Outline of a method for designing domain-specific modelling languages

Frank, Ulrich

In: ICB Research Reports - Forschungsberichte des ICB / 2010

DOI: https://doi.org/10.17185/duepublico/47075

URN: urn:nbn:de:hbz:464-20180918-085429-1

Link: https://duepublico.uni-duisburg-essen.de/servlets/DocumentServlet?id=47075

Source: ICB-Research Report No. 42, December 2010

ICB
Institut für Informatik und Wirtschaftsinformatik

Ulrich Frank

Outline of a Method for Designing Domain-Specific Modelling Languages

ICB-RESEARCH REPORT

UNIVERSITÄT DUISBURG ESSEN

ICB-Research Report No. 42
December 2010

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The ICB Research Reports comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Author's Address:

Ulrich Frank

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
D-45141 Essen

ulrich.frank@uni-due.de

Contact:

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
45141 Essen

Tel.: 0201-183-4041
Fax: 0201-183-4011
Email: icb@uni-duisburg-essen.de

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)

Abstract

In recent years, the development of domain-specific modelling languages has gained remarkable attention. This is for good reasons: A domain-specific modelling language incorporates concepts that represent domain-level knowledge. Hence, systems analysts are not forced to reconstruct these concepts from scratch. At the same time, domain-specific modelling languages contribute to model integrity, because they include already constraints that would otherwise have to be added manually. Even though there has been a considerable amount of research on developing and using domain-specific modelling languages, there is still lack of comprehensive methods to guide the design of these languages. With respect to the complexity and risk related to developing a domain-specific modelling language, this is a serious shortfall. This research report is aimed at a contribution to filling the gap. It presents the prolegomena of a method for developing domain-specific modelling languages, which is based on the experience gathered in several language specification projects. The method consists of two main parts: a meta modelling language and a process model. The MEMO meta modelling language (MEMO MML) is specified in a further report. Therefore, its description is restricted to a brief overview. Instead, the main focus of this report is on the process model, which describes essential steps to be accounted for during the development of a domain-specific modelling language. It includes heuristics to develop requirements and meta modelling guidelines that support frequent design decisions. The description of the method is complemented by examples which are mainly taken from the design of the MEMO Organisation Modelling Language.

Table of Contents

Figures

Tables

1 Introduction

Designing conceptual models of high quality is a demanding task. It is a widespread assumption that most users would be overburdened with developing and using elaborate conceptual models without further support. A plethora of modelling methods is addressing the corresponding demand. At first, these methods made mainly use of general purpose modelling languages (GPML), such as the ERM or the UML. In recent years, the idea of using modelling languages that were designed for more specific purposes – so called domain-specific modelling languages (DSML) – has gained increasing popularity. This is for convincing reasons: It is certainly more appealing to describe large models with domain-level concepts than to construct them from generic concepts such as 'entity type', 'attribute' etc. Imagine one would need to write technical specifications – not to speak of other types of documents – with a vocabulary that is restricted to a handful of basic terms only. Instead, a DSML provides concepts that correspond to the technical language used in the targeted domain. Hence, they promise to promote convenience and productivity of modelling, since users do not have to reconstruct technical terms on their own. At the same time, they contribute to model quality, since the concepts that are provided by a DSML should be the result of an especially thorough development process. The integrity of models, as a specific aspect of their quality, is promoted, too, since the syntax and semantics of a DSML allow for preventing nonsensical models to a certain degree. In addition to that, a DSML will often feature a special graphical notation (concrete syntax) that helps to improve clearness and comprehensibility of models. Figure 1 illustrates the advantages of a DSML – in this case for modelling IT infrastructures – over a GPML. The model designed with a GPML uses the same graphical symbols for any concept in the wide range of its possible applications, which prevents them from being self-expressive. Also, it allows for assertions that are perfectly valid – with respect to the syntax and semantics of the language – but are wrong with respect to an obvious interpretation in the targeted domain.

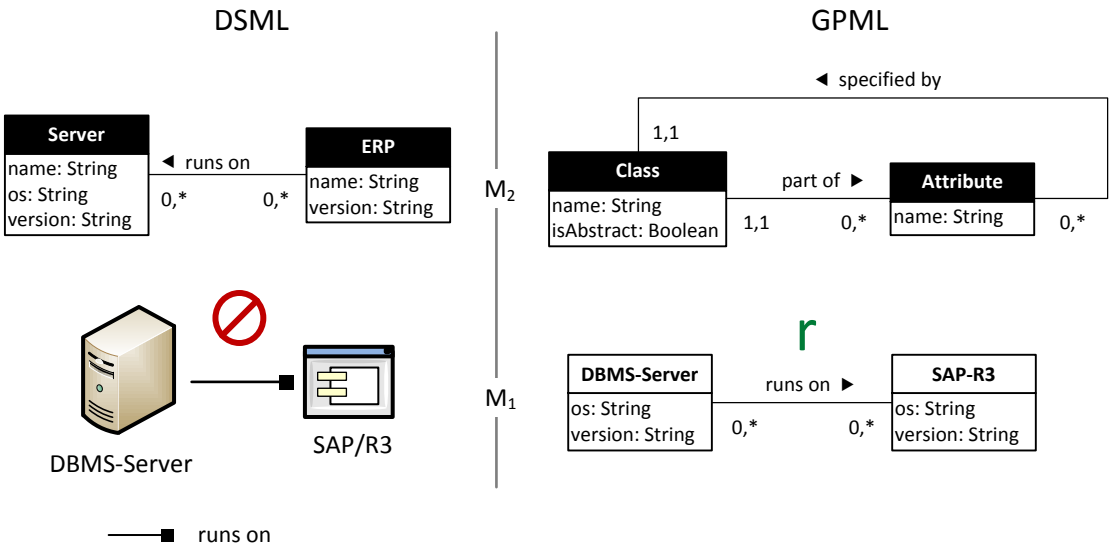

Figure 1: Illustration of DSML in comparison to GPML

A core motivation for developing domain-specific modelling languages (DSML) is the assumption that domain-specific concepts support users more effectively with structuring a domain for a certain purpose than a general purpose modelling language. However, designing a DSML is not a trivial task. Against this background, it is remarkable that there is hardly any method for guiding the development of modelling languages in general, the design of DSML in particular. Work on the evaluation of modelling languages, e.g. Goldstein and Storey (1990), Opdahl and Henderson-Sellers (1999), Frank (1998), provides language designers with criteria they should account for. However, they are restricted to a few aspects only – and do not focus on guidelines for how to satisfy the suggested language features. Formal quality criteria – see for instance: Süttenbach and Ebert (1997) – are more concrete, but not sufficient for guiding the design of a language because they fade out the relationship of a language to the targeted domain and to the prospective users. In recent years, method engineering, i.e. a methodical support for developing modelling methods, has gained remarkable attention. Since a modelling method consists of at least one modelling language and a corresponding process model, one would assume that work on method engineering includes support for the development of modelling languages. However, this is not the case. While there is a plethora of approaches on method engineering (for an overview see Henderson-Sellers and Ralyté, 2010), they all focus on the design of process models and take the modelling language as given. In a comprehensive book on domain-specific modelling, Kelly and Tolvanen (2008) describe the development and use of DSML. While they provide valuable advice that is based on a number of corresponding projects, their focus is mainly on technical aspects, especially on meta modelling and on code generation. Also, their notion of domain and DSML is different from the one that is advocated in this report (see chapter 4).

The current lack of support may be attributed to the fact that those who develop modelling languages are usually highly specialized experts. However, that does not mean that they are

not in need for support, since the design of modelling languages can be an extremely demanding task. For several years, an essential part of our research has been directed towards the development of DSML and corresponding modelling methods. During this time we have faced numerous challenges that are related to certain recurring design decisions and to peculiarities of the development process. Many discussions with various designers of modelling languages have shown that we share this experience with others. This may be regarded as a deluxe problem that is relevant for a few only. However, the lack of substantiated guidance increases the risk of poorly designed modelling languages, which will eventually affect all prospective language users. With the increasing popularity of DSML and corresponding model-based approaches to software development, this problem becomes even more crucial. While a number of tools support the specification of DSML and the realization of corresponding model editors, prospective users can expect only little guidance with designing a language that fits its purpose.

This report is aimed at presenting prolegomena of a method for designing modelling languages. It is focused on language specification through meta models. This is mainly for the reason that meta modelling is the approach of choice not only for the specification of languages for enterprise modelling, but also for most domain-specific languages that are used for model-driven software development. In order to illustrate the method, examples will be shown occasionally. For this purpose, the MEMO-Meta Modelling Language (MML) (Frank, 2011a), which is also part of the presented method, will be used. Note, however, that the use of the process model is not restricted to the MEMO-MML. It can also be applied using other meta modelling languages, which, however, may require some adaptations. The primary focus of the method is on languages for enterprise modelling, e.g. for modelling business processes, resources, IT infrastructure, strategies etc. It is, however, not necessarily restricted to this focus.

2 Background

DSML have not been a research subject for long yet. As a consequence, there is no unified terminology. Even a term like ‘domain’, which has been used for many years in conceptual modelling is not used consistently. Often it is not defined explicitly. This may be contributed to the fact that it is not trivial to define such a basic term – and that most people in our field can be assumed to have a sufficient understanding of it. In this report, we will also go with this assumption. Instead, we will restrict our consideration on discussing basic assumptions, prospects and specific challenges of DSML compared to GPML. Finally, we will briefly characterize enterprise modelling as the intended scope of the presented method. An upcoming report will provide a comprehensive discussion of foundational terms in the field of enterprise modelling including ‘domain’ and ‘domain of discourse’ (Frank, 2011b).

2.1 Domain-Specific Modelling Languages: Basic Assumptions and Prospects

The conception of DSML is based on distinguishing them from GPML. Like GPML, DSML are languages for conceptual modelling. The following definitions express characteristic differences.

> A **GPML** is a modelling language that is thought to be independent from a particular domain of discourse. Instead, it should be suited to cover a wide range of domains. It consists of generic modelling concepts that do not include any specific aspects of a particular domain of discourse.
>
> A **DSML** is a modelling language that is intended to be used in a certain domain of discourse. It enriches generic modelling concepts with concepts that were reconstructed from technical terms used in the respective domain of discourse. A DSML serves to create conceptual models of the domain, it is related to.

The above definitions of DSML and GPML focus on characteristic features. They do not, however, provide a detailed conception of a DSML in comparison to a GPML. In particular, they do not allow for clearly deciding whether a specific modelling language qualifies as a DSML or as a GPML. Instead, they rather emphasize the principle idea. Gaining a better understanding of specific benefits and challenges related to the design of DSML recommends taking a close look at underlying assumptions and prospects in comparison to GPML. A pivotal objective of conceptual models is bridging the gap between two languages: The language that is used in the targeted domain of discourse (the intended application area) and the technical language used to implement corresponding information systems, i.e. the domain of discourse characteristic for IT experts (see Figure 2).

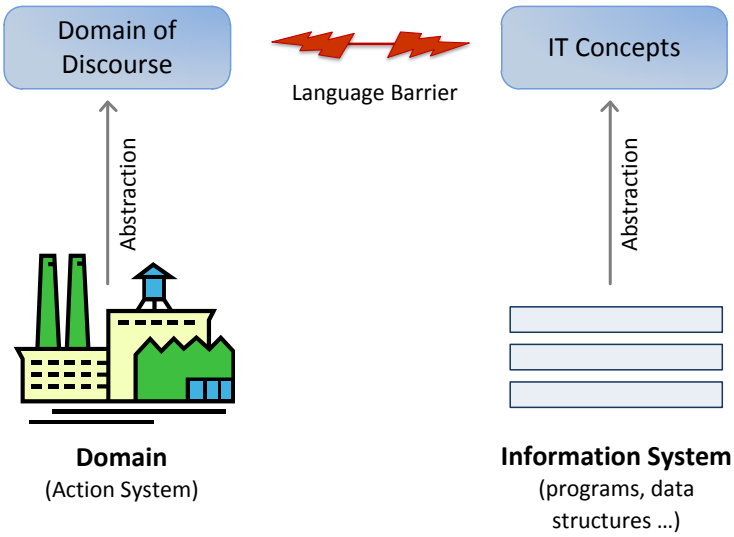

Figure 2: Language Barrier between Application Domain and Information Technology

With respect to productivity and integrity, it would be worthwhile to define a mapping between the two languages. A mapping would require common generic concepts that allow for reconstructing both, the concepts of the domain of discourse and the IT concepts (see Figure 3).

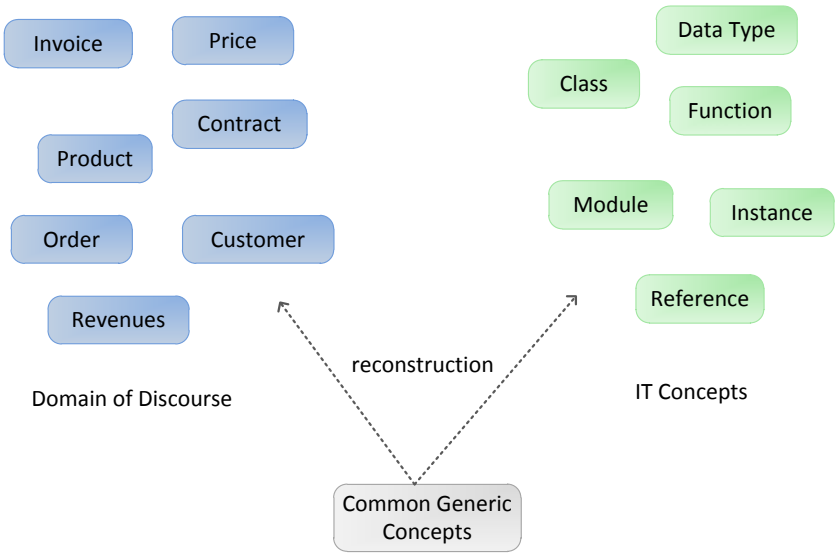

Figure 3: Common Concepts as a Foundation for Mapping Domain Concepts to IT Concepts (Examples)

With respect to information systems, generic concepts are abstractions of the underlying machine. They include concepts such as data type, class, attribute, state, state transition etc. With respect to the domains to be represented, the search for generic language concepts has been subject of Philosophy, but also of Linguistics and Cultural Anthropology for long. In Philosophy, it was motivated by an ontological question: What are generic categories of being? While the objectives of Ontology are not undisputed in Philosophy, it is nevertheless worthwhile looking at proposals for basic ontological categories. Aristotle suggested catego-

ries such as “substance”, “quantity”, “quality”, “state”, and “affection”. Apparently, categories like “quality” or “affection” do not allow for a clear mapping to basic concepts of information systems. This is different with categories suggested by philosophers who aim at more precisely defined concepts. Grossmann, while building on Aristotle, is drawing on modern logics and suggests categories such as “individuals”, “properties”, “relations”, “classes”, “quantifiers” (Grossmann, 1983). Bunge, who suggests regarding the world in terms of systems (Bunge, 1979) in order to gain more precise conceptions, proposes ontological categories he refers to as the “furniture of the world” (Bunge, 1977). They include “things”, “properties of things”, “attributes of things”, “classes of things”, “coupling of things”, “laws and lawful states”. Apparently, Grossmann’s and Bunge’s proposals correspond almost directly to elementary concepts used in systems analysis and design. This is remarkable, since there is no evidence that the specification of the ERM or the UML was inspired by philosophical Ontology. Nevertheless, this correspondence provides the foundation for mapping concepts of a domain of discourse to concepts used for constructing information systems. Figure 4 illustrates this mapping and the respective abstractions. Note that the common generic concepts correspond to concepts typically offered by GPML.

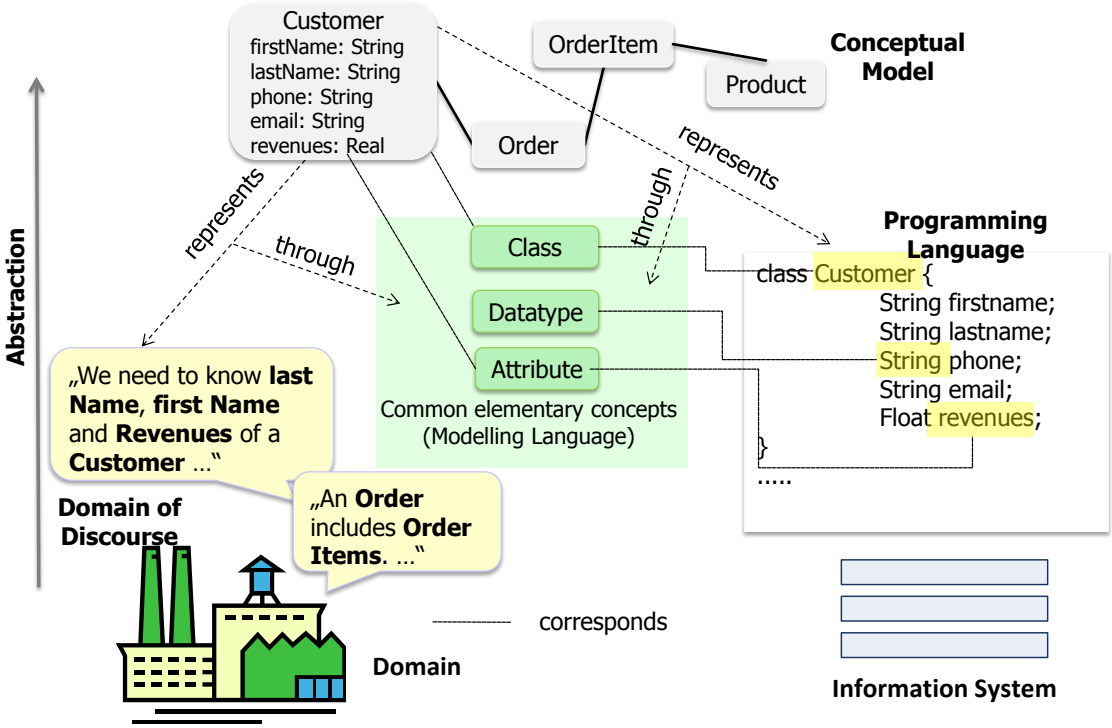

Figure 4: Mapping between both Domains through Generic Concepts

GPML are certainly helpful with bridging the gap between application domains and corresponding information systems. However, creating conceptual models from generic concepts such as ‘class’, ‘attribute’ etc. implies a tremendous effort and is a threat to model quality, too. Imagine one would have to write a business report in a language with primitive generic

concepts only! Providing modellers with a DSML that includes reconstructions of domain-specific concepts they are familiar with is certainly much more attractive.

2.2 Domain-Specific Programming Languages

There are two ways to map the concepts of a DSML to implementation languages. First, they can be mapped to the generic concepts of general purpose programming languages (GPPL). Unfortunately, this will result in a loss of semantics. Take, for instance, a concept like 'Position' of a DSML for modelling organisations. It can be instantiated in a type like 'Sales_Assistant'. In a programming language, 'Sales_Assistant' would be represented e.g. as a class. Hence, the information that it is an instance of the meta type 'Position' would be lost. Second, the implementation language could also be domain-specific. Languages for specifying workflow schemas are a prominent example. Their concepts correspond to those of DSML for business process modelling. In recent years, domain-specific programming languages have been an emerging field of research (see, e.g. Bettin and Clark, 2010, Ducasse and Gîrba, 2006). Usually, they are referred to as DSL. However, since this acronym is sometimes also used for domain-specific modelling languages, we rather use DSPL instead. Different from specifying a domain-specific concept with generic concepts, DSPL allow for making domain-specific concepts part of the programming language. For this purpose, they feature a distinct meta layer. As a consequence, they offer similar advantages to programming as DSML to modelling. Figure 5 illustrates the use of a fictitious DSPL in contrast to the use of a fictitious GPPL referring to the example used to compare DSML against GPML (see Figure 1). While a GPPL allows for defining classes with arbitrary features, the concepts provided by a DSPL allow for predefined instantiations only. In the example, these could be instances of a meta type such as 'Platform' or 'Server' – depending on the intended abstraction.

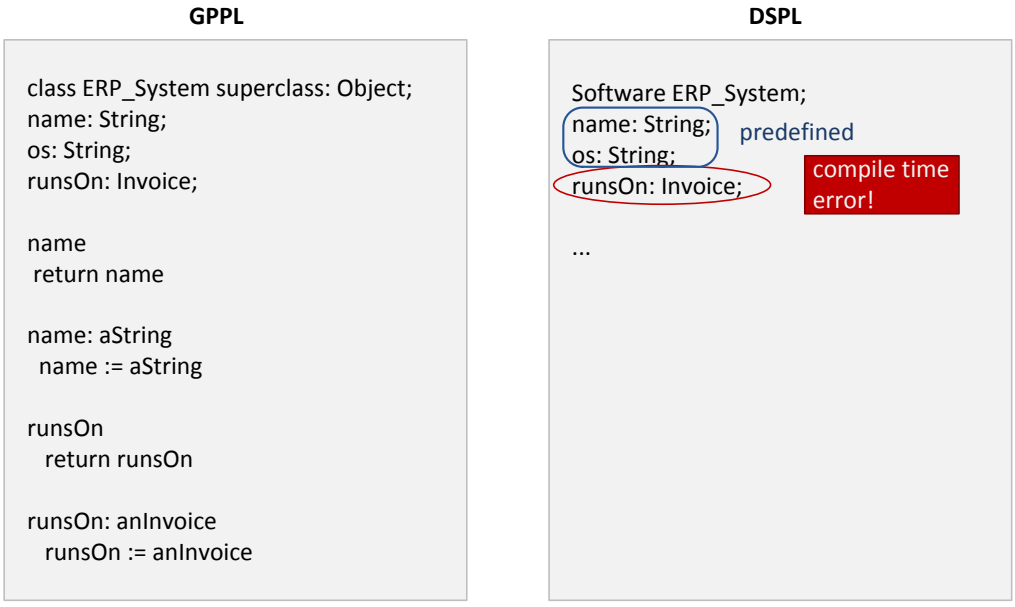

Figure 5: Illustration of DSPL compared to GPPL

Using DSML and corresponding DSPL allows for bridging the gap between application domains and information systems on a higher level of semantics, thus contributing to increased productivity and integrity (see section 2.1). At the same time, it creates an additional challenge: It would not be enough developing a DSML alone. Instead, every DSML development would require developing a corresponding DSPL as well as the related tools.

While generating code from models can substantially contribute to productivity and code quality, it will usually create the problem to synchronize model and code consistently. Therefore, it would be more appealing to aim at a common representation of models and code (for a detailed discussion of this prospect see Frank and Strecker, 2007, p. 20 f.) – in other words: to develop domain-specific languages that serve as a foundation for both conceptual modelling and implementation.

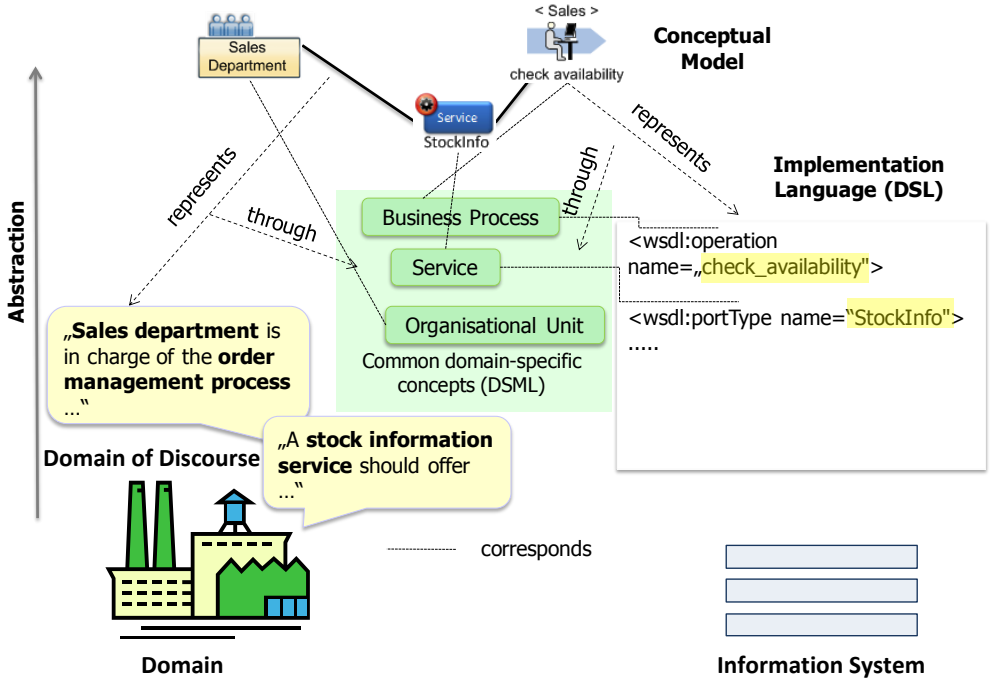

Figure 6: Illustration of Mapping DSML to DSPL

2.3 Multi-Perspective Enterprise Modelling

The method presented in this report is designed for DSML in the realm of enterprise modelling, which is our main field of research. Languages to be used for enterprise modelling cover a remarkable range. They have one characteristic in common that does not have to apply for DSML in other areas. It is implied by the conception of an enterprise model:

> An **enterprise model** serves the purpose to support analysis, design and management of information systems together with corresponding action systems. It comprises conceptual models of software systems, e.g. object or component models, and conceptual models of the surrounding action systems, e.g. business process models or strategy

> models. The models that constitute an enterprise model are integrated through the use of common concepts.

Note that action system and information system are not limited by the boundaries of a particular organisation. Instead, an enterprise model may represent inter-organisational action systems, too. Integrating the various models that are part of an enterprise model does not only foster the integrity of an enterprise model, it also provides a medium for users with different professional backgrounds to communicate more effectively: While everybody should have a good chance to find an abstraction that corresponds to his personal perception (and conception), he is also supported in realizing how the models he prefers are related to those of other stakeholders.

Hence, DSML for enterprise modelling will usually address at least two diverse groups of potential users: people with an IT background on the one hand, and actors that are business-oriented on the other hand. To further emphasize this fact, we use the notion *of multi-perspective enterprise model*:

> A **multi-perspective enterprise model** is an enterprise model that emphasizes accounting for *perspectives*. The term 'multi-perspective' is purposefully overloaded. On the one hand, it represents different conceptions of perspective. On the other hand, it refers to differentiating specific perspectives related to one conception of perspective. The first conception is a psychological one: In this conception, a perspective represents a specific professional background that corresponds to cognitive dispositions, technical languages, specific goals and capabilities of prospective users. The second conception refers to the *representation* of perspectives within an enterprise model. In this sense it relates directly to the respective DSML that are expected to provide concepts that correspond to those characteristic for certain (psychological) perspectives. The third conception goes beyond the factual content of an enterprise model and stresses critical *meta perspectives*. It recommends supplementing – and confronting – an enterprise model with the relevant context, i.e. with aspects that are required for a reflective interpretation of an enterprise model but that bulk against formalisation.

At first sight it may seem that the development of a DSML for enterprise modelling will mainly focus on the second aspect – the specification of modelling concepts. However, without an idea of relevant psychological perspectives, it will be impossible to develop appropriate language concepts. At the same time, it suggests to account for the problem that a psychological perspective cannot be completely represented in a DSML or a corresponding model. Perspective as a psychological concept serves to express that the way an individual perceives and understands the world, i.e. his 'Weltanschauung', is characterized by a cognitive disposition that is shaped by socialisation, experiences, language games, etc. It constitutes sense and is a framework for understanding our surroundings. Experiences relate to patterns of interaction, to the appreciation of other actor's attitudes and expectations. We cannot expect to adequately represent all aspects of this psychological concept with a DSML. This limitation of DSML is a specific problem in those cases where relevant aspects of a concept can be represented only through simplifications. For instance: If a DSML for modelling

organisations includes a concept like ‘competence’ to describe a particular position type, it may cause the problem that prospective users account for these explicit aspects only, neglecting other aspects that bulk against formalisation, such as, e.g. commitment or persistence. The third meaning of the term ‘multi-perspective enterprise model’ in the above definition relates to this possible problem. It is aimed at the reflective assessment and use of enterprise models and suggests an additional, critical perspective that accounts for aspects not represented explicitly in a model. It also stresses that a mere engineering approach to designing and using enterprise models is not sufficient.

Reflecting upon the term ‘perspective’ has two major consequences for the development of a DSML. First, it creates a specific challenge for requirements analysis, because it demands to focus on specific cognitive styles, attitudes etc. that will often not be directly accessible. Therefore, it is especially important to intensively communicate with prospective users aiming at gaining a deeper understanding of their perspective. Second, it demands for supplementing language concepts with explicit remarks that they are based on simplifications that may cause dysfunctional effects in case they are used without critical distance.

The method is intended to support everybody who develops DSML in the area of enterprise modelling, i.e. academics as well as practitioners. However, it is not conceptualized as a *research* method. Nevertheless, some peculiarities of developing a DSML as part of a research project will be accounted for in the process model (for an elaborate study on corresponding research methods see Frank, 2006b). Note that the focus on enterprise modelling does not exclude that the method is suited for developing other DSML, too.

3 Generic Requirements for DSML

The previous definition of DSML gives an impression of core ideas. However, it does not include a differentiated consideration of purposes and quality criteria. To develop a more elaborate conception of DSML, we will look at generic requirements any DSML should fulfil. Publications on requirements for modelling languages can be differentiated into three categories. Approaches that focus on *formal requirements*, usually originating in Computer Science, emphasize aspects such as correctness and completeness. Typically, they do not account for characteristics of the targeted domain or for specific use cases. Therefore, they are not sufficient for covering all relevant requirements. Approaches that stress *pragmatic aspects* focus the question what prospective users demand from an artefact. For this purpose they emphasize an empirical approach by analysing requirements through interviewing users and analysing the patterns of action and/or decision making that should be supported. Following such an approach faces the challenge that most users will lack sufficient knowledge of DSML to answer questions directly related to generic requirements.

Other approaches are aimed at developing a foundation of modelling languages by referring to the field of *Ontology* in Philosophy (for a more comprehensive overview see Frank, 2006a). In order to develop a theoretical foundation of conceptual modelling, Weber suggests referring to the ontology introduced by Bunge (1977) mentioned already above. Based on Bunge's work, he proposes "ontological completeness" and "ontological clarity" as a core requirement (Weber 1997, pp. 92) modelling languages should satisfy. The demand for ontological completeness is satisfied, if a modelling language covers all basic concepts proposed by the reference ontology, which was adapted from Bunge's work (Wand and Weber, 1995). It includes basic concepts to describe static, (e.g. "things", "properties of things") and dynamic (e.g. "events", "states") abstractions. Ontological clarity demands for avoiding concept overload, redundancy or excess. A concept of a modelling language is overloaded, if it maps to more than one concept of the ontology. It is redundant, if there is already another language concept that maps to the same concept of the ontology. It is excessive, if it maps to none, it is regarded as excessive. While Weber's approach was adopted by others – e.g. Opdahl and Henderson-Sellers (1999), Fettke and Loos (2003), it suffers from a severe misconception, which makes it especially useless for DSML. Usually, a modelling language is focusing on a particular abstraction, e.g. static, functional or dynamic. Hence, it is not ontological complete *on purpose* – and for a good reason. Concepts offered by a DSML may be overloaded on purpose, too. Take, for instance, the concept 'process' that may be part of a DSML for modelling business processes. It will usually combine static features (e.g. average costs) and dynamic features.

Due to the lack of adequate guidance, we will not build on ontologies to analyse requirements for DSML. Instead, we apply and refine the framework presented in Frank and Laak (2003). It accounts for formal and pragmatic requirements where the latter are differentiated

into user-oriented and application-oriented aspects. Note that these are not orthogonal dimensions.

Formal requirements are of special relevance, if formal procedures (that allow for automation) are required for analysing or transforming models.

User-oriented requirements refer to the prospective modellers' perception of language concepts and their visualisation.

Application-oriented requirements are related to the intended modelling domains and generic modelling purposes.

3.1 Formal Requirements

While the formalisation of a modelling language is not an end in itself, it can be very useful with respect to analysing and transforming models. In computer science, it is common to demand for a *correct* and *complete* specification of modelling languages. A specification is correct and complete, if it allows for the clear identification of all models, which include syntactic or semantic errors. At the same time, a correct and complete specification allows for generating the set of all models that are syntactically and semantically correct. In other words, a correct and complete specification refers to the formalisation of both, the syntax and the semantics of a language. Formalisation requires the use of a formal (meta) language. For our purpose, the quest for formalizing a modelling language seems too rigid. While it makes sense to formalize the syntax in order to foster formal analysis and the construction of modelling tools, formalizing the semantics is hardly feasible. That is for two reasons. First, the domain of interest may include concepts that balk at formalisation, e.g. 'competence' or 'customer satisfaction'. In this case, the only option is 'pseudo' formalisation. It occurs in two variants, which may be combined. First, a concept can be reduced to a formal specification that represents only certain aspects. Describing a resource type's value by referring to an ordinal scale would be an example of this kind of formalisation. The second variant features representations with no formal semantics that depend on human interpretations, e.g. a string that represents a natural language term or statement. In both cases, one can hardly speak of formal semantics, since it is not possible to formally decide whether a particular description is appropriate or not. Nevertheless, 'pseudo' formalisation can be helpful for documenting and analysing organisational models.

Against this background, we shall relax the demand for formalisation:

> **Requirement F1**: The specification of a modelling language should include a precise and complete specification of its syntax. In an ideal case, this will be a formal specification. In any case, the syntax specification should allow a human to clearly decide whether a specific model is syntactically correct or not. *Rationale*: Syntactical correctness is a prerequisite for model analysis and for keeping models in a consistent state.

Requirement F2: The concepts used to specify a DSML should correspond to concepts of languages used for software design. *Rationale*: A clear correspondence will promote a straightforward implementation of respective modelling tools. Note, however, that this requirement may be in conflict with other requirements.

Within the specification of a language, there is no clear distinction between syntax and semantics. Syntactical rules can be used to exclude models that make no sense. The semantics can be specified by formal and informal rules.

Requirement F3: The rules defining the semantics of a modelling language should be suited to clearly guide prospective users with the construction of appropriate models and their adequate interpretation. These rules should be formalized, if this does not compromise the intended meaning. *Rationale*: Only if the semantics of modelling concepts is precisely defined, conceptual models can be unambiguous, which in turn is a prerequisite for (automated) analysis.

The convenient and safe development and maintenance of models recommends concepts that allow for a high level of abstraction. A high level of abstraction fosters reuse (e.g. through generalisation/specialisation) and integrity.

Requirement F4: The modelling language should feature concepts on a high level of semantics to support model integrity and reuse. *Rationale*: The more (domain-specific) semantics a modelling concepts includes, the better it is suited to prevent users from accidental errors (see Figure 1). Note, however, that this requirement may be in conflict with requirement A1.

3.2 User-Oriented Requirements

The prospective users of a DSML include domain experts, systems analysts, and software developers. While it is likely that these groups will emphasize different specific requirements, there are three generic requirements that make sense for all users: simplicity, comprehensibility and convenience of use. Note that these requirements are not necessarily compatible.

The simpler a language, the more likely it is that users are not overcharged. As a consequence, one can expect that a simple language will foster the construction of error-free models. While there is no clear definition of simplicity, we can assume that a language is the more simple, the less the number of its concepts and the number of the rules to define its syntax and semantics. The *comprehensibility* of a language may be fostered by its simplicity. But it seems that it is more important that the concepts it provides correspond to the terminology prospective users are familiar with.

Requirement U1: The concepts of a modelling language should correspond to concepts prospective users are familiar with. That recommends reconstructing existing terminology. Furthermore, it recommends using graphical symbols that are suited to illustrate the corresponding concepts' meaning. *Rationale*: The more users are familiar with

the concepts of a DSML and their representation, the easier it will be for them to understand and use them properly.

Note, however, that satisfying this requirement may be aggravated by the diversity of domains – and corresponding terminologies – a modelling language is supposed to cover. Convenience of use refers to the effort that is required to build a model. This requirement is contrasting the demand for simplicity. If a language provides domain specific concepts, it supports those users' productivity that develop models for this domain, since they do not need to construct these concepts on their own. However, the more specialized concepts a modelling language includes, the higher will be the effort to learn it.

Requirement U2: A DSML should provide a manageable set of basic concepts that are sufficient for creating simple models. *Rationale*: Often, there will be users that do not need to develop elaborate models. For them, it would be an unnecessary burden, if they could use the language only after they had learned more elaborate concepts, too. In other words: If possible, a DSML should include a subset that represents a lightweight version.

Note that this requirement makes especially sense with more complex DSML. In these cases, it will be beneficial for occasional users, if they do not have to learn the entire language to develop a model that fits their needs.

The following requirement relates to the previous one and also to requirement A3:

Requirement U3: The modelling language should allow for building models on various levels of detail and abstraction. A modeller should not be forced to specify details he does not need. *Rationale*: Usually, different groups of users have different demands for the level of abstraction and detail provided by a model. Some will be bothered by too much detail, while others focus problems that require a higher degree of detail.

3.3 Application-Oriented Requirements

Incorporating domain-specific concepts into a language is only one option to foster productivity. Only if the semantics of a concept is invariant across the entire class of intended models, it is suited to be incorporated into the language. Otherwise, it would limit the usability of the language. In addition to that, to not compromise the quest for simplicity too much, only those concepts should be included in the language that are needed on a regular base. Including a concept in the language has one major advantage over representing it in an accompanying reference model: Its adequate use can be enforced through the language specification.

Requirement A1: A modelling language should provide domain specific concepts as long as their semantics is invariant within the scope of the language's application. *Rationale*: Only if the semantics of a modelling concept is invariant within the range of its intended use, it is possible to satisfy all prospective users

Note that this requirement may lead to exclude concepts that are not invariant from a DSML or to limit the scope of the DSML's application. The concepts a modelling language should

include depend on the purpose the corresponding models should fulfil, hence, on its intended application. In general, the concepts of a language should be *adequate* with respect to the set of intended applications. On the one hand, that implies that a language should not be overloaded with concepts that are not required. On the other hand, it means that the language concepts should allow for descriptions on a level of detail that is implied by the application purposes. This recommends carefully analysing possible applications. However, one cannot show that a certain set of applications is sufficient for all times. Therefore, a language should allow for application-specific extensions.

> **Requirement A2**: The concepts of a language should allow for modelling at a level of detail that is sufficient for all foreseeable applications. To cover further possible applications, it should provide extension mechanisms. *Rationale*: A DSML is intended to cover a certain domain only. At the time of its specification, there needs to be a clear idea of this domain and related applications. However, it cannot be excluded that further aspects of the domain will be discovered that should be represented by the DSML. Extensions mechanisms provide support for dealing with this change. Note, however, that they will usually offer a compromise only. If the extensions are relevant enough, a modification of the language itself may be the better option.

Conceptual models are focusing the type level: The concepts of a model do not represent particular instances, but types. However, sometimes the elaborate design of conceptual models requires further levels of abstraction. Consider for instance the feature that every business process starts at a certain point in time and terminates at a certain point in time. While this feature applies to all instances of a type, it is not a direct feature of the type. The type may have been created at a certain point in time. But that represents clearly a different meaning.

> **Requirement A3**: A modelling language should provide concepts that allow for clearly distinguishing different levels of abstraction within a model. *Rationale*: Conceptual models may represent different levels of abstraction, e.g. types and – in rare cases – instances. Overloading a model with different levels of abstraction compromises an appropriate interpretation of a model.

Often, conceptual models are transformed into other representations, e.g. into implementation level documents. These transformations should not be biased by ambiguity.

Requirement A4: There should be a clear mapping of the language concepts to the concepts of relevant target representations. In an ideal case, all information required by the target representations can be extracted from the model. That requires that the concepts of the language allow for expressing all concepts of relevant target representations. *Rationale*: The most important – but not the only – case for the requirement is software development. The semantics of programming languages is characterized by a few subtle peculiarities. They concern e.g. the notion of class or specialisation/generalisation. A DSML can either adopt the semantics of implementation languages or it can provide a clear mapping to these.

Due to our conception of multi-perspective enterprise models, the development of a DSML should be accompanied by a critical evaluation.

Requirement A5: In case a concept of a DSML is based on a problematic reduction/simplification, this should be clearly stated with the language documentation. *Rationale*: If a credulous user applies respective concepts and eventually beliefs in them, his judgement may get compromised. In other words: The concept would not promote a differentiated problem analysis, but rather hamper it.

4 Specific Challenges: Contingent Target

The rationale for developing a DSML seems to be convincing. Compared to a GPML, a DSML promises to improve both, productivity and quality. As a consequence, prospective users should be eager to get such a powerful tool. And of course, those who are in charge of approving the required investment should be supportive. Unfortunately, deciding for the development of a DSML will usually be not that easy. This is for various reasons:

Lack of support by users: Today, a DSML is still an artefact most prospective users are not familiar with. At the beginning, they do not know sufficiently what they may expect. Often, there will be no similar DSML available that could be used as a demonstrator. The lack of knowledge about the targeted artefact will most likely not promote the prospective users' enthusiasm. But even those who understand the basic idea do not have to appreciate it: The introduction of a new language does not only imply the effort of learning it, it will also require giving up the use of GPML users are familiar with. Hence, it comes with the threat of challenging existing competence and reputation.

Amorphous requirements: As long as users lack a clear imagination of the instrument they may expect, it will be difficult to get them involved in the process of gathering and shaping requirements. On the other hand, language designers will often be not sufficiently familiar with the targeted domain to define requirements on their own. This problem will be the more serious the less experience designers of a DSML have gained in similar projects.

Economics hard to judge: Usually, an economic justification will be most effective. That would require showing that the investment into a DSML produces a satisfactory return. However, calculating the economics of a DSML in advance is facing serious obstacles. On the one hand, the economic benefit to be expected from using a DSML is hard to predict – especially when it comes to quantifying it. While there is good reason to assume that it will foster quality and productivity, it may well happen that an insufficiently designed DSML and corresponding tools do not keep up to their promises. Due to relative little experience, the investment into developing and using a DSML leaves significant risks. As a consequence, it will be challenging to convince managers who decide about respective budgets.

Kelly and Tolvanen emphasize the quest for an economic justification of DSML, too. They respond to it with an optimistic advice: "Domain-specific approaches should generally be used whenever possible. They normally produce better results than general-purpose approaches." (Kelly and Tolvanen, 2008, p. 34). While we sympathize with this advice, it will often be regarded as not sufficient to justify a major investment.

Little incentives for pioneers: The costs of using a DSML depend on the range of its use. The higher the level of (re-) use, the more attractive are the economies of scale. If a DSML and corresponding tools can be obtained at little cost, it is likely that more and more prospective users will try it. Hence, as long as a critical mass has not been reached, there is need for pio-

neers who are willing to invest into the development and establishment of a DSML. To justify such an investment, a company will usually want to know the potential of the respective market and the share it may expect to get. Predicting these numbers is risky. Also, for prospective customers standardisation becomes more and more important. Defining a standard is not only beyond the capabilities of most potential vendors, it will also take an unpredictable long time. As a consequence, the incentives for investing into DSML and related technologies may often be regarded as not convincing. This conflict between the obvious – though not exactly quantifiable benefit – of an artefact and the risk that is related to the initial development costs is well-known. It is the case, too, for reference models and for sophisticated application systems. One approach to overcome this conflict is the establishment of open communities of developers and prospective users following the model of open source software. A corresponding initiative for conceptual models and modelling languages is outlined in Frank and Strecker (2007).

The difficulties in analysing the economics of DSML in advance are also related to a general design conflict. To take advantage of economies of scale, a DSML should be reusable in a wide range of application scenarios. Hence, the language concepts should not comprise too much domain-specific semantics. At the same time, however, a DSML should be a tool that provides effective support. The more domain-specific semantics a DSML includes the better its contribution to productivity – in those cases it fits. Figure 7 illustrates these considerations – and at the same time that trying to determine an optimal level of semantics would be a hopeless undertaking: Not only that measuring the level of semantics provided by a DSML on a metric scale is hardly possible, the benefits of additional semantics will be hard to judge in a single case – not to speak of numerous cases with considerable variance.

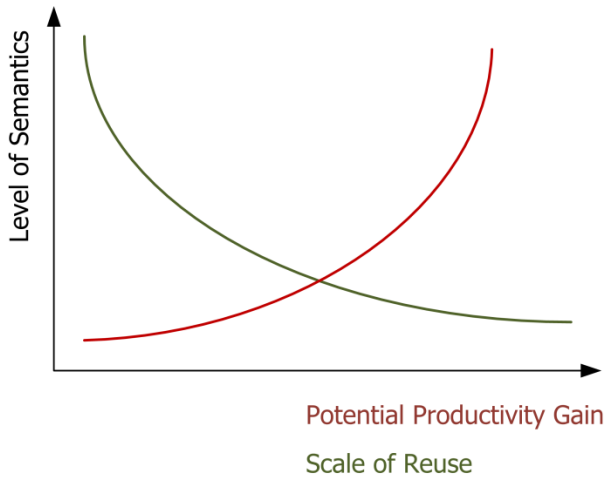

Figure 7: The Effect of Semantics on DSML Benefits

While it does not seem to be a promising idea to pursue an approach to calculate the optimal level of semantics a DSML should feature, it is nevertheless required to make corresponding decisions, namely to determine whether or not to develop a DSML for a certain domain – and how specific it should be. In their approach to developing and using DSML Kelly and

Tolvanen avoid the problem of determining an economic scale of reuse in advance by restricting the application of a DSML to one particular organisation: "A domain here is generally a highly focused problem domain, typically worked on by 5 – 500 developers in the same organisation." (Kelly and Tolvanen, 2008, p. xiii). Apart from the fact that the definition is circular, the authors do not explain why they limit the number of prospective users to 500. It is hard to see why one should not go for a higher number of users. The authors give one reason why to restrict the use of a DSML to one organisation: If it is regarded as an asset that generates competitive advantage, it makes sense not to make it available to competitors. However, this is only one aspect of using a DSML across a range of organisations. It is conceivable that various organisations that are not competitors use a common DSML, which would result in lower costs. Kelly and Tolvanen suggest that investments into DSML will pay off anyway – independent from the scale of reuse:

> "We are not talking here about squeezing another 20% out of our existing developers, programming languages, or development environments. Our industry is long overdue for a major increase in productivity: the last such advance was over 30 years ago, when the move from assemblers to compilers raised productivity around 500%. Our experiences, and those of our customers, colleagues, and competitors, have shown that a similar or even larger increase is now possible, through what we call Domain-Specific Modelling. Indeed, the early adopters of DSM have been enjoying productivity increases of 500-1000% in production for over 10 years now." (Kelly and Tolvanen, 2008, p. xiii)

Even though the claimed productivity gain is backed by a few studies with a very specific focus (Kelly and Tolvanen, 2008, p. 22), the question remains how much the development of a DSML costs and, hence, how its overall economic efficiency can be determined. Therefore, it remains a challenging problem to think about a satisfactory ratio of specific semantics and scale of reuse.

The previous discussion shows that the realisation of DSML faces multiple, mutually enforcing contingencies. The contingency related to defining domain-specific requirements increases the contingency related to determining the economics. The contingency related to prospective users' attitude increases the contingency of requirements analysis. These are serious obstacles. However, they do not discredit the idea of developing and using a DSML. Instead, they emphasize the need for more transparency in order to reduce contingency, risk and cost. A method that guides the design of DSML can contribute to that. Also, the outline of generic requirements for any DSML increases transparency and helps with a better understanding.

5 Requirements for the Method

In general, a method for designing a DSML should provide an appropriate meta modelling language and a process model that provides guidance. Requirements to be fulfilled by a meta modelling language are of outstanding importance. Nevertheless, they are not discussed here, since they are subject of a further research report (Frank, 2011a). The corresponding process model has to account for various requirements, which are in part in conflict. Some of them are well-known from method engineering; others are specific to the design of DSML. For a process model to provide substantial guidance, it should not be too generic. In other words: It should account for demanding facets of designing DSML. On the other hand, the design of a DSML is a complex task. It does not only require the development and assessment of meta models, it also demands to account for the specific peculiarities of its intended use and for economic aspects. Hence, for the development of a DSML it is mandatory to appoint qualified, professional personnel, which is a reason to make a process model not too specific: otherwise it would threaten to compromise its users' creativity or to be declined for being too restrictive. The conflict between making a process model more generic or rather specific applies to its reuse, too. While the range of reuse grows the more generic a process model is, the level of support provided by reusing it in a particular case grows the more specific it is – provided it fits the requirements of the particular case. This corresponds to the ambivalent effects of semantics incorporated into a DSML (see Figure 7). While this principal conflict cannot be avoided, a method for designing DSML should account for it, which is addressed by Requirement **P1** to **Requirement P10**. While these requirements are in part related to the meta modelling language, too, the focus here is mainly on the process model. These requirements did not result from a survey. Instead, they reflect the experience we have gathered with developing various DSML.

Requirement P1: A method for guiding the design of DSML should include a description of the range of possible DSML it covers. *Rationale*: For a method to be not too generic, it will make sense to narrow its scope. Hence, potential users of the method need to be informed whether it fits their purpose.

Requirement P2: Peculiarities and challenges related to DSML design should be addressed on a level of detail that is demanded by professional decision making. *Rationale*: The development of a DSML is confronted with design decisions that may challenge even experienced language analysts. If a method does not provide respective support, it would jeopardize the quality of the targeted DSML. Note, however, that a method can only address design decisions that are typical for the range of DSML it covers.

The following requirement supplements the previous one:

Requirement P3: Conflicting design goals should be made explicit, even if no solution can be provided. *Rationale*: As with many other artefacts, design goals may be compet-

ing or even in conflict. In the case of DSML, these conflicting goals are sometimes not obvious. Therefore, the method should promote transparency.

Requirement P4: Within the range of its intended use, the method should provide guidelines to adapt it to specific needs. *Rationale*: There are good reasons both for making a process model generic and more specific. To relax the resulting conflict, the method should include guidelines for adapting the process model without jeopardizing its integrity.

The pivotal challenge for a method to support the design of DSML is to encounter the contingencies related to analysing and determining requirements. They include both, the features of a DSML and the economics of developing and using it.

Requirement P5: Prospective users should be effectively supported in gaining a clear understanding of the central ideas the conception of a DSML is based on. This implies to give them a clear impression of prospects related to the use of the targeted DSML and corresponding tools. *Rationale*: Often, prospective users do not know what they can expect from a DSML and from corresponding modelling tools. Only, if they can clearly relate the DSML to the tasks they perform, they can be expected to actively participate in analysing requirements.

Requirement P6: The method should provide a description of requirements that are generic for the range of targeted DSML. *Rationale*: On the one hand, generic requirements contribute to a better understanding of the conception of a DSML. Therefore, this requirement supplements Requirement **P5**. On the other hand, providing prospective users with a description of generic requirements, promotes productivity and quality.

Usually, it will be mandatory to justify investments into DSML development. Often, there will be no or only little experience with this kind of artefact available.

Requirement P7: The method should account for the need to determine the economics of developing and using a DSML and corresponding tools in advance. For this purpose, it should foster transparency by pointing at essential factors that influence DSML economics. *Rationale*: The development of a DSML and corresponding tools will usually require substantial resources. Especially, it demands for appointing highly qualified personnel. At the same time, experiences with the prospective economic benefit of deploying DSML are scarce. While it will usually be no option to reliably calculate costs and benefits in advance, it is nevertheless important to increase transparency to give decision makers a chance to recognize and assess influencing factors.

Requirement P8: The method should provide support for assessing and selecting a meta modelling language and corresponding development tools. *Rationale*: Usually, it will be no option to specify a new meta modelling language. Also, the development of a modelling tool from scratch will be far too expensive in most cases. Selecting a meta modelling language is not independent from selecting a development environment such as a meta modelling tool, because the development environment will usually feature some kind of meta language. Both, assessing meta modelling languages and respective development environments are complex tasks that demand for support.

Requirement P9: The method should account for protection of investment and therefore emphasize a longer term perspective. *Rationale*: The development of a DSML and corresponding tools will usually require a significant investment. It includes training and change management. The longer a DSML can be used, the higher the prospective return. Therefore, adequate measures to enhance or supplement a DSML to meet possible future requirements are of great value.

Requirement P10: The method should provide guidelines for the design of graphical notations. *Rationale*: The graphical notation can be of major relevance for a DSML's suitability and productivity. Also, language designers will often lack specific expertise in this area. Therefore corresponding guidelines are important.

Requirement P10: The tasks that need to be performed during the development of a DSML should be assigned to roles that are characterized by a clear profile. *Rationale*: The complexity of developing a DSML will usually require division of labour among stakeholders with different skills. This implies in turn the need for collaboration. Descriptive roles help with staffing a project and with defining and communicating core responsibilities.

6 Elements of the Method

The core elements of a modelling method are a modelling language and a corresponding process model. Applied to a method for developing DSML – which includes the development of a meta model – this conception implies the existence of a meta modelling language and a process model that guides its use. Despite its relevance, the meta modelling language will not be considered in much detail in this report. This is for two reasons. First, the conception of a meta modelling language is a complex matter on its own. Therefore, the meta modelling language that we use, MEMO-MML, is subject of a dedicated research report (Frank, 2011a). Second, the method should not be restricted to one particular meta modelling language. Often, the choice of a meta modelling language depends on contextual factors such as the need to comply with certain standards or with previous decisions. It also depends on the choice of the environment for developing corresponding tools. Hence, the method is conceptualized in a way that allows for replacing the MEMO-MML with other meta languages. This requires special attention only when design decisions relate to specific features of a language.

6.1 Meta Modelling Language

The description of the MEMO MML will be restricted to an overview of core concepts. Most of these concepts should be provided by other meta modelling languages as well. There are, however, a few concepts included that are not prevalent. They will be briefly described, since they are relevant for some design decisions that will be discussed later. The MEMO MML is specified by a meta meta model that comprises a set of OCL (Object Constraint Language, OMG, 2010) constraints (Frank, 2011a). Its graphical notation corresponds to the style of the Entity Relationship Model (ERM) or the UML. However, in order to support a clear distinction of meta models from models on the object level, names of meta types are printed in white on a black background (see Figure 8).

The MEMO-MML includes four specific features that are not available with most other meta modelling languages (for a comprehensive description and specification see Frank, 2011a):

Derived attribute: An attribute of a meta type within a meta model can be marked as "derived" to indicate that its value can be calculated from other values within the meta model – if they are available. An example would be the number of event types included in a business process type.

Obtainable attribute: An attribute of a meta type within a meta model can be marked as "obtainable" to indicate that its value can be obtained from an external source. For instance: The performance level assigned to a business process type could be provided by an ERP system that calculates a corresponding indicator based on the analysis of the corresponding process instances.

Intrinsic feature: As a default, the concepts of a meta model are instantiated to types. Sometimes, this results in the problem that the essence of a term includes features that do not apply directly to the type level, but to the instances represented by a type. Marking these features as 'intrinsic' indicates that their instantiation is delayed to the instance level. For example: The meta type 'Process' that is part of a language for modelling business processes includes attributes like 'name','averageExecutionTime' etc. They are attributes of the types, e.g. 'Order Management' that can be instantiated from 'Process'. In addition to that, one may want to express that every process instance has a start and a stop time. These are not features of a process type. Hence, they would be marked as intrinsic to avoid their initialisation on the type level.

Language-level types: A model is an abstraction. Therefore, it should not represent particular instances. However, there are exceptions to this rule. For instance: A language for modelling logistic networks should include concepts to represent cities. If 'City' was used as a type within a particular model, the level of abstraction would usually be too high. Instead, it will often be the case that one wants to represent concrete cities. This would be possible only, if the meta model used to specify the language included types in addition to meta types. To satisfy this demand, the MEMO-MML features language-level types, even though they result in overloading meta models, which compromises their appropriate interpretation.

Figure 8 gives an overview of key language concepts and corresponding notational elements provided by the MEMO-MML.

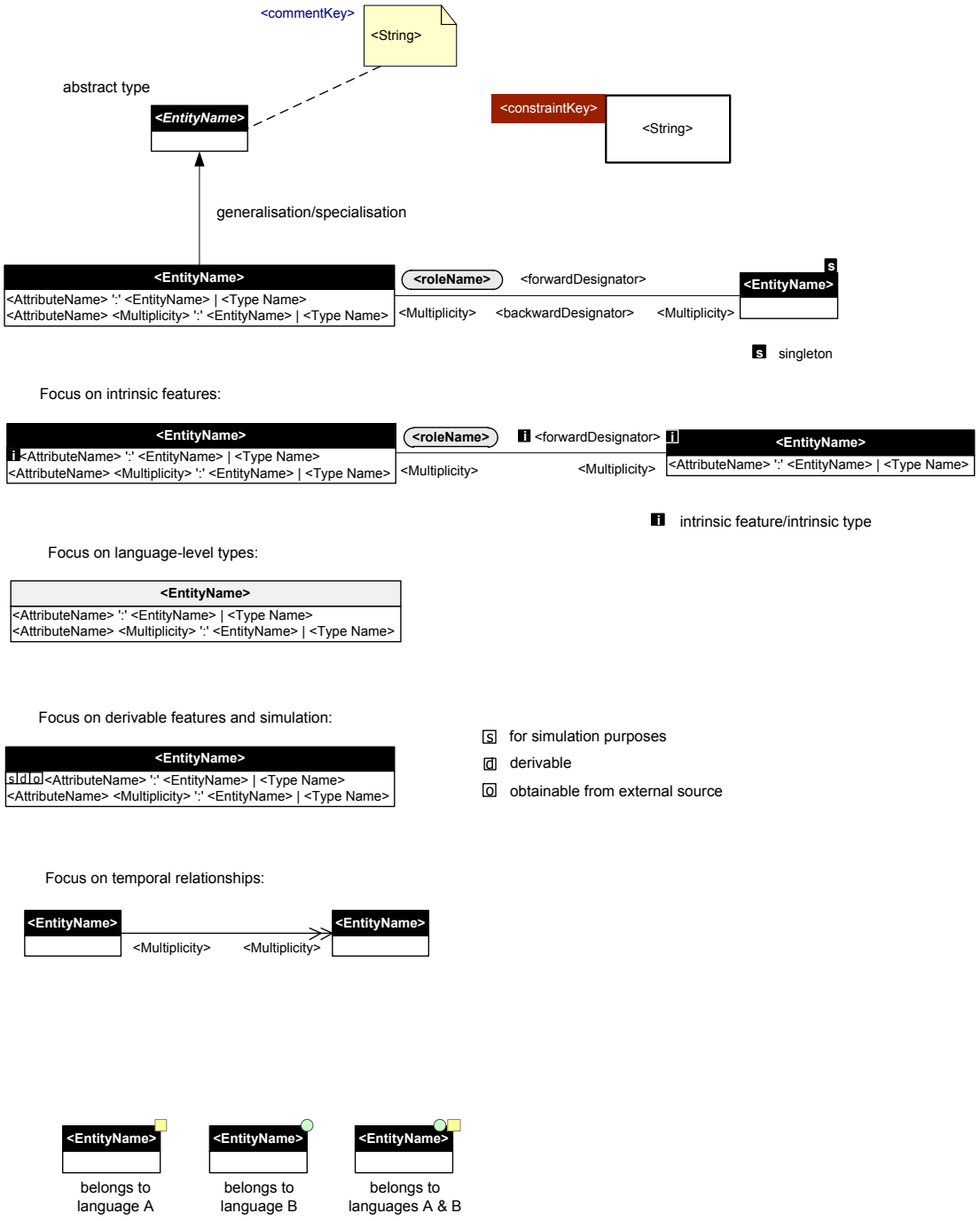

Figure 8: MEMO-MML: Overview of Concepts and Graphical Notation

6.2 Process Model

The specification of a DSML can be a task of remarkable complexity. It depends on intellectual creativity, especially on the ability to discover (or create) abstractions that are powerful with respect to a given purpose. The act of abstraction is a matter of individual cognitive dispositions. While the impact of a process model on cognitive dispositions is very limited, it may still help with reducing the overall complexity – thus promoting productivity and quality. Hence, the process model that is proposed in this report is intended to provide an in-

formed *orientation* – certainly not a cookbook. It is based on experience made with the development of various DSML. The process model is composed of four elements. A *macro process* describes the major phases. To guide its organisation, each phase is described according to a certain *structure*. Among other things it refers to a *micro process* that represents the suggested course of action for each phase of the macro process and to a set of roles, which are described in a corresponding *role model*. While the efficient use of a DSML will usually require a corresponding tool, the selection and development of a modelling tool is not covered in detail by the process model, because that would require accounting for peculiarities of demanding software development projects, which are not at the centre of this study.

6.2.1 The Role Model

Developing a DSML requires a variety of skills. This is even more the case as it will usually involve the selection or even development of corresponding tools. The following roles are abstractions that represent a certain set of skills and responsibilities. The conceptualisation of roles is based on the assumption that the corresponding skills and responsibilities are required for the development of a DSML. The profiles of two roles may partially overlap. A role may be assigned to one or more actors. Also, an actor may hold many roles. The proposed role profiles represent an ideal case that serves as an orientation for staffing. Note that it is not mandatory that these roles are filled by actors, who are actually working in the domain of interest. Instead, it is possible that somebody else fills a role – provided he has the necessary competence and skills.

Domain Expert

rationale	A domain expert is required to provide relevant knowledge about goals, constraints and patterns for action within the targeted domain.
professional training	• should have a formal education in a field that corresponds to essential tasks and problems within the domain • should be in command of a sophisticated technical language • should have professional experience and/or analytical skills to be sensitive for critical success factors and risks
information technology skills	• no particular skills required; however, should have an idea of the preconditions for automation
mental capabilities	• should be interested in analysing preconditions of successful action • should appreciate the use of methods (the targeted DSML would be part of)
attitude	• should be open for change • should be willing to engage in cross-disciplinary projects

User

rationale	This role represents prospective users of the DSML and correspond-

	ing tools. It is required for gathering and refining requirements.
professional training	• should be trained for the tasks he performs • the training should include the use of an advanced domain-specific technical language
information technology skills	• should be able to appreciate and successfully use software tools; the level of required sophistication may vary
mental capabilities	• should be able to understand and willing to appreciate the change of traditional patterns of work
attitude	• should be open for change • should be interested in improving productivity and quality of his work

Business Analyst

rationale	This role is implied by the scope of the presented method: Enterprise modelling is always related to a business perspective. For this reason, a business analyst is required to focus on a DSML's contribution to business performance.
professional training	• should have a formal education that provided him with a technical language, frameworks and methods to analyse and assess patterns of action in organisations • should be trained in the use of methods
information technology skills	• should be able to assess the preconditions, benefits and risks to be expected from using software tools
mental capabilities	• should be able to abstract from traditional patterns of work • should be able to reflect upon the role of language for successful action in business • should be able to act as a mediator to managers • should be able to realize the potential of a DSML – and the preconditions of its successful use
attitude	• should be interested in cross-disciplinary collaboration • should be open for learning about new methods and tools

Language Designer

rationale	This role is of pivotal relevance for the development of a DSML. Language designer is not an established profession yet, nor are the required skills entirely taught in prevalent academic programs. This makes staffing a critical success factor – and may require the need to first develop the required competence.
professional training	• should have a formal education – at best in Information Systems or Computer Science with additional studies in linguistics – that enables him to design conceptual models on different levels of abstraction • should be trained in enterprise modelling which implies substantial knowledge about methods to analyse and design organisational action systems

	• should have advanced experience in the design and evaluation of DSML and corresponding tools
information technology skills	• should be familiar with common approaches to specify (semi-) formal languages • should have a background in Software Engineering. This is required to understand and overcome the semantic gap between meta models and implementation languages.
mental capabilities	• should have an advanced competence in developing ambitious abstractions • should be aware of the role of language as a core instrument for structuring and managing the targeted domain • should appreciate the use of methods (the targeted DSML would be part of) • should be able and willing to think beyond current work practices
attitude	• should appreciate cross-disciplinary collaboration • should combine a distinctive preference for language quality with an appreciation of business goals and constraints

Tool Expert

rationale	The efficient use of a DSML will usually require a corresponding modelling tool. This implies either to select an existing tool or even to develop a new one. Both tasks demand for a professional with specific competencies.
professional training	• should have a formal education in Computer Science or Information Systems with emphasis on software engineering • should be trained in the design and implementation of modelling tools
information technology skills	• should be familiar with various paradigms for meta modelling tools • should have experience with using some of these tools and respective frameworks • should have an overview of relevant products
mental capabilities	• should be able and willing to mediate complex software technical issues to non-experts • should understand and appreciate the potential of DSML-based modelling tools
attitude	• should appreciate cross-disciplinary collaboration • should be interested in reflecting upon language – and the difference between natural language and formal languages

Graphic Artist

rationale	For a DSML to serve as an intuitive and convenient tool, its concrete syntax, i.e. its graphical notation can be of outstanding relevance. Designing an adequate graphical notation is not trivial, which de-

	mands for professional support by a specialized graphic artist.
professional training	• should have a formal education graphic design • should be trained in the design of iconographic symbols
information technology skills	• should have at least a basic understanding of modelling tools
mental capabilities	• should be able to understand the tasks to be supported by the DSML • should be able to empathize with prospective users
attitude	• should be interested in improving the use of software through professional graphical design • should be open for developing his skills into new directions

Manager

rationale	The investment into the development, use and maintenance of a DSML are often remarkable. Therefore, managerial approval is mandatory. The hierarchical level depends on the relevance of the investment for the respective organisation.
professional training	• should have formal education in Business and Administration or Information Systems • should be trained in the use of models and methods
information technology skills	• should be able to assess the preconditions, benefits and risks to be expected from using software tools • should be able to understand essential functions of complex software systems, especially of modelling tools
mental capabilities	• should be able to think beyond current work practices • should be able to appreciate the role of language for successful action
attitude	• should be willing to embrace progress enabled by new methods and tools, but keep a critical attitude with respect to feasibility and the effect on organisational competitiveness • should be willing to take risks, if he sees opportunities • should be willing to engage in cross-disciplinary collaboration • should be able and willing to endure periods of frustration caused e.g. by conflicting design goals • should be willing to take responsibility for critical decisions that are not agreed upon by everybody

Table 1: Role Model

There is a further role that is not mandatory for DSML design, but nevertheless deserves attention. The role of a *researcher* is important for two reasons. First, the development of a DSML is and undertaking that demands for creativity and innovation. This demand applies more or less to all roles listed above. Thus, a research attitude is a useful supplement to the skills assigned to these roles. Second, the development of DSML can be a research objective, especially in Information Systems. In this case, some of the above roles will be filled by re-

spectively qualified researchers. Hence, the role of a researcher would need further differentiation with respect to organizing cross-disciplinary research projects. Therefore, it is not described in the above schema. It is used within the process model as an abstraction to indicate where a research perspective could be beneficial.

6.2.2 The Macro Process

The introduction of a process model is usually accompanied by the advice not to interpret it as a strict sequence, but to allow for feedback loops. For the development of a DSML, this advice is of crucial importance. The contingent nature of the subject will often require stepping back to reconsider previous assumptions. Also, the macro process is not meant as the only way to structure the overall process (a proposition that could hardly be justified anyway), but as one approach that makes sense. Subsequently, each phase of the macro process shown in Figure 9 will be characterized by further details including a micro process. The graphical illustration of a micro process makes use of a circle to indicate that the phases of the process may need to be run through more than once. The phase 'Development of modelling tool' is included in the macro process, since it will often be necessary in order to make the DSML usable. It is, however, not described in detail, because that would require to focus on software engineering aspects and thus to leave the primary focus of this report.

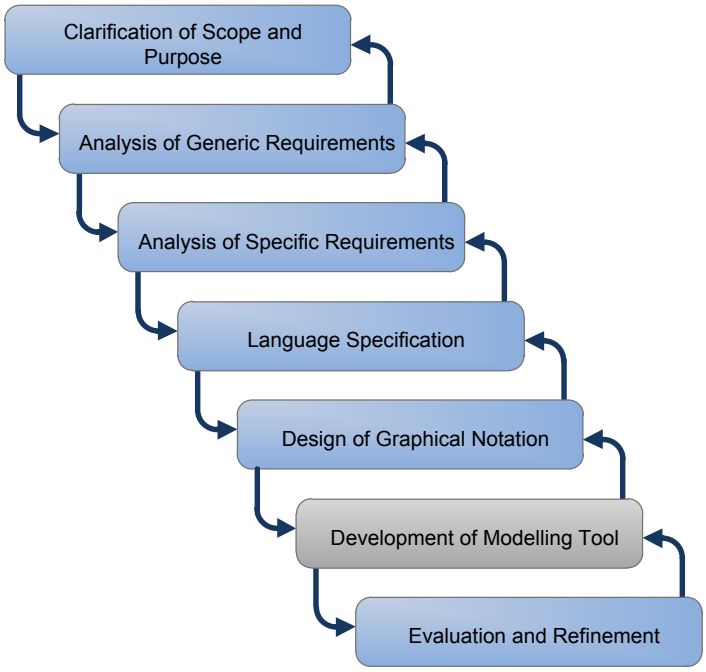

Figure 9: Illustration of Macro Process

6.2.3 Clarification of Scope and Purpose

The contingencies related to the conception and prospective use of DSML will usually demand for a clarification of scope and purpose. This includes the outline of a convincing motivation and rationale for the project.

Objectives: In the ideal case, the phase should produce a description of essential design objectives and a definition of the budget. However, due to the unavoidable contingencies, it may first require to move on to subsequent phases before this outcome can be realized.

Micro Process:

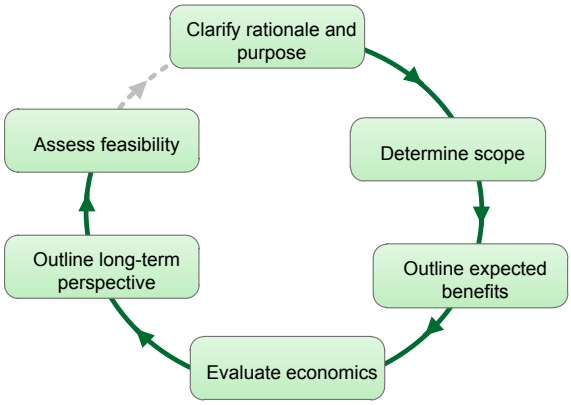

Figure 10: Micro Process 'Clarification of Scope and Purpose'

The rationale of a DSML will usually be related to generic prospects such as promoting productivity and quality. These need to be clarified with respect to the specific purpose and scope of the DSML to be developed. For this purpose, it is required to identify the modelling tasks that are to be addressed. To get a clearer picture it is important to look at previous projects that were aimed at these modelling tasks asking what impact a DSML would have had on performance and outcome. To outline the scope of the targeted DSML, modelling projects can be categorized with respect to modelling subject, complexity and frequency. The higher the complexity and frequency of modelling a certain subject, the higher the benefit to be expected from a respective DSML. Also, the prospective users' attitude and skills need to be accounted for. If they are reluctant or even reject the idea of a DSML, a negative impact on economics can be expected. Against this background, one can conduct a high-level assessment of benefits with respect to certain classes of modelling tasks and corresponding users. To evaluate the economics, costs and risks of introducing, using and maintaining the DSML need to be addressed. This requires accounting for the availability of development tools, the qualification of developers and users and changes of requirements to be expected in future times. Sometimes, the availability of a DSML will enable additional options such as the integration with other DSML or the use of models at run time. With respect to evaluating the investment into the development of a DSML, these options should be taken into account. Even if the economic perspective of a DSML seems to be promising, its feasibility may still be a challenge – especially if no previous experiences with similar projects exist. In these cases, it may be a good idea to start with a smaller scale project.

Input: profiles of developers and users, outline of modelling scenarios, reports on previous modelling projects.

Participants: Manager, Business Analyst, User, Domain Expert, Language Designer

Risks: Lack of information and knowledge may contribute to an inappropriate outline of the DSML's intended scope and to a misleading assessment of its benefits. To counter this risk, it is crucial to include respectively qualified experts.

Results: preliminary project plan, budget at least for first project phase, assignment of personnel at least to first phase, including external service providers, outline of long-term perspective

6.2.4 Analysis of Generic Requirements

The conception of DSML should account for generic requirements. They apply to every DSML, however, with different weight. Also, they may need to be adapted to a particular DSML.

Objectives: Specify and generic requirements and create corresponding documentation.

Micro Process:

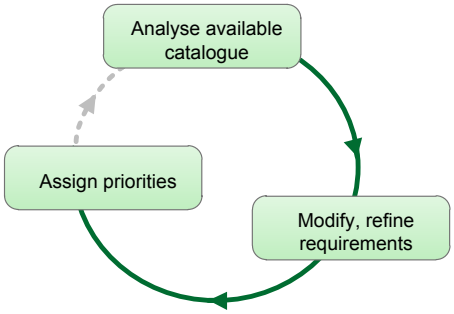

Figure 11: Micro Process 'Analysis of Generic Requirements'

There are not many catalogues of generic requirements available. Also, with respect to the fact that the field has not reached a mature state yet, not all proposals need to be convincing. Therefore, the analysis of available catalogues – like the one presented in 3 – should pay special attention to the rationale given for each requirement.

Input: Existing catalogue(s) of and publications on generic requirements for DSML.

Participants: Domain Expert, User, Language Designer, Tool Expert

Risks: If no appropriate catalogue is available, the development of generic requirements is a cumbersome activity that implies the risk to miss requirements. Even if one can build on an existing catalogue, there is no guarantee that it is comprehensive.

Results: Catalogue of generic requirements. Each requirement should be described and justified with respect to the purpose of the DSML. Also, each requirement should be characterized with respect to its relevance.

6.2.5 Analysis of Specific Requirements

As already elucidated above, it is probably the most challenging peculiarity of developing a DSML that specific requirements can often not be analysed in a straightforward way. There-

fore, there is need for an approach that does not rely on merely analysing existing modelling tasks and asking users for their expectations.

Objectives: Develop a comprehensive list of specific requirements.

Micro Process:

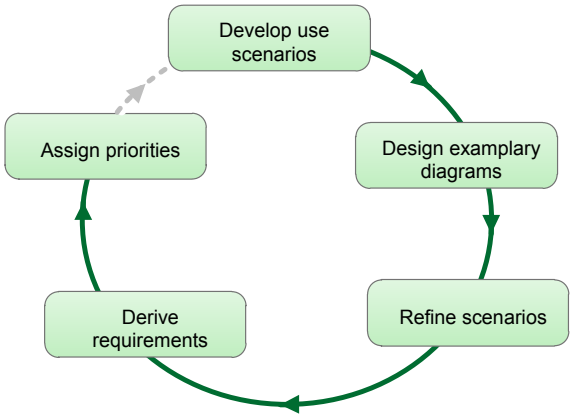

Figure 12: Micro Process 'Analysis of Specific Requirements'

To analyse specific requirements, users, business analysts and language designers need to be supported with developing a clear idea of what they may expect from the DSML. Our experience with developing DSML suggests that one approach is especially suited for this purpose. Based on use scenarios that are developed with respect to previous and future tasks, the potential use of the DSML is illustrated through the design of preliminary diagrams (for a definition of the terms 'diagram' and 'diagram type' see Frank, 2011b). These serve as a medium for further refining the use scenarios. To support the derivation of specific requirements, it has proven successful for us to describe the scenarios in a certain structure.

For developing use scenarios relevant modelling scenarios from the past should be identified and described. In addition to that further possible use scenarios may be developed. This can be promoted by presenting rudimentary scenarios which are then further refined. Each scenario is related to a certain diagram type. To get an idea what information should be represented in respective diagrams, one can start with a rudimentary graphical representation and then develop a list of questions that are related to the diagram. With respect to preparing for a corresponding modelling tool, it is helpful to specify for each question whether it can be answered by a machine (**A**), by a human only (**H**), or in a partially automated way (**P**). Note that the stages 'develop use scenarios', 'design exemplary diagrams' and 'refine scenarios' are interweaved. Each diagram type should be clearly described with respect to its purpose and its key concepts. Furthermore, it should be related to other diagram types that might supplement it with respect to specific purposes. The actual example diagrams have an important function especially for novice users, since they provide an illustration of what they might expect from the intended DSML. Therefore, designing exemplary diagrams should account for an illustrative graphical representation – even though it will be replaced by a

more professional notation later on. Based on the example diagram and the extensible list of questions, all participants – supported by domain experts and language designers – are supposed to commonly develop specific requirements. If a requirement poses a substantial challenge to language design or the realisation of a corresponding tool, it should be marked as such. Hence, a challenge is related to design decisions that result from requirements. They need to be addressed at the design stage at the latest and may result in relaxing corresponding requirements.

The following examples illustrate this approach. The numbers used for requirements and challenges correspond to those in the original source, too.

Organisational Chart

This diagram type corresponds to prevalent graphical representations of organisational structures. It depicts organisational units as well as associations between them. Note that – different from typical organisational charts – organisational charts expressed through the MEMO OrgML feature a precise semantics of relationships between organisational units. It can also be applied to represent the temporal organisational structure of a project.

Purpose: Versatile representation of organisational structure to support the analysis of division of labour, decision and control mechanisms etc. as well as their redesign. At the same time: a representation/documentation of the organisational structure as part of organisational knowledge management, e.g. to support new employees or external consultants with getting an appropriate picture of an organisation. Possible questions to be addressed – depending on the level of detail features by an organisational chart:

- What is the average span of control (number of directly subordinated units)? **A**
- What is the percentage of positions that require at least a bachelor degree? **A**
- What is the percentage of positions that constitute the organisation's core competence? **A**
- Which organisational units suffer from poorly qualified employees? **A**
- Are there conflicting responsibilities of organisational units? **P**
- Are there similar positions that could be merged? **P**
- Does is make sense to reduce or widen the span of control? **H**
- What kind of impact would this have on the qualification required for employees? **H**
- What are the effects of organisation culture on work coordination? **H**

Potentially integrated with: Human Resource Management, ERP, IT Management

Key concepts: organisational unit, various kinds of relationships

Example: The example in figure E1 illustrates an organisational chart of a strategic business unit. The edges between the symbols representing organisational units express aggregations.

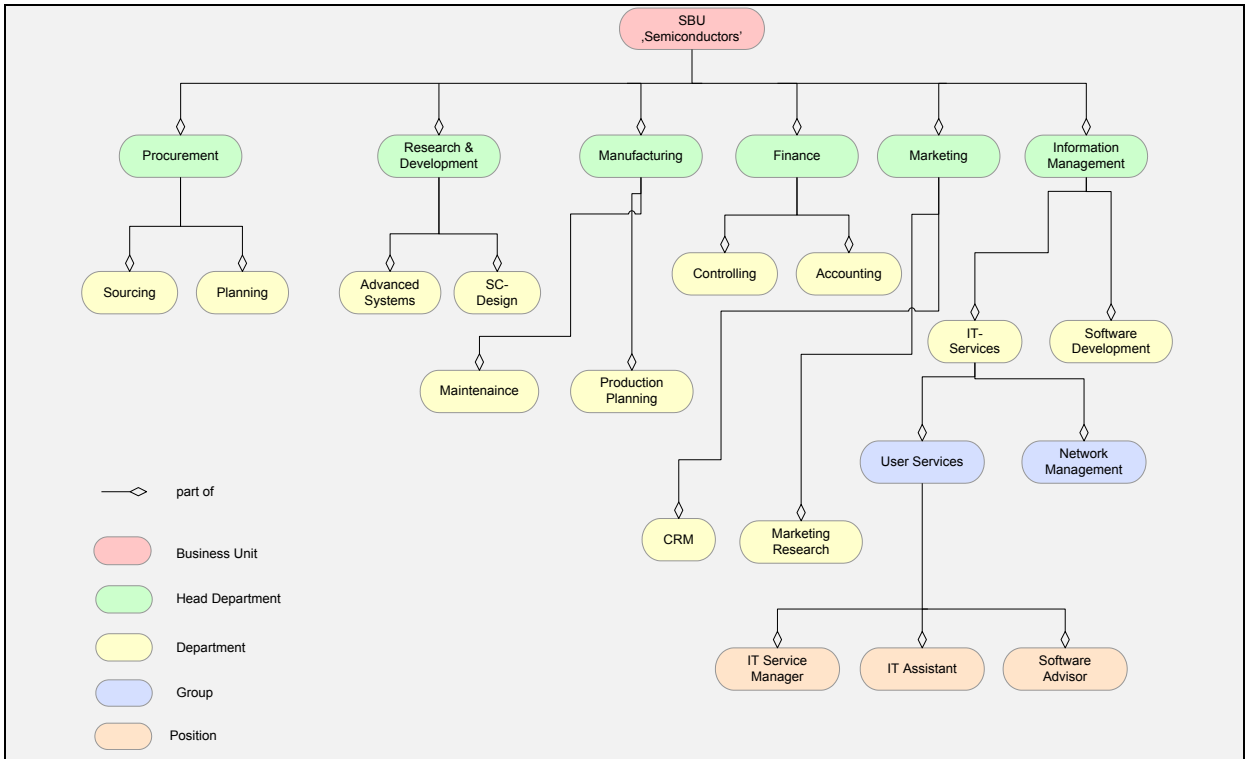

Figure E1: Example of Organisational Chart

Specific Requirements

Requirement SR1: To allow for elaborate analyses, it should be possible to describe organisational units in a differentiated way. This includes concepts to describe formal qualification, skills, tasks, responsibilities etc.

Requirement SR2: An organisational chart represents types of organisational units. Therefore, it would not be appropriate to assign instance-level features, e.g. the actual number of employees within a department to a particular type. However, in those cases, where there is only one instance of a type (e.g. 'Marketing Department'), this kind of semantic overload would not be too harmful – at least from a pragmatic point of view. Therefore, the OrgML should provide concepts that allow for a precise description, which can be combined with a more pragmatic, overloaded representation.

Requirement SR3: Sometimes, certain assertions do not only apply to one organisational unit (or role or committee), but to more. It may be, for example, that various organisational units are in charge of a certain process. To elegantly deal with this kind of variance, there is need for concepts that allow for expressing abstractions over a set of organisational units (or roles and committees respectively).

Requirement SR4: While some organisational units will usually occur only once, others – this is typically the case for positions – can come with multiple instances. Firstly, there is need to allow for differentiating between multiplicity constraints ("There must not be more than one marketing department.", "There must be one and exactly one head of the marketing department.") and actual numbers ("The current headcount of the marketing department is 26."). The notation should support a clear differentiation of these two meanings.

Challenges

Challenge C1: The representation of actual headcounts – or the actual number of positions of a certain kind – faces a conflict. On the one hand, a model should feature a level of abstraction

that makes it widely independent from state changes on the instance level. On the other hand, information about these numbers may be regarded as relevant for an organisational chart. In case the model is managed by a tool which is integrated with a corresponding information system, these numbers could be updated automatically whenever changes occur. Otherwise it will probably be better to use numbers that are marked as estimations.

6.2.6 Business Process Association Diagram or Business Process Map

Business process diagrams serve to model one business process type only. Sometimes there is need for looking at a group of business process types or even all business process types of a company at the same time. Business process association diagrams allow for representing several business process types and the relationships that exist between them. They are sometimes referred to as 'business process maps'. Note that relationships between business process types may recommend referring to additional models, e.g. resource models. The business process types included in a business process map can be decomposed into decomposition diagrams, which allows for analysing commonalities of business process types on a more detailed level.

Purpose: Representation of various business process types and various types of relationships between them, such as support, competition, dependence – in order to foster analysis and (re-) design of business process systems, i.e. sets of interrelated business process types. Potential questions include:

- Are there process types that compete for the same resources? **A**
- Are there process types that target conflicting goals? **P**
- What are the most important processes? **P**
- Is there need for reducing the number of processes? **H**
- Are there any problematic inter-process dependencies? **P**
- Is there need for improving inter-process coordination? **P**

Key concepts: business process, goal, various types of relationships

Example: Figure E2 shows a simplified process association diagram. It refers to organisational units, thereby supporting the analysis of organisational conflicts.

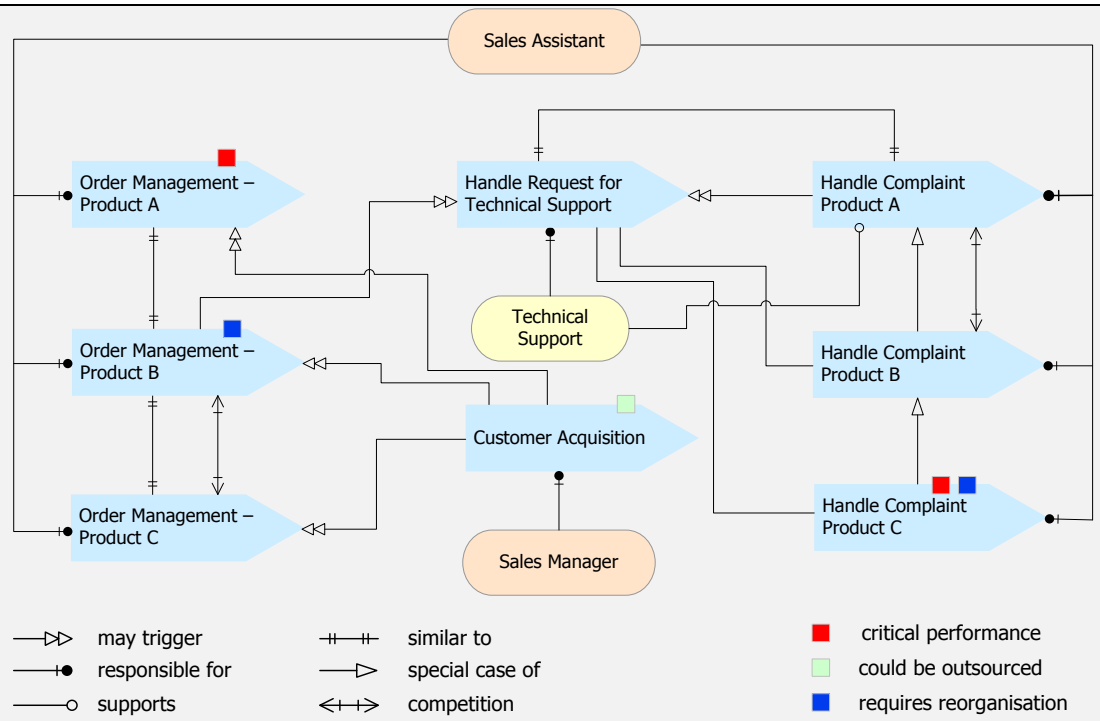

Figure E2: Illustration of business process association diagram

Specific Requirements

Requirement SR5: In addition to a set of predefined relationship types, it should be possible to define further relationship types (corresponds to **Requirement A2**).

Challenges

Challenge C2: Again, specialisation relationships mark a challenge. The example in figure E2 gives an impression of the benefit that could be generated by specialisation: If all features that are specified for a superordinate process type (e.g. supported by 'Technical Support' for 'Handle Complaint Product A') would be inherited to the specialized process types, this would not only contribute to a model's clarity, but also foster its maintainability. At the same time, it seems extremely unlikely for most cases that the quest for substitutability could be satisfied. Probably, the only option will be to aim at a relaxed concept of specialisation.

Input: previous modelling scenarios; collections of complex analysis and decision tasks, especially those that require accounting for multiple perspectives

Participants: Business Analyst, Domain Expert, Language Designer, Manager, Tool Expert, User; optional: Graphic Artist

Risks: If all participants lack a background in modelling with DSML, it may be difficult or even frustrating to develop appropriate scenarios and corresponding diagrams. This, however, is crucial for the analysis of requirements. While the scenarios and corresponding exemplary diagrams are an important instrument to analyse requirements by illustrating the purpose of the prospective DSML, they may also compromise the analysis of requirements by restricting the scope to particular aspects. Therefore, selecting scenarios and creating ex-

emplary diagrams require experienced domain experts and language designers that have an idea of how the targeted DSML could look like, but are open minded enough to appreciate suggestions and requests made by other participants.

Results: documentation of specific requirements and corresponding rationale; documentation of specific challenges

6.2.7 Language Specification

The specification of the abstract syntax and semantics is the pivotal part of designing a DSML. It requires accounting for the range of potential applications. It will usually include various design decisions, some of which are common in conceptual modelling while others are specific for the design of meta models. The specification of a meta model implies the selection of a meta modelling language. For corresponding requirements see section 6.1 and Frank (2011a).

Objectives: Specification and documentation of meta model and corresponding constraints.

Micro Process:

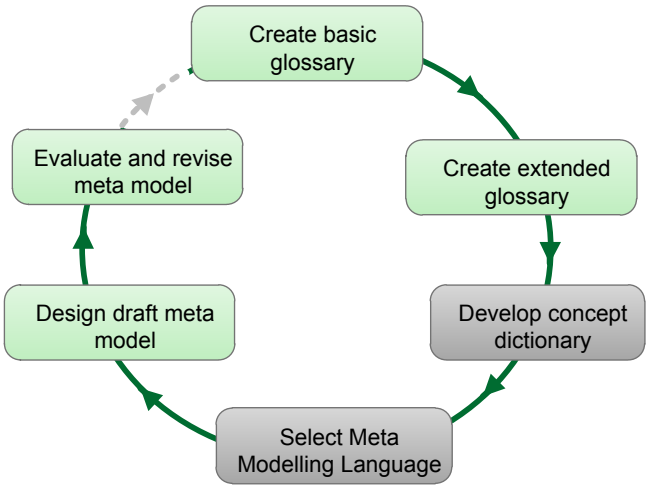

Figure 13: Micro Process: 'Language Specification'

The specification of a DSML is directed towards reconstructing concepts of the respective domain of discourse. For this reason, it makes sense to first develop a glossary with key terms. At its first development step that we refer to as 'basic', the glossary is a dictionary of terms with corresponding descriptions. It may be presented either in alphabetic order and/or with respect to semantic categories, such as 'Organisation Structure', 'Processes', 'Resources' etc. The following excerpt illustrates structure and content of a basic glossary.

Term	Description
Board	A board is an organisational unit with command line authority. A board can include positions and roles. Note that within the language the concept is used in a wider sense than in *board of directors* (which would be a special type of board).
Committee	A committee consists of group of people. Committees can address a wide range of different

	purposes. Some may make decisions that are mandatory for other organisational units. Others may serve counselling purposes only. In any case, a committee does not include positions, but only roles. The challenges that are related to expressing different levels of abstraction are dealt with by using intrinsic features. Note that the graphical notation may be subject of further revisions.
Department	A department is an organisational unit, which will usually include further organisational units. It may be part of a superordinate organisational unit.
Organisational Unit	An organisational unit is a part of an organisation (institutional) that reflects a permanent principle of the division of labour. An organisational unit may contain other organisational units. The definition of organisational units can be based on functional aspects (e.g. 'Finance', 'Production', 'Marketing' ...), on objects (e.g. 'trucks', 'sport cars' 'power train'), market-oriented (e.g. 'North America', 'Europe', 'consumer', 'reseller' ...) or combinations of these. Usually there is one position that is in charge of an organisational unit.
Organisation	An organisation is a goal-oriented social or socio-technical system – like a business firm, a non-profit organisation, public administration etc. While it could be regarded as the top-level organisational unit, it makes sense to define it as a special concept, because it has features that do not apply to organisational units.
Position	A position is the smallest organisational unit. It does not contain any other positions. Usually, a position is assigned to one employee. There are, however, exceptions of this rule ("job sharing").
Role	A role is defined by a set of responsibilities, the actor who fills the role, has to perform. It is usually less formal than a position - and it is orthogonal to position: An employee who holds a position can also fill one or multiple roles at the same time.

Table 2: Illustration of Glossary

The basic glossary serves as a collection of terms that are used in the targeted domain of discourse. That does not mean, however, that each of these terms is suited to be included in the DSML. Instead, it needs to be decided whether a term should be reconstructed as part of the intended DSML or whether it should rather be specified with the DSML. To support this decision, we use an extended structure of the glossary that reflects key decision criteria. Currently, I am working on corresponding guidelines, which will be published shortly. In this report, only a few of them are used to illustrate key ideas. The conception of DSML that we favour is aimed at a covering a wider range of (re-) use. In other words: Usually we do not develop a DSML for just one project as it is suggested e.g. by Kelly and Tolvanen (2008). Therefore, it is important that the *semantics* of a language concept is *invariant* throughout the entire domain and in time. To promote comprehensibility and usability of a DSML it should be avoided to include concepts that are not needed. Hence, each collected term should be checked for its *relevance*. A language concept is – usually – intended as an abstraction over types. This results in two demands: First, the semantics of the respective instances should vary – otherwise it would literally not make much sense to bother with instantiations. Second, it should be checked whether it corresponds to an intuitive understanding that possible instances are perceived as types. The following example illustrates a possible glossary entry for the term 'Organisational Unit'. '+' indicates that the term clearly fulfils the corresponding criterion. 'o' expresses that it fulfils the criterion to a satisfactory degree, while '-' indicates that it fails to satisfy the corresponding criterion. 'c' means that – with respect to

the considered criterion – the use of the term is contingent, i.e. in some cases it fulfils the criterion, in others it does not. Note that there are no precise guidelines based on 'hard facts' for evaluating domain terms against the suggested criteria. Instead, it is recommended to make use of discursive evaluation that is aimed at developing reasons that are regarded as convincing by the participants.

Organisational Unit		
An organisational unit is a part of an organisation that reflects a permanent principle of the division of labour. An organisational unit may contain other organisational units.		
Example instantiations	'Division Electronic Devices', 'Marketing Department', 'Car Manufacturing Plant', 'Human Resources Department'	
a) invariant semantics	The term is used on a high level of abstraction. The essence of its semantics should not vary substantially.	+
b) relevance	This is a key term for describing organisation structures.	+
c) variance of type semantics	The semantics of types of organisational units can vary significantly.	+
d) instance as type intuitive	At least some of the potential instances will be regarded as types almost intuitively, e.g. 'Department', 'Division'. Other potential instances, such as 'Marketing Department', 'Consumer Electronics Division', will probably not be regarded as types by many. Hence, the final assessment of this criterion depends on the recommended instantiation.	+ c

Table 3: Structure for Checking Suitability of Potential Language Concept

Subsequently to assessing the collected terms, it is required to decide for each term whether or not to include it in the DSML. This decision recommends reflecting upon the intended scope, i.e. the targeted domain, of the language. As a consequence, one may decide to exclude a term from the language or to narrow the scope in order to an extent where a particular term is characterized by invariant semantics. This step results in a revised version of the extended glossary. Before specifying the language with a meta model, a concept dictionary can be created. It serves two purposes: On the one hand, it should prepare for the construction of the meta model; on the other hand, it serves as a documentation of the terms specified in the meta model. The structure of a concept dictionary reflects specific design decision to be made with the construction of meta models. For this purpose, it does not only represent the collection of attributes and associations that are required for the specification of a concept. In addition to that it proposes a structure to support a more elaborate description. Attributes are separated into two groups. Attributes on the type level are supposed to describe characteristic features of a type that is instantiated from the respective meta type (in the example below: from 'OrganisationalUnit') – independent from its own instances. This is different with the category 'Attributes with references to instance level'. It serves to group those attributes of a type that can be initialized only by accounting for its instances. The following two examples serve to illustrate this category: 'numberOfPositions' is an attribute that requires counting the number of actually assigned positions. The attribute 'averageCostPerPosition' recommends collecting data about actual costs for each of the respective positions.

With respect to the design of a corresponding modelling tool it is useful to specify how the required instance-level data could be determined. The predefined structure of the dictionary allows to differentiate between two approaches: 'Obtainable' serves to express that corresponding values could be (not: have to be) obtained from external sources, e.g. from an ERP system. That would require caring for an appropriate interface. 'Derivable' indicates that a value might be calculated from other parts of a model. Take, for instance, the attribute 'numberOfPositions' in the below example. If within the model of an organization, an organizational unit is assigned all its positions (as type), and each position type is characterized by the number of its instances, then the total number of positions assigned to an organizational unit can be calculated, i.e. derived. In addition to these two categories, an attribute can also be marked as 'intrinsic' or 'simulation'. An intrinsic attribute is not an attribute of a type in a strict sense. It is, however, characteristic for a type in the sense that it applies to all of its instances. In other words: It is intrinsic to the concept represented by the type. For example: Every business process instance has a starting and a termination time. However, these features cannot be represented as attributes of a type, because they do not apply to the type. Marking such a feature as intrinsic dissolves this dilemma by specifying that it can be initialized on the instance-level only. For a detailed description of intrinsic features see Frank (2011a, p. 19 f.). 'Simulation' indicates that an attribute might be used for simulation purpose. For instance: A certain position type, e.g. 'Sales Agent', could be assigned the attribute 'availability' to represent an assumption about the availability of corresponding employees throughout a reference period. In addition to attributes, associations can also be described both on type level and with reference to the instance level (the latter is not include in the below example). Associations can be intrinsic, too. If associated concepts are marked as intrinsic, they can be marked as 'obtainable'. If the meta type 'Position' is associated to the intrinsic type 'Employee', it is possible to express that instances from 'Employee' can be obtained from an external source. Finally, each entry of the concept dictionary is supplemented with corresponding constraints in natural language. There may be concepts in the relevant domain of discourse that are relevant for certain kinds of analysis, but that bulk against formalization. To protect against dysfunctional effects (see 2.3), the description of respective concepts in the concept dictionary should include corresponding remarks. In the dictionary shown below, this is the case for the concept 'CoporateRelevance'. Figure 15 shows how to explicitly account for limitations of formalisation in a meta model.

The below example of only one entry illustrates that a concept dictionary can grow to a remarkable size. Creating – and maintaining – it can therefore require a substantial effort. It is a particular challenge to synchronize it with corresponding parts of the glossary and the meta model. Therefore, creating a concept dictionary will usually make sense only, if there is a tool that synchronizes corresponding parts of the dictionary and the meta model.

OrganisationalUnit	This is a key abstraction of the language. It can be instantiated in various ways. On the highest level of aggregation, it can be instantiated into the type of the entire organisation. To differentiate types according to the technical language used in an application domain, it is possible to refer to a corresponding type description by initialising the attribute *orgLevel*. Often, there will be only one instance of a type that is instantiated from this meta type.
Attributes on type level	
name	Allows for assigning a type name. Note that the name of this type will usually be the same as the corresponding instance name since there will often be only one instance.
mission	Serves to describe the mission and the responsibilities.
orgDimension	Serves to specify the dimension the definition of the type of organisational unit is based on. If no value is specified, the value of the *OrganisationalUnit*, this *OrganisationalUnit* is part of, applies. The value specified for *OrganisationalUnit* must not violate the value specified for the corresponding *Organisation*.
staffUnit	If this attribute is set to #true, the corresponding organisational unit type is regarded as staff unit. The semantics of staff units varies to a remarkable degree. In general, a staff unit is regarded to serve counselling purposes. Hence, the staff experts are charged with gathering and summarizing information and giving technical assistance to generalist managers who are responsible for making final decisions. Staff units are not included in the regular line of command. Hence, a position that is in command of a staff unit cannot be in command of another organisational unit that is not a staff unit.
corporateRelevance	Allows for expressing the relevance the organisational unit has for an organisation's competitiveness. The assigned value represents the degree of relevance, e.g. 0: no need; 1: could do without; 2: needed; 3: essential. Note that relevance may include multiple aspects. Therefore, mapping it to an ordinal scale can lead to a loss of information, which may produce dysfunctional effects when using it without critical reflection.
subjectOfOutsourcing	Allows for expressing whether the corresponding type of organisational unit should be outsourced. The assigned value expresses the degree of urgency, e.g. 0: no need; 2: should be considered; 3: seems reasonable; 4: urgent need.
subjectOfReorganisation	Allows for expressing whether the corresponding type of organisational unit should be reorganized. The assigned value represents the degree of urgency, e.g. 0: no need; 2: should be considered; 3: seems reasonable; 4: urgent need. While both attributes, *subjectOfOutsourcing and subjectOfReorganisation,* may reflect the actual performance of the *OrganisationalUnit,* hence, an instance level feature, they are still assigned to the type level, because they are related to a potential change of the organisational structure.
Attributes with reference to instance level	
<<derivable>> *averageSpan*	This attribute serves to represent the average span of control across all positions of the *OrganisationalUnit* that fulfil managerial functions, i.e. that are superior to other positions.
<<obtainable>> *<<derivable>>* *numberOfEmployees*	The actual number of employees working for the organisational unit.
<<obtainable>> *<<derivable>>*	The percentage of females among the employees.

genderRatio	
<<obtainable>> *<<derivable>>* *numberOfPositions*	The actual number of positions assigned to the organisational unit. Often, this number will be the same as for the attribute employees. But it does not have to be, since positions may be vacant or filled by more than one employee. It can be obtained from an external system, assigned manually or calculated from the positions assigned to this organisational unit and all its subunits.
<<obtainable>> *<<derivable>>* *fluctuation*	The fluctuation of all positions assigned to this organisational unit within a certain period (e.g. a year). There are various formula for defining fluctuation. At best, this feature allows for selecting from a list of corresponding concepts. The actual value could be provided by a corresponding information system, e.g. a HRM system.
<<obtainable>> *performance*	The value of a performance indicator on an ordinary scale, e.g. 0: critical; 1: satisfactory; 2: outstanding. The value can either be calculated from data provided by e.g. an ERP system or assigned manually.
<<obtainable>> *<<derivable>>* *averageCostPerPosition*	The average cost of a filled position assigned to this *OrganisationalUnit*. This value can be provided by an external system. It can also be calculated from the values assigned to the positions of the organisational unit (provided the costs are specified for all positions).
<<obtainable>> *shareOfPersonnelCost*	The share of costs for personnel from the entire costs assigned to an organisational unit – to be obtained from an external system.
<<obtainable>> *<<derivable>>* *averageAge*	The average age of all employees of this organisational unit– also to be obtained from an external system.
<<obtainable>> *<<derivable>>* *shareOfUnderPerformers*	The share of positions that are assessed as not performing satisfactorily. This value can be aggregated from corresponding values of the included organisational units, for which in turn the value would be calculated from corresponding values of the assigned position types or positions.
Associations on type level	
composed_of	with *OrganisationalUnit.* A type of organisational unit can be composed of zero or more (0,*) other organisational unit types. On the other hand, an organisational unit type can be part of zero or one other organisational unit type. If the attribute *orgLevel* is specified for the types of organisational unit that are involved in this relationship, the level of the composed type must be higher than the levels of its parts.
hosts *(corresponds to hosted_by for Committee)*	with *Committee.* A type of organisational unit can host zero or more (0,*) committee types. On the other hand, a committee unit type can be hosted by one (default) or more than one organisational unit type.
includes *(corresponds to part_of for PositionShare)*	with *PositionShare.* A type of organisational unit can include zero or more (0,*) *PositionShare*. On the other hand, a *PositionShare* can be part of one or more organisational unit types.
assigns *(corresponds to assigned_to for Role)*	with *Role.* A type of organisational unit can assign zero or more (0,*) role types. On the other hand, a role type can be part of zero or more organisational unit types.
characterizedAs *(*corresponds to *characterizes* with *LocalUnitType)*	With *LocalUnitType*. An *OrganisationalUnit* has zero to one *LocalUnitType*.
supervisedBy	With *Position*, *Role* or *Board*. Multiplicity is zero to many on both sides.

*(*corresponds to *superior_of* for *Position, Role* or *Board)*		
subordinated_to *(*corresponds to *superior_of* for *Position, Role* or *Board)*		With *Position*, *Role* or *Board*. This relationship is used in cases where no particular definition of superiority exists or matters. Multiplicity is zero to many on both sides. The number of superiors must not exceed the value of *maxLineOfCommand* within the corresponding *Organisation*.
functionalSubordinated_to *(*corresponds to *functionalSuperior_of* for *Position, Role* or *Board)*		With *Position*, *Role* or *Board*. Multiplicity is zero to many on both sides.
objectSubordindated_to *(*corresponds to *objectSuperior_of* for *Position, Role* or *Board)*		With *Position*, *Role* or *Board*. Multiplicity is zero to many on both sides.
disciplinarySubordinated_to *(*corresponds to *disciplinarySuperior_of* for *Position, Role* or *Board)*		With *Position*, *Role* or *Board*. Multiplicity is zero to many on both sides.
part_of		With *Organisation*. An *OrganisationalUnit* can be assigned to zero or one *Organisation*.
Constraints		
C8	Exclude staff units from regular line of command	If the attribute *staffUnit* is specified as #true, it is not possible that the *OrganisationalUnit* includes any other *OrganisationalUnit* that has this attribute set to #false. Furthermore, it is not possible that a *Position* that is in command of this *OrganisationalUnit* is in command of any *OrganisationalUnit* other than an included staff unit.
C3	No cyclic *composed_of* relationships	An organisational unit type A must not be composed of a further organisational unit type B, which in turn is composed of A (Constraint C3).
C1	Not part of unit with lower or equal level	An *OrganisationalUnit* must not be part of another *OrganisationalUnit* that has a level assigned (through an associated *LocalUnitType*), which is lower than or equal to its own level (Constraint C1). Note that this constraint applies only, if the *OrganisationalUnit*, it refers to are associated with an initialized instance of *LocalUnitType*.
C2	consistent use of *orgDimension*	If the attribute *orgDimension* within *Organisation* is specified, then the corresponding attribute of *OrganisationalUnit* must have a compliant value. If, e.g. *orgDimension* is specified as #functional, it is not possible for an *OrganisationalUnit* to use #matrix.
C7	No more superiors than *maxLineOfCommand*	The sum of superiors (functional, object, disciplinary) must not exceed the value of *maxLineOfCommand* specified for the corresponding *Organisation*.
C5	No joint use of generic and specific superiority	If a *subordinated_to* relationship is specified for an *OrganisationalUnit*, there must be not further special subordinate relationship for this *OrganisationalUnit*.

Table 4: Exemplary Representation of Concept in Concept Dictionary

Usually, models constructed with DSML focus on types, i.e. the so called M_1 level, and not on instances. This is for a good reason: Often, a model is supposed to provide an abstraction that is widely invariant over time. In many cases, instances are not invariant. There are, however, exceptions to this rule. In case the following presuppositions apply to a modelling scenario, representing instances in a model may be an option:

- The purpose of a model recommends accounting for instances.
- Abstracting instances to the type level would not fit the intended applications of a model anymore.
- The existence and the relevant state of an instance are stable throughout the intended lifetime of a model.

Usually, the question whether there is need for modelling instances can be addressed by analysing the exemplary diagrams developed during requirements analysis. Possible examples of instances that could be included into models are cities, countries, organisational units (e.g. 'Marketing Department') or organisations (e.g. a particular company). With respect to designing a meta model, these considerations imply a serious problem: The conception of a meta model is usually related to the idea that all concepts of a meta model are located on the M_1 level. Corresponding to that, concepts of a meta modelling language are located on the M_2 level. In order to include types into a modelling language, it is required that a meta modelling language allows for expressing that – or, in other words, that a meta model can be overloaded with respect to the level of abstraction it represents. While most meta modelling languages do not allow for specifying types, it is nevertheless recommended to identify potential candidates that could/should be included in the intended DSML.

The next step concerns the selection of a meta modelling language and a corresponding meta modelling tool. It is optional: In some cases there will be no choice anymore because the use of a certain meta modelling language is mandatory. Selecting a meta modelling language recommends considering and assessing respective requirements. For a comprehensive discussion of requirements and a comparative assessment of meta modelling languages see Frank (2011a, p. 3 f.). In any case, it is recommended to use a graphical notation with the meta modelling language that allows to clearly distinguish a meta model from prevalent models on the M_1 level at first sight. In order to promote abstraction and reuse, it can be useful to introduce specific types for specifying attributes of meta types. Different from the types that are built in the meta modelling language, these customized types require an explicit specification. Therefore it is a good idea to distinguish the representation of specific types from that of generic types. In the excerpt shown in Figure 14 specific types are printed in boldface. If the specification of a concept goes along with semantic reductions, there should be at least a comment in the meta model that points at this potential problem. In addition to that a corresponding warning can also build into the language specification. In that case, the language user will be forced to account for possible dysfunctional effects. Figure 15 shows the excerpt of a meta model to specify a DSML for modelling indicator systems. To prevent users from the careless application of indicators, the definition of a certain indicator type requires the user to explicitly comment on intended effects and ('presumedOrganizationalImpulse') and possible dysfunctional effects ('potentialDysfunctionalConsequences').

The MEMO meta modelling language (Frank, 2011a) does not only provide a corresponding graphical notation, it also allows for representing intrinsic features and for marking features

as obtainable or derivable (see section 6.1). Figure 14 shows an excerpt of a meta model to illustrate the use of the MEMO MML.

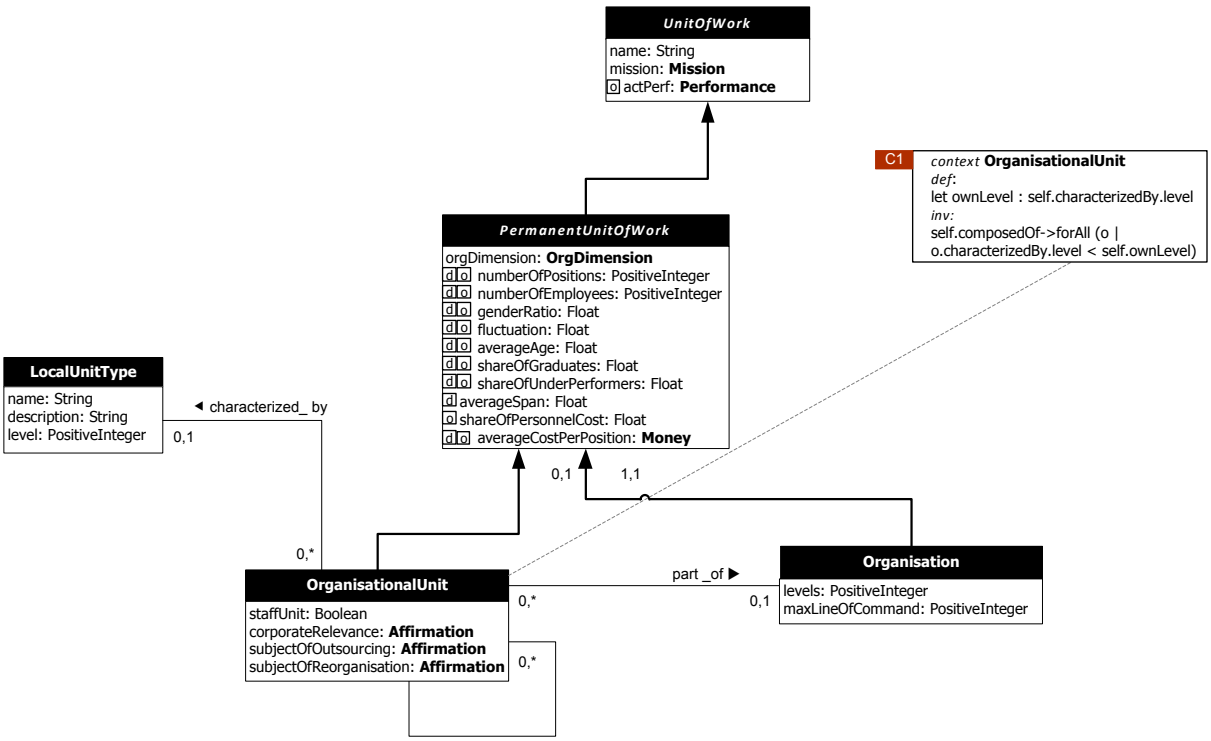

Figure 14: Excerpt of a meta model specified with the MEMO MML

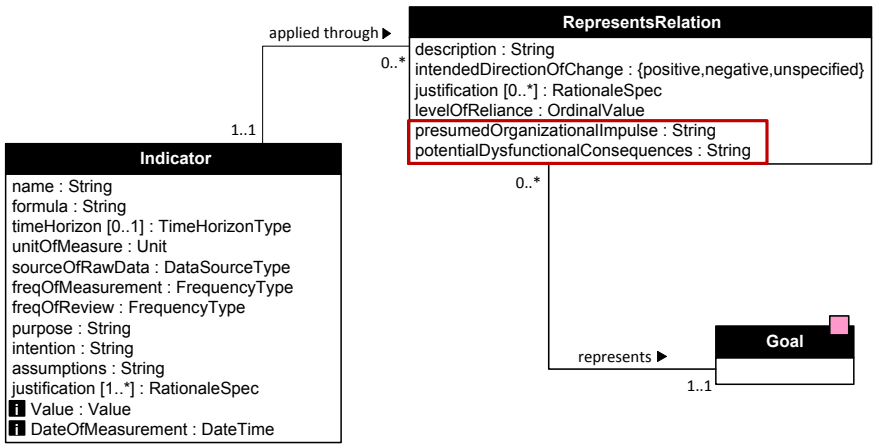

Figure 15: Illustration of Enforcing a Critical Use of a DSML through Respective Attributes – adapted from Strecker et al. (2011)

Like with any other complex abstraction, the design of a meta model will usually require demanding decisions. They will often relate to conflicting design goals. All design decisions that do not seem to be trivial should be documented using a common structure, e.g. 'problem description', 'design alternatives', 'selected alternative', 'rationale'. In addition to design decisions, the specification of a meta model is also based on principle attitudes and styles of the language designer. For instance, some designers prefer to specify as much of a DSML's se-

mantics through the abstract syntax, while other tend to keep the abstract syntax simple and represent semantics in additional constraints. The example in Figure 16 illustrates these two specification styles. Both styles have specific advantages and shortcomings. By putting more emphasis on additional constraints, a meta model can be kept simpler. Also, representing semantic constraints on a syntactical level will often require to introduce artificial concepts, like 'SimpleAttribute' in the below example. Therefore, aiming at a lean meta model with additional constraints may – at first sight – contribute to improved comprehensibility. However, additional constraints should be specified in a formal language such as the OCL. For many observers, a larger number of formal constraints will not improve the readability of a meta model. With respect to implementation, this style delegates more responsibility to programmers. Usually, the meta types of a meta model will be represented as classes in a corresponding model editor. In the case of emphasizing syntactical specification, the correctness of larger parts of a model can be checked on the class level. In the other case, constraints have to be implemented and checked with respect to particular instance states. Therefore, with respect to integrity, there is good reason to avoid specifying "types" through instance states. However, at the same time, introducing additional meta types to emphasize syntactical specification may threaten integrity: Certain changes applied to a model are likely to require class migration – e.g. migrating an instance representing an attribute from the class 'SimpleAttribute' to the class 'Attribute'. Class migration is not only costly, but also risky. Due to conflicting requirements, choosing a particular specification style is also a matter of subjective preferences. To guide the reader of a meta model with a better understanding, it is a good idea to briefly comment on the preferred specification style.

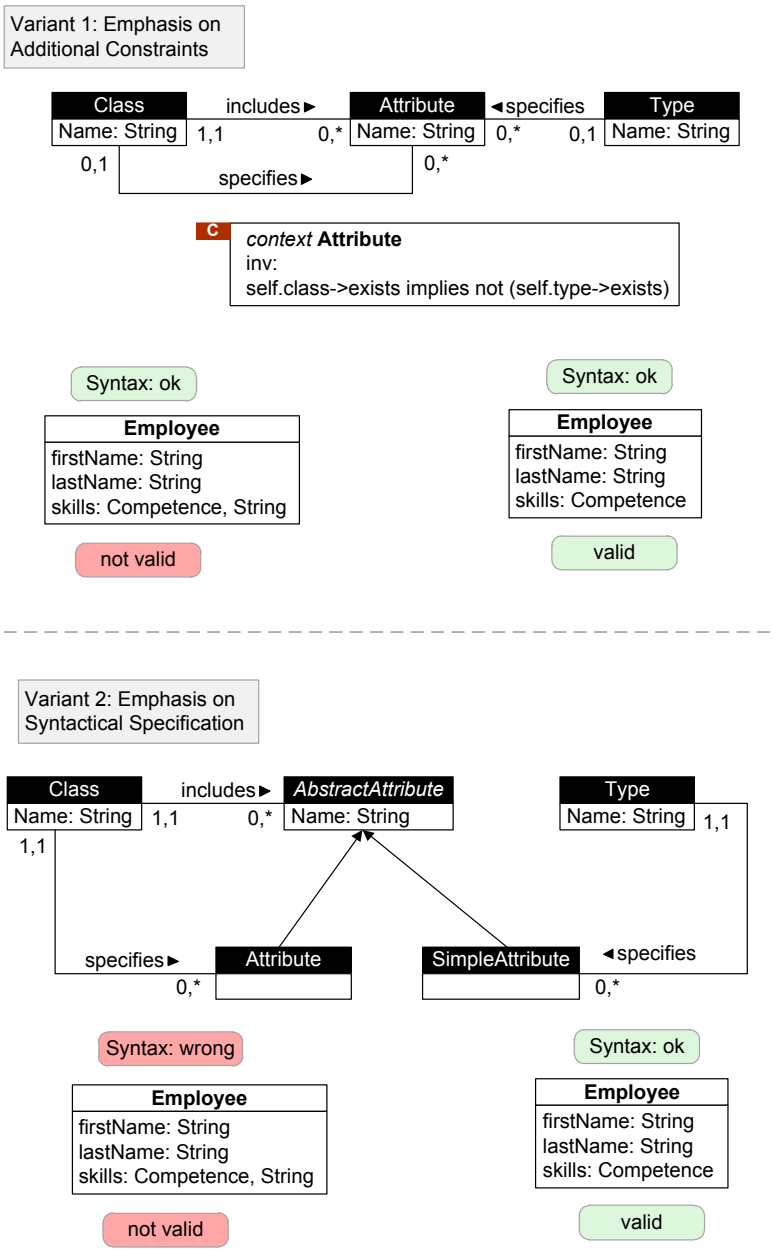

Figure 16: Illustration of different Specification Styles

After the specification of a meta model, an intensive and thorough evaluation is mandatory. This is mainly for two reasons: First, a meta model usually includes specific pitfalls related to subtle differences in abstraction levels, neglected modelling scenarios or inappropriate constraints. Second, a language specification should be as stable as possible, because later changes will usually result in costly adaptations of models and especially of related tools. Unfortunately, the evaluation of a meta model faces particular challenges. Even more than other conceptual models, a meta model cannot be tested directly against targeted domain languages. This is for various reasons. In general, the concepts specified through a meta model are linguistic constructions that include a prescriptive element in the sense that they propose how to structure the targeted domain. Therefore, the concepts may intentionally differ from existing technical concepts, because they are supposed to be superior with re-

spect to certain purposes. On the other hand, with respect to comprehensibility and usability of language concepts, it will often be a design goal that they correspond to terms prospective users are familiar with. Conflicting goals like this are a further reason why a simple comparison against an existing terminology will often not be sufficient. It is certainly a good idea to involve prospective users. However, it may be asked too much of prospective users to judge a meta model directly. Instead, the concepts could be illustrated by showing how to use them for exemplary modelling purposes. In addition to that, a discursive evaluation is recommended. It requires knowledgeable participants and should include a systematic review of the meta model with special emphasis on critical design decisions. The revised meta model needs to be documented comprehensibly.

Input: examples of meta models, previously developed conceptual models of the targeted domain, guidelines for designing meta models

Participants: Language Designer, Domain Expert, User, optional: Tool Expert

Risks: The specification of a DSML will usually face remarkable challenges. Among other things, they relate to conflicting requirements and to the differentiation of language and language application (see above). Language designers with too little experience may overlook these challenges and produce meta models that are not satisfactory. On the other hand, language designers who are aware of the specific problems may struggle for a long time without developing convincing solutions.

Results: meta model, preferable specified with a meta modelling tool that allows for further transformation; extensive documentation

6.2.8 Design of Graphical Notation

It seems reasonable to assume that the graphical notation is of considerable relevance for the comprehensibility, usability and productivity of a DSML. This assumption is backed by a various studies (for an overview see Moody, 2009, p. 758). Therefore, it will usually be no good idea to regard the graphical notation as a meaningless (in the sense of formal semantics) and, hence, marginal feature of a modelling language. Instead, it is recommended to design it with special care and consideration.

Objectives: Design and document graphical notation.

Micro Process:

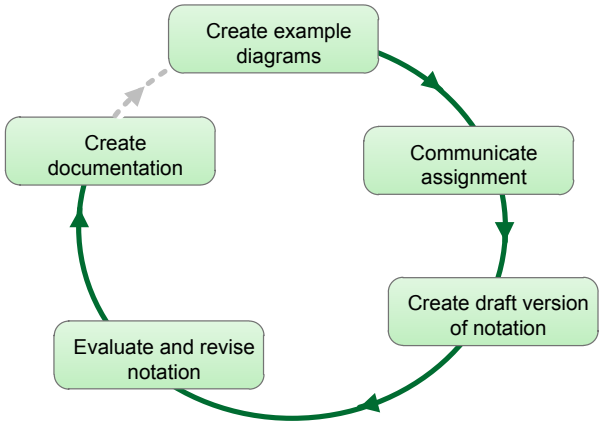

Figure 17: Micro Process 'Design of Graphical Notation'

The creation of a graphical notation is special challenge for at least two reasons. First, there is not much solid ground – in the sense of a theoretical foundation – to build on. Moody proposes probably the most ambitious approach to address this problem (Moody, 2009). However, as we shall see it is still not sufficient to clearly guide the design of a graphical notation. Second, those who are experts in the specification of a modelling language focus on semantics and abstract syntax. Usually, they are no experts in the design of a graphical notation. Sometimes, they will not even be interested in this aspect of language design. Third, those who are trained in the design of iconographic symbols will often lack knowledge about the use of DSML. Even though the existing theoretical background is not satisfactory yet, there are a few guidelines that are backed by theoretical considerations. Therefore, we suggest accounting for existing guidelines – but not without carefully analysing whether and how to apply them appropriately. The following guidelines reflect our own experience and also build on suggestions in the respective literature.

Guideline 1: Build semantic categories of concepts. A category should be characterized by clear features and be intuitively distinguishable from other categories. *Rationale*: Semantics categories are an important prerequisite for designing expressive and discriminating symbols. Example: In process modelling one key category could comprise processes and further category could comprise events.

Guideline 2: Create generic symbols for each category. On the one hand, the concepts covered by a category should be represented by variations of the respective generic symbol. On the other hand, generic symbols of different categories should be distinguishable at first sight. *Rationale*: This guideline should foster the appropriate perception and interpretation, hence, the comprehensibility of a graphical notation. It is further refined in guideline 3.

Guideline 3: The bigger the semantic difference between two concepts, the bigger the graphical difference of the corresponding symbols should be. Moody speaks of "visual distance" (p. 763). This principle applies both to the semantic differences between categories and to the semantic differences between concepts of a particular category. While

there is lack of a convincing precise definition of visual distance, notational differences can be created through shape, colour or text (see guideline 5). *Rationale*: Using a modelling language effectively requires discriminating quickly between concepts. The more different two concepts look, the faster discrimination should be possible. At the same time, a low visual distance should help with identifying and appropriately using similar concepts.

Guideline 4: Prefer icons over shapes. This principle supplements the previous one. While two geometric shapes can be clearly different, e.g. a circle and a rectangle, they lack a reference to the represented concept. An icon is characterized by creating such a reference. This can be realized by an iconic representation of an object, e.g. a letter or a computer, or by the representation of a certain characteristic feature, e.g. symbol depicting a lightning for representing an exception. *Rationale*: Including a perceptual reference into a symbol will contribute to a more intuitive understanding of a notation and, hence, improve its usability and its suitability as a communication medium (see also requirement U1).

Guideline 5: Combine shape (including icons), colour and text effectively. There are studies in Cognitive Psychology which suggest that shape is a more effective instrument to accomplish visual discrimination (Moody, 2009, p. 763). Nevertheless, Moody's resulting recommendation is not entirely convincing: "For this reason, shape should be used as the primary visual variable for distinguishing between different semantic constructs." (ibid) Instead, a more differentiated approach seems to be better suited. In general, semantic categories should be represented by shapes (or icons). This will usually involve the use of colours. The same principle applies to concepts of a certain category, too. However, there are exceptions to this rule. On the one hand, colour can be used as an additional discriminator. This makes sense, if a concept is very similar to others so that it would be difficult and/or misleading to define a separate symbol for representing it. In addition to that, colour can be used as an instrument for users to express additional meaning that is not part of the DSML. Text can be used similarly to colour. It is of particular importance, if the variety of concept occurrences is too large to be covered by symbols. For example: Expressing multiplicities can be accomplished through a set of symbols as it is done in various flavours of the ERM. Apart from the question how comprehensible these symbols are for people who do not use them on a regular base, they are restricted to a given set of multiplicities. If the concept of multiplicities should allow for defining any specific upper and lower bound (as long as the upper bound is greater zero and greater or equal to the lower bound), such an approach simply does not work anymore. Text as an instantiation of a modelling concept is inevitable whenever it is required to name concepts of a model. *Rationale*: Each representation type has specific advantages that need to be evaluated against the requirements a DSML is supposed to satisfy. Therefore, combining representation types appropriately allows for improving a DSML's usability. Figure 18 shows an example of symbols that express starting and terminating events where colour and text are added to a shape in order to foster intuitive understanding.

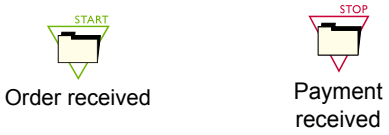

Figure 18: Example for Combining Shape, Colour and Text

Guideline 6: Avoid "symbol overload" (Moody, 2009, p. 763). A symbol should be clearly assigned to one particular modelling concept. *Rationale*: Overloading symbols would contribute to ambiguity and confusion.

Guideline 7: Avoid redundant symbols (Moody, 2009, p. 762). *Rationale*: If a concept can be expressed by alternative symbols, both modellers and model observers are stressed with additional cognitive effort without additional benefit.

Guideline 8: Represent monotonic semantic features of a concept through compositions of symbols. Sometimes, concepts of a DSML need to be further refined to allow for expressing a more specific meaning. If this is accomplished by adding features in a monotonic fashion, each of these features can be represented by a respective symbol. A particular occurrence of the concept will then be represented by a composition of related symbols. *Rationale*: Combining graphical elements contributes to a more systematic construction of a graphical notation that is in line with the semantics of the related concepts. It also helps avoiding overwhelming users with huge palettes of prefabricated symbols. With respect to building tools, it implies a larger effort for implementing the composition of more complex graphical symbols. At the same time, it improves flexibility. The example in Figure 19 illustrates the composition of various event types from symbols that express certain semantic features.

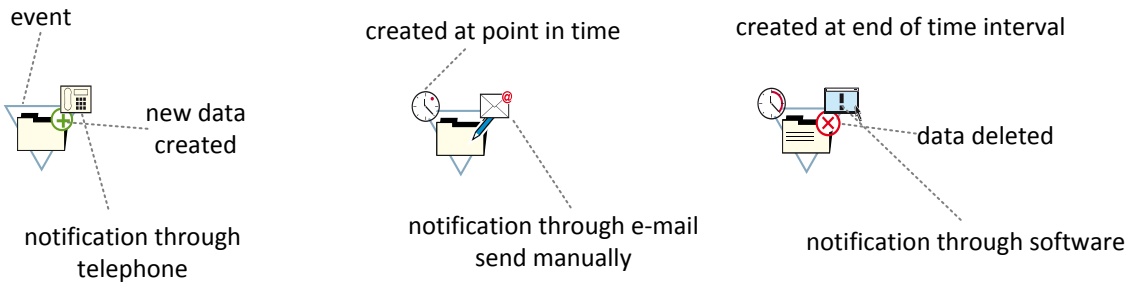

Figure 19: Example of Combining Notation Elements for Representing differentiated Concepts

In addition to principles which guide the design of graphical symbols that represent certain modelling concepts, further notational elements may be required to cope with the complexity created by large models.

Guideline 9: A graphical notation should include symbols that allow for reducing diagram complexity. On the one hand, these are symbols that represent compositions of a set of other symbols. If required, they serve as a starting point for decompositions. A typical example would be a symbol used to represent aggregate processes. On the other hand, special symbols can be introduced that allow for depicting a set of identical diagram parts by one common representation. The example in Figure 20 illustrates this kind of simplification: In a business process model, a branching decision results in two

alternative paths of execution. If the alternative paths at some point continue in identical ways, these identical parts of both paths could be merged into one common representation. *Rationale*: Lowering the visual complexity of a diagram can substantially contribute to a better understanding and, hence, to increased productivity.

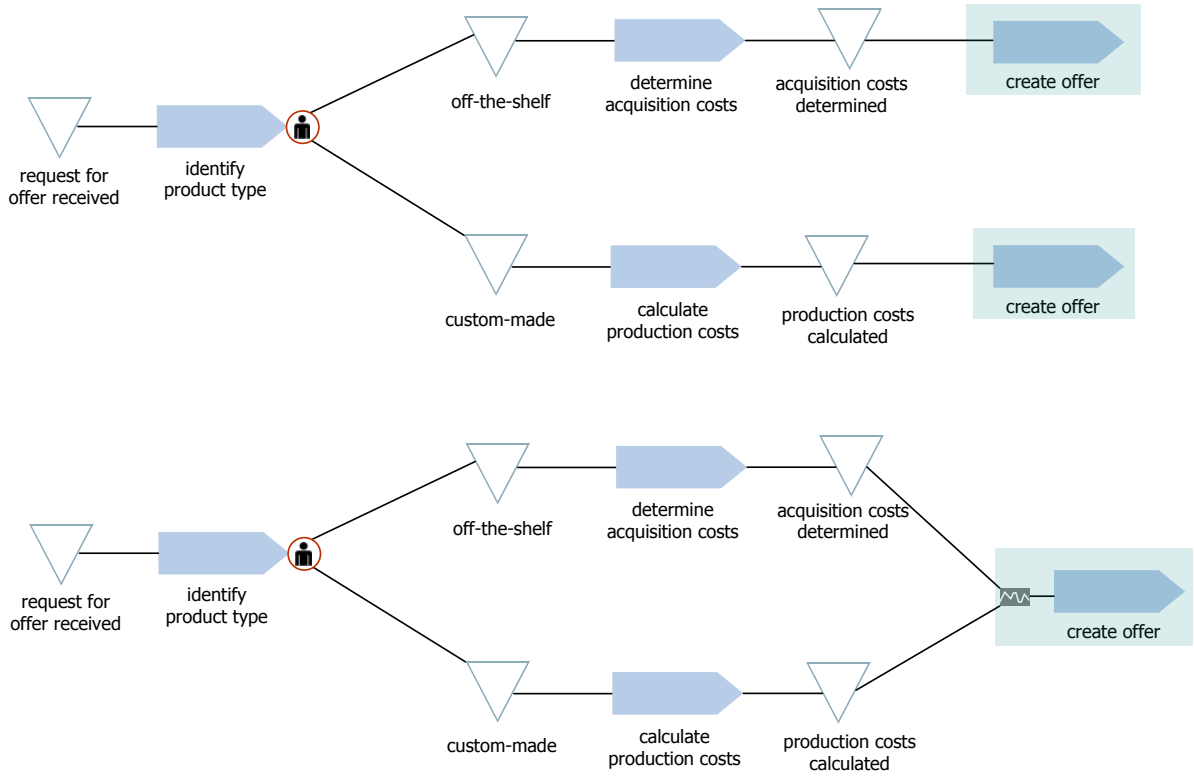

Figure 20: Example of merging two identical Paths of Execution

In order to apply the suggested guidelines appropriately, it is mandatory to develop a clear idea of the prospective language users and the modes of use to be covered. This recommends answering the following questions:

- Are prospective users already familiar with graphical modelling languages?
- Are they supposed to use the DSML on a regular base?
- What are typical use scenarios?
- What are the concepts of the DSML users are most interested in?
- What are aesthetic preferences of the DSML users?
- Is the group of prospective users homogeneous or rather heterogeneous with respect to the above questions?

Depending on the answers to these questions, further refinements may be required. If the prospective users do not only comprise experts but also novices that will use the DSML only rarely, it may be an option to provide a simplified, "light" version of the notation that covers only those concepts that are sufficient for inexperienced users. Note, however, that this may require substantial effort with respect to modifying/extending the syntax specification. If aesthetic preferences are expected to vary, one could provide different flavours of a notation,

e.g. one that features artistic icons and a further one that emphasizes a more business-like style.

The design of an elaborate graphical notation recommends involving a professional graphic designer, which implies to somehow specify the corresponding assignment. Experience gained in previous projects indicates that example diagrams featuring a preliminary notation serve as a useful illustration for communicating concepts and modelling purposes to a graphic designer. Subsequently, the graphic designer is supposed to develop a draft version of the notation, which will then be – if required – repeatedly evaluated and revised until a satisfactory state is accomplished.

Finally, a documentation of the graphical notation is created. It consists at least of a comprehensive listing of all symbols and respective descriptions. Furthermore, it helps to illustrate the notation with a few example diagrams.

Input: (Revised) exemplary diagrams from requirements analysis.

Participants: Domain Expert, User, Language Designer, Tool Expert, Graphic Designer

Risks: Often, most of the participants will not have a specific expertise in designing and judging a graphical notation. While graphic designers should be familiar at least with the design of iconographic symbols, they may lack a clear understanding of the very purpose a DSML should serve. As a consequence, there is a clear risk that the graphical notation remains dissatisfactory. To reduce this risk, special attention should be applied to the selection of the graphic designer and to the execution of test procedures.

Results: Documentation of graphical notation, illustrated through exemplary diagrams

6.2.9 Development of Modelling Tool

Often, the effective and efficient use of a DSML will require a corresponding modelling tool. Tool development is not an inherent part of the proposed method to developing a DSML. Also, the development of a modelling tool can require a major software development project. Nevertheless, a brief consideration of developing a tool is included here. It serves to point at key aspects relevant for developing a tool that may – in part – be accounted for already with the language specification.

Objectives: realization of a modelling tool for the previously developed DSML

Micro Process:

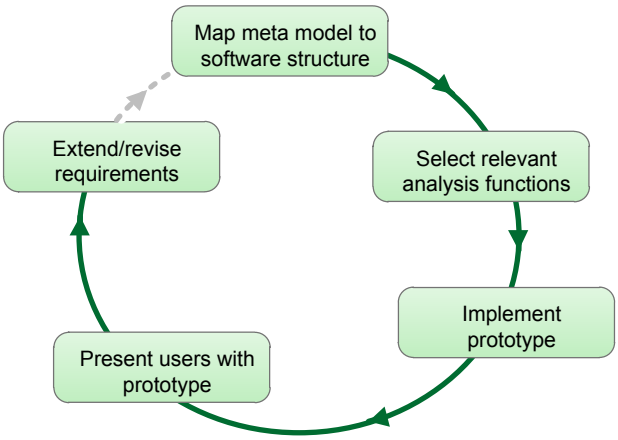

Figure 21: Micro Process 'Development of Modelling Tool'

A meta model provides a good foundation for developing a modelling tool: Meta types can be represented by classes. However, it is not sufficient to simply map each meta type to a corresponding class. On the one hand, the specific requirements related to the modelling tool may recommend deviating from the structure of the meta model. For example: If it can be expected that the meta type of model element is frequently changed during the lifetime of a model, it may be a good choice to represent the respective meta types in one class and differentiate the "type" through states of corresponding objects (see section 6.2.7). On the other hand, further aspects that are not part of a meta model need to be accounted for. At first, there are features that will usually be considered as mandatory, such as adding time stamps to model elements and to provide for access control. In addition to that, persistency will always be an issue. In the simplest case, persistency could be achieved by storing a diagram together with the corresponding model as a file. More demanding solutions would separate a diagram from the respective model, allowing for storing various diagrams of one model. In this case, it will usually be more appropriate to use a database in order to not jeopardize integrity. The user-interface should offer effective patterns of interaction and – if required – account for different user profiles. In order to design a tool that fits its prospective users' needs, it is necessary to perform a requirements analysis that supplements the analysis of requirements related to the DSML. A good starting point for analysing tool requirements are the exemplary diagrams used already before. For each diagram, the required tool functions as well as corresponding user interface requirements have to be analysed. Often, prospective users will not be familiar with DSML tools. Therefore, it is recommended to implement a prototype at an early stage to promote feedback by users. Based on that, requirements may be revised or further requirements may evolve. These feedback circles are performed until the tool has reached a satisfactory state. Since a DSML editor will usually be a complex software system, a thorough design of its architecture is mandatory.

Often, developing a tool from scratch will be no option, since it causes too much effort. Therefore, it will usually be crucial to deploy a meta modelling tool. It should allow for specifying the meta model of the targeted DSML with an appropriate meta meta modelling language. Subsequently, the meta modelling tool should create a first version of the intended DSML editor. Usually, this will be accomplished by generating corresponding code from the meta model. For a more detailed description of generating modelling tools see Frank (2011a), Steinberg et al. (2008), Gulden and Frank (2010), for an approach that avoids the obstacles of code generation see Bettin and Clark (2010). Sometimes, a DSML tool may not just be used during build-time, but also during run-time, which implies additional requirements and challenges (see Frank and Strecker, 2009).

Input: exemplary diagrams from requirements analysis, meta model; optional, but recommended: meta modelling environment

Participants: Domain Expert, User, Language Designer, Tool Expert

Risks: Developing a DSML editor requires experienced and highly qualified software developers. This is not only the case for developing the tool from scratch, but will usually be the case for deploying additional tools or frameworks. At the same time, there is only little specific experience with developing DSML editors. Against this background, two specific risks have to be accounted for. First, if the available software developers are not sufficiently qualified, the entire project is likely to fail. This risk implies carefully checking on the developers abilities in advance, which may result in looking for specialized software companies or freelancers. Second, cost estimates will often be hard to create and judge. This risk needs to be explicitly accounted for. It can be addressed by starting with a small prototype to get a better idea of the required effort. In any case, it should be made sure that a project's success does not depend on tight deadlines.

Results: DSML tool, optionally as prototype

6.2.10 Evaluation and Revision

In order to ensure a certain quality level, it is mandatory to finally evaluate and possibly revise the tool. The evaluation needs to account for specific challenges.

Objectives: creation of systematic and comprehensible evaluation; if needed, revision of tool

Micro Process:

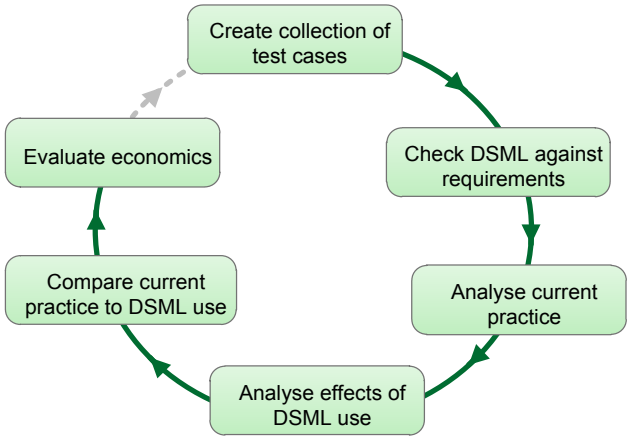

Figure 22: Micro Process 'Evaluation and Revision'

First, the evaluation of a DSML and a corresponding modelling tool recommends checking them against the requirements – both generic (see section 3) and specific. For some requirements, it will be fairly easy to decide whether or not they are fulfilled. For example: "There should be concepts that allow for assigning probabilities to alternative paths of executing a business process." In these cases, it will be sufficient to determine and document whether or not a respective requirement is satisfied. With other requirements, an assessment may be harder. For example: "The modelling language should provide domain-specific concepts as long as they are regularly used and their semantics is invariant within the scope of the language's application." or: "The concepts of the language should be easy to comprehend by different groups of users." There are various approaches to reduce the respective uncertainty. First, it will usually be helpful to check language features against the background of a use case. Therefore, evaluation recommends building on the use scenarios created for requirements analysis. Each use case serves to analyse whether and how corresponding requirements are satisfied by the DSML. For this purpose, a discursive evaluation is of pivotal importance. It should include language designers, domain experts and prospective users. Users as the primary addressees of the language play a key role. However, that does not mean to simply rely on users' assessments. Instead, it is important to account for their respective background, such as experience with similar or other modelling languages, their formal education, the time they had to learn the language and/or the tool, their attitude towards DSML in general etc. Similar considerations apply to other participants, too. However, language designers can be expected to have a different background which should enable them to provide more elaborate assessments – which, however, may also cause subtle bias in their judgment. In general, the evaluation of linguistic artefacts faces a substantial challenge: In order to assess a language, we need to speak it (to some degree). In addition to that we need to know some kind of meta language that enables us to speak about the language to assess. Hence, the language we know is a necessary instrument, but it also limits our reasoning powers because we cannot entirely go beyond our language (see section **Fehler! Ver-**

weisquelle konnte nicht gefunden werden. and section 3). Hence, it may well be that a different language that lies beyond our imagination would provide for a more effective and efficient structuring of the problem. At the same time the problem itself might appear different, if it is described/constructed in a different language. While these considerations seem to be of mainly philosophical nature, they are nevertheless important for the reflective evaluation of a DSML. On the one hand, they recommend being sensitive to the effect of existing languages skills. On the other hand, they emphasize the contingency of the subject: Language skills as well as language use are subject of change. Therefore a particular DSML is a moving target. As a consequence, its assessment should not be restricted to a certain point in time, but rather be organized as a permanent process.

Evaluating a DSML against requirements does not necessarily allow for assessing its contribution to achieving pivotal goals such as improving productivity or quality of problem solving. To analyse this aspect, the current practice can be investigated for selected analysis scenarios. For each of the selected scenarios, the effects of using the DSML can then be analysed in detail. Based on these analyses current practice can be compared against the use of the DSML – with respect to aspects such as learning effort, productivity, comprehensibility, quality of solution and documentation etc. Since an economic evaluation will often be important to justify investments into future DMSL projects, it may be required to perform a comparative economic assessment of utility and costs. Note, however, that this assessment addresses a moving target, since costs and utility may change with the availability of further tools and with users' qualification.

Input: documentation of DSML, tool implementation, documentation, exemplary diagrams, use scenarios

Participants: Domain Expert, User, Language Designer, Tool Expert

Risks: The evaluation of languages may be jeopardized by various sources of sometimes subtle bias. In addition to that, there may be political interests involved that contribute to opportunistic assessments. Therefore, each assessment should be supplemented by a justification. If it is not possible to give a convincing justification, the assumptions an assessment is based on should be made explicit.

Results: evaluation report, revised and approved tool

7 Evaluation and Conclusions

DSML seem as a logical evolution of GPML. Their emergence corresponds to the evolution of elaborate technical languages, which are a key instrument of promoting industrial and post-industrial productivity. There would be no advanced craft, no engineering and no medicine without specific technical languages – just to name only a few fields. Against this background, it seems astonishing that complex systems are still analysed and designed with GPML that are restricted to a few generic concepts such as 'class', 'attribute', 'state' etc. DSML represent a clearly more productive instrument for describing and analysing problems as well as for designing systems. However, the specification of a DSML is a remarkable challenge. This is for various reasons. First, we are entering terra incognita to a certain degree, because so far there is only little experience with designing and using DSML. This results in the problem that it will often be difficult to ask the right questions. Second, the design has to account for competing or conflicting goals. One specific challenge is the quest for sustainability – even more than a model, a language should be stable for a longer time – which is contrasted by the contingent evolution of many domains. Third, the specification of a DSML requires a meta modelling language. However, it is not trivial to assess the benefit of an instrument one has little experience with for a complex task one is not too familiar with. The method presented in this report is aimed at providing a framework and guidelines to reduce complexity and risk related to the development of DSML. More specifically, it should satisfy the requirements outlined in chapter 5. Table 5 presents an evaluation of the method against these requirements.

Requirement	Evaluation
P1: The range of possible DSML should be outlined	The method is aimed at DSML for enterprise modelling. Hence, prospective users should get a relatively clear idea of the subjects the method can be applied to. Nevertheless, it is still possible that further, not yet foreseen DSML in the realm of enterprise modelling will create additional requirements for a method. Also, it is likely that the method can be applied to areas other than enterprise modelling, too.
P2: Peculiarities and challenges related to DSML design should be accounted for in detail.	Although it would be inappropriate to claim that the method includes a comprehensive list of peculiarities and challenges, it provides advanced users with extensive discussions of frequent problems (see e.g. 6.2.7). In addition to that the method includes references to more specific publications.
P3: Conflicting design goals should be made explicit.	Generic patterns of conflicting goals are accounted for, e.g. the conflict between a higher and a lower level of semantics or the benefits and shortcomings of syntactic-centred versus semantic-centred specification.
P5: Prospective users should be effectively supported in understanding the DSML.	Exemplary diagrams and corresponding use scenarios are suggested as a pivotal instrument to develop a clearer picture of an artefact that we – and especially prospective users – may not sufficiently understand at first.
P6: The method should pro-	Generic requirements of DSML are analysed in 5.

vide a description of generic requirements.	
P7: The method should account for the need to determine the economics of developing and using a DSML and corresponding tools.	The method emphasizes the relevance of an economic justification. It points at key impact factors such as semantics and range of reuse. Furthermore, the micro process 'Evaluation and Revision' includes guidelines for analysing the economics of a DSML and a corresponding modelling tool.
P8: The method should provide support for assessing and selecting a meta modelling language and corresponding development tools	In this report, requirements to be accounted for with the selection of a meta modelling language are only briefly and indirectly accounted for in 6.1. The method includes, however, a reference to a comprehensive requirements analysis in Frank (2011a). Selecting meta modelling tools is addressed only briefly in this report. There are, however, references to more specific publications included.
P9: The method should account for protection of investment and therefore emphasize a longer term perspective.	This aspect is emphasized repeatedly to foster awareness. Developing and analysing use scenarios should always be accompanied by the question, whether they will still be relevant in future times. However, the stability of a DSML depends chiefly on the quality of abstractions and on the contingency of the subject – which go both beyond the contributions of the method.
P10: The method should provide guidelines for designing graphical notations.	In addition to a set of guidelines (6.2.8), the method recommends involving a graphic designer and suggests how to communicate a respective assignment.
P11: The tasks that need to be performed during the development of a DSML should be assigned to roles.	The method includes a role model. The respective roles are used in the description of the micro processes.

Table 5: Overview of evaluating the method against requirements

The method has gradually evolved from our experience with designing DSML. In its current state, it represents a major improvement compared to the times when we did not have a dedicated method. That does not mean, however, that we would regard it as mature. Instead, we rather see it as a repository of knowledge about developing DSML that should grow with the number of respective projects. In other words: Prospective users of the method should not take all guidelines for granted, but reflect upon them and – if required – adapt them. This is especially the case for researchers who pursue the design of a DSML as a scientific task: Any research that either aims at analysing a language and its use or at inventing new "language games" (i.e. artificial languages and actions built upon them), has to face a subtle challenge that is caused by the demand to justify prospective contributions to scientific knowledge: Every researcher is trapped in a network of language, patterns of thought and action he cannot completely transcend – leading to a paradox that can hardly be resolved: Designing a language is not possible without using a language. At the same time, any language we use for this purpose will bias our perception and judgment.

8 References

Bettin, J. & Clark, T. 2010. Advanced Modelling Made Simple with the Gmodel Metalanguage. *Proceedings of Model-Driven Interoperability 2010.* Springer.

Bunge, M. 1977. *Treatise on Basic Philosophy. Vol. 3: Ontology I: The Furniture of the World,* Dordrecht, Reidel.

Bunge, M. 1979. *Treatise on Basic Philosophy: Volume 4: Ontology II,* Dordrecht, Reidel.

Ducasse, S. & Gîrba, T. 2006. Using Smalltalk as a Reflective Executable Meta-language. *In:* NIERSTRASZ, O. & AL., E. (eds.) *Proceedings of Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006.* Springer.

Fettke, P. & Loos, P. 2003. Ontological evaluation of reference models using the Bunge-Wand-Weber-model. *Proceedings of the Ninth Americas Conference on Information Systems.* Tampa, FL.

Frank, U. 1998. Evaluating Modelling Languages: Relevant Issues, Epistemological Challenges and a Preliminary Research Framework. *Arbeitsberichte des Instituts für Wirtschaftsinformatik.* Koblenz: University of Koblenz.

Frank, U. 2006a. Evaluation of Reference Models. *In:* FETTKE, P. & LOOS, P. (eds.) *Reference Modelling for Business Systems Analysis.* Idea Group.

Frank, U. 2006b. Towards a Pluralistic Conception of Research Methods in Information Systems Research. *ICB Research Reports.* Universität Duisburg-Essen.

Frank, U. 2011a. The MEMO Meta Modelling Language (MML) and Language Architecture. 2nd Edition. *ICB Research Report.* Essen.

Frank, U. 2011b. Multi-Perspective Enterprise Modelling: Background and Terminological Foundation. *ICB Research Report* University Duisburg-Essen.

Frank, U. & Laak, B. v. 2003. Anforderungen an Sprachen zur Modellierung von Geschäftsprozessen. *Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz.* Koblenz.

Frank, U. & Strecker, S. 2007. Open Reference Models – Community-driven Collaboration to Promote Development and Dissemination of Reference Models. *Enterprise Modelling and Information Systems Architectures,* 2**,** 32-41.

Frank, U. & Strecker, S. 2009. Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems. *ICB-Research Report* University Duisburg-Essen.

Goldstein, R. C. & Storey, V. 1990. Some findings on the intuitiveness of entity-relationship constructs. *In:* LOCHOWSKY, F. H. (ed.) *Entity-Relationship Approach to Database Design and Querying.* Amsterdam: Elsevier Science.

Grossmann, R. 1983. *The Categorial Structure of the World,* Bloomington, Indiana University Press.

Gulden, J. & Frank, U. 2010. MEMOCenterNG. A full-featured modeling environment for organisation modeling and model-driven software development. *22nd International Conference on Advanced Information Systems Engineering (CAiSE '10).* Hammamet.

Henderson-Sellers, B. & Ralyté, J. 2010. Situational Method Engineering: State-of-the-Art Review. *Journal of Universal Computer Science,* 16**,** 424-478.

Kelly, S. & Tolvanen, J.-P. 2008. *Domain-Specific Modelling. Enabling Full Code Generation,* Hoboken, N.J., John Wiley & Sons.

Moody, D. L. 2009. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering,* 35**,** 756-779.

OMG 2010. Object Constraint Language. Version 2.2.

Opdahl, A. L. & Henderson-Sellers, B. 1999. Evaluating and Improving Object-Oriented Modelling Languages Using the BWW-Model. *Information Systems Foundations Workshop (Ontology, Semiotics and Practice).* Sidney.

Steinberg, D., Budinsky, F., Paternosto, M. & Merks, E. 2008. *EMF: Eclipse Modeling Framework*, Addison-Wesley Professional.

Strecker, S., Frank, U., Heise, D. & Kattenstroth, H. 2011. MetricM: A modeling method in support of the reflective design and use of performance measurement systems (forthcoming). *Information Systems and e-Business Management*.

Süttenbach, R. & Ebert, J. 1997. A Booch Metamodel. *Fachberichte Informatik.* Universität Koblenz.

Wand, Y. & Weber, R. 1995. On the deep structure of information systems. *Information Systems Journal,* 5**,** 203-23.

Previously published ICB - Research Reports

2010

No 41 (December)

Adelsberger,Heimo; Drechsler, Andreas (Eds): "Ausgewählte Aspekte des Cloud-Computing aus einer IT-Management-Perspektive – Cloud Governance, Cloud Security und Einsatz von Cloud Computing in jungen Unternehmen"

No 40 (October 2010)

Bürsner, Simone; Dörr, Jörg; Gehlert, Andreas; Herrmann, Andrea; Herzwurm, Georg; Janzen, Dirk; Merten, Thorsten; Pietsch, Wolfram; Schmid, Klaus; Schneider, Kurt; Thurimella, Anil Kumar (Eds): "16th International Working Conference on Requirements Engineering: Foundation for Software Quality. Proceedings of the Workshops CreaRE, PLREQ, RePriCo and RESC"

No 39 (May 2010)

Strecker, Stefan; Heise, David; Frank, Ulrich: "Entwurf einer Mentoring-Konzeption für den Studiengang M.Sc. Wirtschaftsinformatik an der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen"

No 38 (February 2010)

Schauer, Carola: "Wie praxisorientiert ist die Wirtschaftsinformatik? Einschätzungen von CIOs und WI-Professoren"

No 37 (January 2010)

Benavides, David; Batory, Don; Grunbacher, Paul (Eds.): "Fourth International Workshop on Variability Modelling of Software-intensive Systems"

2009

No 36 (December 2009)

Strecker, Stefan: "Ein Kommentar zur Diskussion um Begriff und Verständnis der IT-Governance - Anregungen zu einer kritischen Reflexion"

No 35 (August 2009)

Rüngeler, Irene; Tüxen, Michael; Rathgeb, Erwin P.:"Considerations on Handling Link Errors in STCP"

No 34 (June 2009)

Karastoyanova, Dimka; Kazhamiakan, Raman; Metzger, Andreas; Pistore, Marco (Eds.): "Workshop on Service Monitoring, Adaption and Beyond"

No 33 (May 2009)

Adelsberger,Heimo; Drechsler , Andreas; Bruckmann, Tobias; Kalvelage, Peter; Kinne, Sophia; Pellinger, Jan; Rosenberger, Marcel; Trepper, Tobias: „Einsatz von Social Software in Unternehmen – Studie über Umfang und Zweck der Nutzung"

No 32 (April 2009)

Barth, Manfred; Gadatsch, Andreas; Kütz, Martin; Rüding, Otto; Schauer, Hanno; Strecker, Stefan: „Leitbild IT-Controller/-in – Beitrag der Fachgruppe IT-Controlling der Gesellschaft für Informatik e. V."

No 31 (April 2009)
Frank, Ulrich; Strecker, Stefan: "Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems – Requirements, Conceptual Foundation and Design Options"

No 30 (February 2009)
Schauer, Hanno; Wolff, Frank: „Kriterien guter Wissensarbeit – Ein Vorschlag aus dem Blickwinkel der Wissenschaftstheorie (Langfassung)"

No 29 (January 2009)
Benavides, David; Metzger, Andreas; Eisenecker, Ulrich (Eds.): "Third International Workshop on Variability Modelling of Software-intensive Systems"

2008

No 28 (December 2008)
Goedicke, Michael; Striewe, Michael; Balz, Moritz: „Computer Aided Assessments and Programming Exercises with JACK"

No 27 (December 2008)
Schauer, Carola: "Größe und Ausrichtung der Disziplin Wirtschaftsinformatik an Universitäten im deutschsprachigen Raum - Aktueller Status und Entwicklung seit 1992"

No 26 (September 2008)
Milen, Tilev; Bruno Müller-Clostermann: " CapSys: A Tool for Macroscopic Capacity Planning"

No 25 (August 2008)
Eicker, Stefan; Spies, Thorsten; Tschersich, Markus: "Einsatz von Multi-Touch beim Softwaredesign am Beispiel der CRC Card-Methode"

No 24 (August 2008)
Frank, Ulrich: "The MEMO Meta Modelling Language (MML) and Language Architecture – Revised Version"

No 23 (January 2008)
Sprenger, Jonas; Jung, Jürgen: "Enterprise Modelling in the Context of Manufacturing – Outline of an Approach Supporting Production Planning"

No 22 (January 2008)
Heymans, Patrick; Kang, Kyo-Chul; Metzger, Andreas, Pohl, Klaus (Eds.): "Second International Workshop on Variability Modelling of Software-intensive Systems"

2007

No 21 (September 2007)
Eicker, Stefan; Annett Nagel; Peter M. Schuler: "Flexibilität im Geschäftsprozess-management-Kreislauf"

No 20 (August 2007)
Blau, Holger; Eicker, Stefan; Spies, Thorsten: "Reifegradüberwachung von Software"

No 19 (June 2007)
Schauer, Carola: "Relevance and Success of IS Teaching and Research: An Analysis of the ‚Relevance Debate'

No 18 (May 2007)
Schauer, Carola: "Rekonstruktion der historischen Entwicklung der Wirtschaftsinformatik: Schritte der Institutionalisierung, Diskussion zum Status, Rahmenempfehlungen für die Lehre"

No 17 (May 2007)
Schauer, Carola; Schmeing, Tobias: "Development of IS Teaching in North-America: An Analysis of Model Curricula"

No 16 (May 2007)
Müller-Clostermann, Bruno; Tilev, Milen: "Using G/G/m-Models for Multi-Server and Mainframe Capacity Planning"

No 15 (April 2007)
Heise, David; Schauer, Carola; Strecker, Stefan: "Informationsquellen für IT-Professionals – Analyse und Bewertung der Fachpresse aus Sicht der Wirtschaftsinformatik"

No 14 (March 2007)
Eicker, Stefan; Hegmanns, Christian; Malich, Stefan: "Auswahl von Bewertungsmethoden für Softwarearchitekturen"

No 13 (February 2007)
Eicker, Stefan; Spies, Thorsten; Kahl, Christian: "Softwarevisualisierung im Kontext serviceorientierter Architekturen"

No 12 (February 2007)
Brenner, Freimut: "Cumulative Measures of Absorbing Joint Markov Chains and an Application to Markovian Process Algebras"

No 11 (February 2007)
Kirchner, Lutz: "Entwurf einer Modellierungssprache zur Unterstützung der Aufgaben des IT-Managements – Grundlagen, Anforderungen und Metamodell"

No 10 (February 2007)
Schauer, Carola; Strecker, Stefan: "Vergleichende Literaturstudie aktueller einführender Lehrbücher der Wirtschaftsinformatik: Bezugsrahmen und Auswertung"

No 9 (February 2007)
Strecker, Stefan; Kuckertz, Andreas; Pawlowski, Jan M.: "Überlegungen zur Qualifizierung des wissenschaftlichen Nachwuchses: Ein Diskussionsbeitrag zur (kumulativen) Habilitation"

No 8 (February 2007)
Frank, Ulrich; Strecker, Stefan; Koch, Stefan: "Open Model - Ein Vorschlag für ein Forschungsprogramm der Wirtschaftsinformatik (Langfassung)"

2006

No 7 (December 2006)
Frank, Ulrich: "Towards a Pluralistic Conception of Research Methods in Information Systems Research"

No 6 (April 2006)
Frank, Ulrich: "Evaluation von Forschung und Lehre an Universitäten – Ein Diskussionsbeitrag"

No 5 (April 2006)
Jung, Jürgen: "Supply Chains in the Context of Resource Modelling"

No 4 (February 2006)
Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part III – Results Wirtschaftsinformatik Discipline"

2005

No 3 (December 2005)
Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part II – Results Information Systems Discipline"

No 2 (December 2005)
Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part I – Research Objectives and Method"

No 1 (August 2005)
Lange, Carola: „Ein Bezugsrahmen zur Beschreibung von Forschungsgegenständen und -methoden in Wirtschaftsinformatik und Information Systems"

Research Group	Core Research Topics
Prof. Dr. H. H. Adelsberger Information Systems for Production and Operations Management	E-Learning, Knowledge Management, Skill-Management, Simulation, Artificial Intelligence
Prof. Dr. P. Chamoni MIS and Management Science / Operations Research	Information Systems and Operations Research, Business Intelligence, Data Warehousing
Prof. Dr. F.-D. Dorloff Procurement, Logistics and Information Management	E-Business, E-Procurement, E-Government
Prof. Dr. K. Echtle Dependability of Computing Systems	Dependability of Computing Systems
Prof. Dr. S. Eicker Information Systems and Software Engineering	Process Models, Software-Architectures
Prof. Dr. U. Frank Information Systems and Enterprise Modelling	Enterprise Modelling, Enterprise Application Integration, IT Management, Knowledge Management
Prof. Dr. M. Goedicke Specification of Software Systems	Distributed Systems, Software Components, CSCW
Prof. Dr. V. Gruhn Software Engineering	Design of Software Processes, Software Architecture, Usability, Mobile Applications, Component-based and Generative Software Development
Prof. Dr. T. Kollmann E-Business and E-Entrepreneurship	E-Business and Information Management, E-Entrepreneurship/E-Venture, Virtual Marketplaces and Mobile Commerce, Online-Marketing
Prof. Dr. B. Müller-Clostermann Systems Modelling	Performance Evaluation of Computer and Communication Systems, Modelling and Simulation
Prof. Dr. K. Pohl Software Systems Engineering	Requirements Engineering, Software Quality Assurance, Software-Architectures, Evaluation of COTS/Open Source-Components
Prof. Dr.-Ing. E. Rathgeb Computer Networking Technology	Computer Networking Technology
Prof. Dr. E. Rukzio Mobile Mensch Computer Interaktion mit Software Services	Novel Interaction Technologies, Personal Projectors, Pervasive User Interfaces, Ubiquitous Computing
Prof. Dr. A. Schmidt Pervasive Computing	Pervasive Computing, Uniquitous Computing, Automotive User Interfaces, Novel Interaction Technologies, Context-Aware Computing
Prof. Dr. R. Unland Data Management Systems and Knowledge Representation	Data Management, Artificial Intelligence, Software Engineering, Internet Based Teaching
Prof. Dr. S. Zelewski Institute of Production and Industrial Information Management	Industrial Business Processes, Innovation Management, Information Management, Economic Analyses

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)