 2004.

[Za87] Zachman, J. A.: *A framework for information systems architecture. IBM Syst. J.* 26(3):276–292. 1987.

[Za10] Zacarias, M.; Pinto, H. S.; Magalhães, R.; Tribolet, J.: *A 'context-aware' and agent-centric perspective for the alignment between individuals and organizations. Inf. Syst.* 35:441–466. 2010.

Nomenclature

ASM	Abstract State Machine, page 74
ATL	Atlas Transformation Language, page 230
BEAMS	building blocks for EA management solutions, page 5
BEAMS	building blocks for EA management solutions, page 93
BEAMS	building blocks for EA management solutions, page 206
BPMN	Business Process Modeling Notation, page 74
CMDB	Configuration Management Database, page 147
DSL	domain specific language, page 224
DSL	domain-specific language, page 222
E/R	Entity Relationship, page 113
EA	Enterprise Architecture, page 1
EMF	Eclipse Modeling Framework, page 220
EMLs	Enterperise Modeling Languages, page 61
ER	entity-relationship, page 24
FRISCO	Framework of Information System Concepts, page 46
GBB	glossary building block, page 138
GBB	glossary building blocks, page V
GBB	glossary building blocks, page 11
GEMCs	Generic Enterprise Modeling Concepts, page 61
GERA	Generalised Enterprise Reference Architecture, page 60
GERA	Generalised Enterprise Reference Architecture, page 98
GERAM	Generalised Enterprise Reference Architecture and Methodology, page 60
GME	Generic Modeling Environment, page 122
GME	Graphical Modeling Environment, page 223
GMF	Graphical Modeling Framework, page 122

GMF Graphical Modeling Framework, page 223
I-pattern Information model patterns, page 124
IBB information model building block, page 138
IBB information model building blocks, page V
IBB information model building blocks, page 11
IC identity condition, page 114
IC identity condition, page 135
IEEE Institute of Electrical and Electronics Engineers, page 47
IFAC International Federation of Automatic Control, page 60
IFIP International Federation of Information Processing, page 60
IM2L information model meta-language, page 11
IM2L information model meta-language, page 138
IM2L information model meta-language, page 167
IM2L information model meta-language, page 242
ITML IT Modeling Language, page 67
KPIs key performance indicators, page 2
LBB language building block, page 136
M-pattern Methodology patterns, page 124
MBB method building block, page 136
MEMO Multi-perspective Enterprise Modeling, page 67
MML MEMO Meta Modeling Language, page 168
MML MEMO meta-language, page 68
MOF Meta Object Facility, page 168
MOF Meta Object Facility, page 221
OCL Object Constraint Language, page 120
OML Object Modeling Language, page 67
OrgML Organization Modeling Language, page 68
ORM Object Role Model, page 168
QVT Query/View/Transformation, page 168
ResML Resource Modeling Language, page 67
ScoreML Score Modeling Language, page 67
SEAM Systemic Enterprise Architecture Methodology, page 73
SML Strategy Modeling Language, page 68
SOM Semantic Object Model, page 63
SOX Sarbanes-Oxley Act, page 98
SVG Scalable Vector Graphics, page 223
TAFIM Technical Architecture Framework for Information Management, page 69
TOGAF The Open Group Architecture Framework, page 1
UML Unified Modeling Language, page 113
UML Unified Modeling Language, page 168