Some Guidelines for the Conception of Domain-Specific Modelling Languages

Ulrich Frank

Information Systems and Enterprise Modelling
University of Duisburg-Essen
ulrich.frank@uni-due.de

Abstract: While the potential prospects of domain-specific modelling languages (DSML) are undisputed, the design of a DSML faces specific challenges that have raised only little attention so far. They relate to the boundaries between a DSML and corresponding models as well as to the question how specific a DSML should be. Addressing these challenges does not only contribute to the development of meta modelling methods, it also relates to judging the economics of a DSML. This paper suggests guidelines to support typical decisions that occur with the design of a DSML. They mainly concern the level of abstraction a potential language concept should be defined on. The guidelines are not intended to serve as recipes for design, but rather to improve the transparency of design decisions.

1. Motivation

In recent years, the conception of domain-specific modelling languages (DSML) has gained increasing attention. This is for comprehensible reasons: A general purpose modelling language (GPML) provides modellers with rudimentary concepts such as 'Class', 'Attribute' etc. only. Different from that, a DSML includes higher level concepts that are reconstructions of technical terms in the targeted domain, e.g. 'Organisational Unit', 'Business Process', 'Strategic Resource' etc. Therefore, the modeller is not forced to specify these high level terms on his own. Instead he may reuse the concepts of the DSML. This will not only promote modelling productivity. It also contributes to model quality: Different from a GPML, the concepts of a DSML include implicit integrity constraints that prevent the construction of nonsensical models. Furthermore, it should be expected that the concepts of a DSML are developed with specific expertise and care. Also, a DSML will typically feature a specific graphical notation that should foster comprehensibility of respective models. The small, simplified example in Figure 1 illustrates the benefits of DSML compared to GPML. The prospects of deploying DSMLs are contrasted with remarkable challenges. They concern both, the economics of DSMLs and their development as well as principal considerations of language design. 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. Wand's and Weber's approach to evaluate the quality of modelling languages by referring to ontological categories [WaWe93] does not help much with the specification of a DSML in general, nor with differentiating levels of abstraction in particular. Take,

93

for instance, the construct "ontological completeness": Depending on the purpose of a DSML, completeness in this sense may be not relevant.

Work on modelling language design is usually focusing on formal features of meta modelling languages or on meta modelling tools (e.g. [Jeus07], [JaSz06]). Since a modelling method consists of at least one modelling language and a corresponding process model, one would assume that work on method engineering has produced support for the development of modelling languages.

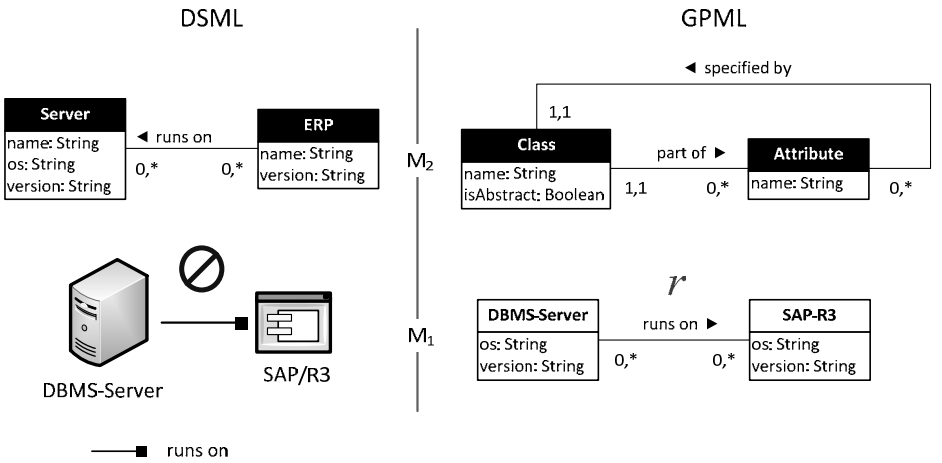

Figure 1: Illustration of DSML in comparison to GPML

However, this is not the case. While there is a plethora of approaches on method engineering (e.g. [Brin96], [Roll09], for an overview see [HeRa10]), they mainly focus on the construction and configuration of process models and take the modelling language as given. Recent publications on DSML such as [Klep08] and [KeTo08] describe the construction and use of DSMLs, but do not go into the details of meta model design.

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 domain-specific modelling languages 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

94

realisation of corresponding model editors, prospective users can expect only little guidance with designing a language that fits its purpose.

This paper is intended to suggest guidelines for supporting design decisions that are specific to the development of DSML. For this purpose, it addresses mainly three aspects: *scope and economics*, *differentiating between language and language application* and *dealing with abstraction conflicts*. The background of this research is work on languages for enterprise modelling. Nevertheless, the results should be applicable to DSML in other areas as well.

2. Scope and Economics of DSML

Usually, the decision for developing a DSML will require some sort of economic justification. The costs for developing – and maybe distributing – a DSML should not exceed the additional benefit to be expected from its deployment. Unfortunately, development costs and benefits are hard to judge in advance. If a DSML is developed for one particular project only (as recommended e.g. by [KeTo08]), it may be possible to estimate the expected cost/benefit ratio in a seemingly satisfactory way. However, as soon as the development costs require a higher scale of reuse, the decision whether a DSML will pay off or not will be of remarkable complexity and contingency. It seems reasonable to assume that – ceteris paribus – the development costs will be the higher, the more elaborate and specific a language is, i.e. the more semantics (in the sense of information content) its concepts incorporate. However, the effect of semantics on a DSML's benefit is ambivalent. On the one hand, semantics should promote modelling productivity and model quality, hence the benefit of reuse in a particular case. On the other hand, semantics will compromise the scale of reuse: The more specific a DSML is, the fewer the number of cases it can be applied to. Note, however, 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. Despite these obstacles, 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. At the same time, there is need for reducing the costs and risks related to the development of a DSML. Due to the contingency of the subject, deciding on the scope and economics of a DSML in advance needs to account for specific aspects of a particular case. Hence, the following criteria serve as general guidelines only that need to be interpreted against the background of a particular project.

To outline the scope of the targeted DSML, modelling projects can be categorized with respect to modelling subject and purpose, complexity and frequency. The higher the complexity and frequency of modelling a certain subject for certain purposes, 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. To develop an elaborate idea of the benefits to be expected from a DSML, a thorough requirements analysis is

mandatory. Unfortunately, analyzing the requirements a DSML should satisfy is a particular challenge. Often, prospective users will not be familiar with the idea of a DSML and, hence, will not know what they can expect from such an artefact. Our experience in developing DSML suggests that it is a promising approach to develop prototypical diagrams for certain analysis scenarios, e.g. the economic analysis of an IT infrastructure (see [Fran10]).

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 prospective 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 as well as the volatility 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.

3. Differentiating between Language and Language Application

In general, a domain-specific modelling language is aimed at providing its prospective users with technical terms suited to describe the targeted domain. However, sometimes it is not evident, whether a concept should be part of the language specification or instead be defined through the language. To illustrate this problem, it is useful to clarify the targeted level of abstraction. The quest for abstraction suggests to not represent instances – for a good reason: Instances may change over time, which would compromise a model's validity. On the other hand, models that represent the M_2 or even higher levels of abstraction are mainly restricted to specialized applications such as the specification of modelling languages. Therefore, we focus on the M_1 level, since we assume that the vast majority of conceptual models are assigned to it. As a consequence, designing a DSML requires analyzing whether a domain-level term is suited to be represented as a meta type – and therefore can be incorporated into the language – or not, which would suggest representing him in a model specified with the prospective DSML. In natural language, there is no explicit distinction between the type and the meta type level. Sometimes a term can be used on both levels and often it is used for representing instances, too. For instance: 'Business Process' can be instantiated into various business process types, which would qualify it as a meta type. One could argue that it could also represent a type that then could be specialized into further business process types. However, this is no option, since there is no suitable specialization concept for process types.[1] 'Organisational Unit' can be seen as the meta type of 'Department', which in turn could have instances like 'Sales Department', 'IT Service Group' etc. But does 'Sales

[1] While there are a few approaches to introduce relaxed concepts of process specialization, a specialization concept that accounts for substitutability is not feasible, because it would require monotonic extensions.

96

Department' need to represent an instance? Take for example a multi-national corporation that defines a certain organisational structure for all its national subsidiaries. In this case, 'Sales Department' may serve as a type. While it seems undisputed that it makes sense to model goals, it is not clear whether a goal like 'minimize throughput time' should be regarded as a meta type, a type or an instance. As a consequence, the respective design decisions are even more demanding: In some cases, the foundational presupposition that models are represented on the type level (M_1) has to be dismissed. For example: A DSML 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, hence, particular instances in a logistic network.

Table 1 illustrates the decision to be made when analyzing a term with respect to its suitability for being incorporated into a DSML. In the regular case, the decision will focus on the two alternatives in the upper row: whether a term should be represented as a meta type to become part of the language or whether it is supposed to be specified by the language. In some cases, it will be required to decide whether a term should rather be represented as a meta type or a type, where the latter implies that it is appropriate to represent instances in respective models.

DSML	model
Meta type 'Business Process'	*Type* 'Order Management'
Type 'Location'	*Instance* 'Berlin'

Table 1: Language vs. language application on different levels of abstraction

3.1 General Criteria

To develop guidelines for the above decision scenario, we will first look at criteria that need to be accounted for with every prospective language concept. Hence, we focus on the principle decision whether a term is suited to be incorporated in a language. To illustrate the criteria, we refer to examples form the domain of organisational analysis and design. Note that the following criteria do not account for the question whether a concept should be reconstructed as a type or a meta type, but only whether or not it is suited for becoming part of a DSML:

a) *Noteworthy level of invariant semantics*: If a term can be expected to have the same meaning across all application areas a DSML is supposed to cover, it can be regarded as a candidate for being incorporated in the language. In other words: It should be possible to define the essential meaning of the concept. Essential means that the features used to specify the concept are sufficient to capture at least a minimal set of characterizing features, i.e. features that would allow for distinguishing it from other concepts. Furthermore, one should know how to describe the range of its more concrete specifications. In other words: There should be substantial knowledge about the conceptual variance the

97

abstraction covers. These aspects are related to the level of semantics, a potential language concept includes. The level of semantics corresponds to its information content: The more interpretations are excluded, the higher the level of semantics. Hence, the level of semantics depends on the semantic determination of its features, i.e. its attributes and the associated (meta) types. The level of semantics embedded in a language concept is a core indicator of its value as an abstraction: The more we know about possible instances (types or instances), the more can we express about them on an abstract level – without having to care for the peculiarities of particular instances. If the meaning of a concept changes significantly with the application context, this could be handled only with a specification on a lower level of semantics. Alternatively, it may be a reasonable decision to reduce the scope of a language, i.e. to make it more special. Hence, applying this criterion is interrelated with determining a DSML's intended scope (see 2).

Test: At first sight, it may appear that the most satisfactory approach to test an assumption about a concept's meaning across the set of intended applications would be to conduct an elaborate and representative empirical investigation. However, such an approach faces two major drawbacks. First, it will often be not appropriate to focus on the actual meaning of a concept, but rather to pursue a semantic reconstruction that fits the purpose of the DSML. Second, for economic reasons, a comprehensive empirical investigation will usually be no option. Therefore, introspection (reflecting upon known uses of a term – which makes it empirical, too) and discursive evaluation will usually be the option of choice. In any case, the results of such a test should be regarded as a hypothesis that may still be refuted.

b) *Relevance*: If a concept is expected to be used on a regular base, making it part of the language would increase the language's value for most users. If it is required in rare cases only, it would increase the size of the language, hence make it more difficult to learn, while at the same time most users would not benefit from it. While this criterion is related to all other criteria, it is different in the sense that it puts explicit emphasis on the purpose of the language and the actual range of applications.

Test: Creating exemplary models within the scope of intended applications helps to find out how relevant the use of the concept would be with respect to intended purposes of corresponding models. Analyzing intended applications and related purposes may suggest empirical studies. However, such an approach has its limitations: It does not seem plausible to expect a comprehensive assembly of relevant purposes from prospective users if they are not familiar with the considered kind of modelling (which will be especially the case with DSMLs that target new domains).

98

3.2 Language Concept as Meta Type or Type?

In addition to determining whether a term is in principle suited to become part of a DSML, it needs to be decided whether it should be reconstructed as a meta type – which can be regarded as the regular case – or rather as a type. The first two criteria focus on checking the suitability as a meta type, while the third and fourth address the suitability as a type.

c) *Noteworthy semantic differences between types*: From a formal point of view, a meta type should allow for a range of types with clear semantic differences. If the types that can be instantiated from the potential meta type are all too similar, the effort it takes to define the types is questionable. This criterion, too, does not only apply to the entire range of intended applications, but also to the semantic variance of types within particular models: It does not contribute to the benefit of the abstraction in a particular case, if there are no noteworthy semantic differences between potential types. Again, if the variance is noteworthy only in some cases, while in others it is not, it would indicate the contingency of a subject.

   Test: From a formal point of view, the meta type should allow for a range of significantly different types. That implies that the types can be defined on a high level of semantics – otherwise there would not be much chance to express clear differences between types. If this is the case, one would have to check whether there is a noteworthy number of clearly different types within the range of possible types.

d) *Instance as type intuitive*: This criterion emphasizes language ergonomics. Would it correspond to the common use of the term to regard its instances as types? Hence, would instances be regarded as types intuitively by prospective language users or would they rather be interpreted as representing instances of the respective domain?

   Test: Since many people do not consciously differentiate between type and instance level in many cases, an empirical test would have to face a remarkable challenge. Therefore, referring to our *Sprachgefühl* (sense of language) and to its discursive evaluation will usually be the only feasible option.

The following criteria serve to check whether a term that does not qualify as a meta type could instead be represented as a type offered by a DSML.

e) *"Instance" as abstraction*: There are terms that emphasize abstractions by their very nature. Therefore, they resist against a clear distinction between the type and the instance level. For example: Is a goal such as 'minimize throughput time of a set of business processes' a type or rather an instance? It may be regarded as a type, since it applies to a set of processes. At the same time, it hardly allows for further instantiation. Nevertheless, it may be important to represent a corresponding goal in a model. In these cases, it is suitable to make a respective type, e.g. 'Goal', part of the language.

Test: Since there is hardly a formal criterion to guide this test, it depends chiefly on reflection and introspection to find out whether a term that does not allow for further instantiation can still be regarded as an abstraction.

f) *Invariant and unique instance identity*: Sometimes, it makes sense to represent objects in a model that clearly qualify as instances. Examples include cities, countries or particular organisations. Apparently, these examples have in common that the respective instances have a unique identity that is relevant, i.e. it makes a difference, for the modelling purpose. Also, the features that are relevant for the modelling purpose are (widely) invariant within the intended lifetime of a model (e.g. the geographical location of a city). Therefore, including them in a model does not mean to compromise the quest for abstraction.

Test: To test a term against this criterion, a number of models where it is represented as an instance need to be analyzed. The focus of the test is on the following questions: Are there useful analysis scenarios related to the intended type of models that require accounting for instances? Are these instances (widely) invariant with respect to the modelling purpose?

Against this background, we suggest the following guidelines.

Modelling Guideline R1: If a concept fulfils criteria a) to b), it should be considered for becoming part of the language.

Modelling Guideline R2: If a concept fulfils criteria c) and d), it should be considered for being represented as meta type.

Modelling Guideline R3: If a concept does not satisfy guideline R2 and fulfils either criterion e) or f), it should be considered for being represented as type.

To illustrate the use of the suggested criteria, we consider three terms selected from the examples discussed above. The following tables show the result of evaluating them against the criteria of the modelling guidelines. "+" indicates that the term clearly fulfils the corresponding criterion.

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.		
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

100

"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 guidelines R2 and R3 are considered only, if guideline R1 applies. R3 is relevant only, if R2 does not apply. 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.

Apparently, the term 'Organisational Unit' qualifies as language concept to be represented as a meta type. Nevertheless, it may be an option to include a more specific term in a DSML, e.g. 'Department': It may be regarded as a relevant term of the corresponding technical language and since it is more specific than 'Organisational Unit' it could increase a DSML's contribution to productivity and integrity.

Department		
A department is an organisational unit, which will usually include further organisational units. It may be part of a superordinate organisational unit.		
a) invariant semantics	As the description of the term indicates, there is not much that is specific to the term department, which would clearly distinguish it from the more general term organisational unit. Even on a high level of abstraction, the meaning of the term varies. In some cases, a department will be part of a superordinate head department or a division, in other cases not.	**- c**
b) relevance	This term is frequently used for describing organisation structures.	**+**

Since criterion a) is not satisfied, guideline R1 does not apply to 'Department'. Therefore, testing it against guidelines R2 and R3 is obsolete. Note, however, that criterion a) is contingent, which means that for some domains the term's semantics may be invariant (see 4.1).

Location		
A location represents a geographical area and a corresponding settlement such as a village or a city.		
a) invariant semantics	The essence of the term as it is outlined in the above description should not vary substantially.	**+**
b) relevance	In various domains, e.g. in cross-organisational logistics, this is a key term.	**+**
c) variance of instances	The semantics of instances may vary to some degree.	**o**
d) instance as type intuitive	It is conceivable to regard an instance as type, e.g. the type 'City' as opposed to the type 'Village'. Often, however, it will be more appropriate to instantiate directly into a particular location such as 'Berlin' or 'Hamburg'.	**o c**

101

Since guideline R2 does not clearly apply, R3 can be tested additionally. Note that in this case, it is assumed that the concept's instances represent particular instances such as cities.

e) "Instance" as abstraction	If an instance is a particular location, then this criterion is not satisfied.	-
f) invariant and unique instance identity	Cities, urban districts and villages satisfy this criterion without a doubt. Buildings will be regarded as invariant and unique in most applications, too.	+

The result of the last evaluation shows that 'Location' is suited as a language concept. The decision on the level of abstraction depends on the intended purpose. If models are supposed to represent particular locations, then the term is suited for being represented in a DSML as a type. Otherwise, a representation as meta type would be more preferable. Note, however, that there may be cases where it is required to account for both levels of abstraction (see 4).

Applying the suggested guidelines to a few examples produces an ambivalent result. On the one hand, the criteria help to reduce the complexity of decisions related to the design of DSML. On the other hand, they are not easy to apply. Instead, they require a thorough and differentiated interpretation that may be regarded as arduous by some. This is especially the case for those criteria that are marked as contingent. Therefore, the guidelines need to be interpreted with respect to the intended scope and purpose of a DSML. However, scope and purpose of a DSML are sometimes hard to determine in advance. Our experience suggests that developing modelling scenarios helps to get a better idea of scope and purpose – and to dissolve the principle contingencies related to some criteria.

4. Dealing with Conflicting Levels of Abstraction

The previous evaluation of the guidelines has shown that they do not always produce clear results, which requires further analysis. Sometimes, however, the results may even be partially contradictory with respect to the appropriate level of abstraction.

4.1 "Local" Types

There are terms that are not perfectly suited to be incorporated into a DSML, because their semantics is not invariant within the intended language scope. Nevertheless they may be important elements of respective technical languages - and may be useful for particular applications of a DSML. For example: The semantics of the term 'Department' is not invariant across a wider range of organisations. Therefore, it seems appropriate at first to not include it in a DSML for organisation modelling. Instead, the DSML could provide the more abstract concept 'Organisational Unit', which could be instantiated into types such as 'Marketing Department', 'Finance Department' etc. However, this would not be satisfactory either, because it would not be possible to express that a marketing

102

department and a finance department (both defined on the M_1 level) are two types of the same kind. To deal with these contradictory requirements, a trade-off is recommended, which is expressed in the following guideline:

> **Modelling Guideline R4**: If a term is not used consistently throughout the set of intended applications, it should not be incorporated into the language – in order to not violate modelling rule R2. However, if this term is an important part of the respective technical language and its semantics is fairly invariant within a certain application area, it can be represented as a "local type".

A local type is an auxiliary abstraction that allows for adapting a DSML to specific needs. In the case of 'Department', it would enable a company to define its own conception of department. Figure 2 illustrates the specification of a local type. The local types ‘Department’ and ‘Division’ are associated to instances of ‘OrganisationalUnit’ to specify their semantics in a specific local scope.

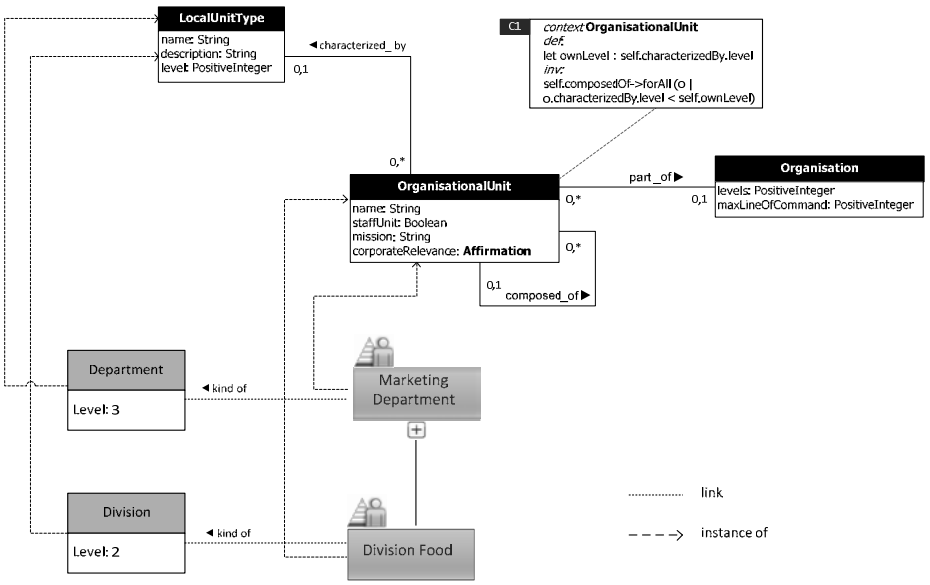

Figure 2: Illustration of „local types“

In order to protect model integrity, the local type of an organisational unit can be defined by an instance of an additional meta type (‘LocalUnitType’), which can be associated with an instance of ‘OrganisationalUnit’. ‘LocalUnitType’ provides the option to specify the hierarchical level and a characteristic designation. Furthermore, it would be possible to assign a specific graphical notation. A level is defined by a positive Integer. The lowest level, 0, is reserved for those organisational units that do not include further units and/or that do not include any position that is in command of further organisational units. If these levels are specified for all local types of organisational unit types, it is possible to enforce the constraints that they imply, e.g. that no organisational unit on level 2 can be part of a unit on level 1. The definition of a local terminology through

103

instantiations of ‘LocalUnitType’ can be regarded as defining language concepts with a limited scope, or as pseudo meta types. This approach can be further supported by supplementing a DSML with instantiations of ‘LocalUnitType’ for specific subdomains, e.g. for industrial enterprises, for public organisations, universities etc.

4.2 Intrinsic Features

On the one hand, specifying a meta model requires reflecting upon the ontological essence of a term. On the other hand, it recommends taking into account that instances of a meta concept are types. Sometimes, this results in the problem that the essence of a term includes features that do not apply directly to the type level. Instead, they apply to the instances represented by a type. For example: A language for modelling business processes includes the meta type ‘BusinessProcess’, which has attributes such as ‘name’, ‘averageExecutionTime‘ and various associations to other meta types like ‘Event’, ‘OrganisationalUnit’ etc. Within a particular model, it is instantiated to a certain business process type, e.g. ‘Order Management’, which includes the instantiation of attributes from corresponding meta types. While we know that every business process instance has a particular start and termination time, these materialized features do not apply to the corresponding business process type. Since a meta type may only define features that can be instantiated to describe features of a type, it is not possible to express features that apply to the instances of this type only. However, assigning these features to every instance would not only ignore an obvious abstraction, it would also result in redundancy. This problem is well known in conceptual modelling and addressed by concepts such as “power types” [Odel98] or “clabjects” [AtKü07]. A similar concept, called "intrinsic feature" is proposed in [Fran11] to enrich a meta modelling language. A (meta) type may have (regular) attributes that apply to its instances or “intrinsic attributes” that can be instantiated only from the instances of its instances. The concept can also be applied to an entire entity type (an instance of a meta type): An entity type that must not be instantiated directly, but only on the level below the one it is presented on, is called an “intrinsic type”. Figure 3 illustrates how to specify the example concepts ‘OrganisationalUnit’ and ‘Location’ with the use of intrinsic features. Intrinsic features are marked by an ‘i’ printed white on black. Specifying e.g. ‘Location’ with intrinsic features would allow for representing both, the semantics of location types such as ‘City’, ‘Village’ etc., and respective instances. A corresponding model would include instances, e.g. ‘Hamburg’, that are characterized by a type, e.g. ‘City’, which was defined by instantiating the meta type ‘Location’. Hence, a respective model editor would need to provide modellers with two levels of abstractions. Unfortunately not every meta modelling language features concepts for specifying instance level features. But even in these cases, it is a good idea to somehow add their description to the meta model – either by comments or by marking type level features.

> **Modelling Guideline R5**: If the essential meaning of a meta type that is part of a DSML includes aspects that apply to instances of the types of the meta type and these aspects are relevant for the intended application of the DSML, then these instance level features should be included in the specification. Preferably, this should be done with meta modelling languages that feature corresponding

104

concepts. Alternatively, instance level features can be marked or added in comments.

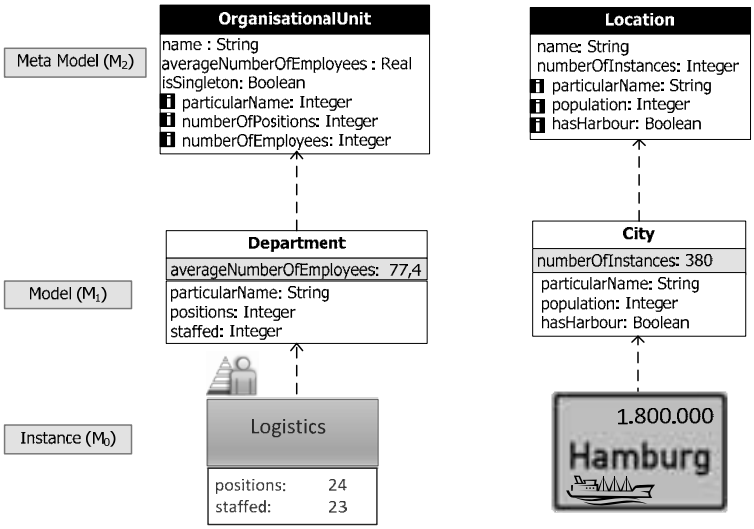

Figure 3: Example of Specification with Intrinsic Features

5. Concluding Remarks

In contrast to the remarkable attention generated by the idea of DSML, there has been only little work on specific design decisions related to the conception of a DSML. While there has been some work on differentiating instance and type level features (see 4.2), questions that concern the distinction of a prospective DSML and its application have hardly been addressed so far. This is the case, too, for the decision on how specific a DSML should be. This paper has shown that these questions are essential for the development – and economics – of DSMLs and very challenging at the same time. Due to the remarkable contingency of modelling subjects and purposes, it may not come as a surprise that the proposed guidelines do not serve as recipes. Instead, their application requires a considerable effort. As a consequence, they are mainly intended to improve the transparency of design decisions and draw the reflective language designer's attention to aspects that require further investigation. Also, they are meant to contribute to the discourse on the quality of meta models and DSML in general (see, e.g., [Fran98], [OpSe99], [FeLo03]). In addition to that, the guidelines suggest specific requirements for meta modelling languages and corresponding meta model editors that are not satisfied by most of today's solutions. Every guideline has been motivated with respect to certain presuppositions and has been tested against a few examples. Nevertheless, there is still need for further tests and discursive evaluations, since we are only at the beginning of developing expressive methods for designing DSML.

105

6. References

[AtKü08] Atkinson, C.; Kühne, T.: Reducing accidental complexity in domain models. In: Software and Systems Modelling. Vol. 7, No. 3, 2008, pp. 345-359

[Brin96] Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. In: Information and Software Technology, Vol. 38, No. 4, 1996, pp. 275–280

[DaPi+02] Dahchour, M.; Pirotte, A.; Zimányi; E.: Materialization and Its Metaclass Implementation. IEEE Transactions on Knowledge and Data Engineering Vol. 14, No. 5, pp. 1078-1094

[FeLo03] Fettke, P.; Loos, P.: Ontological evaluation of reference models using the Bunge-Wand-Weber-model. In: Proceedings of the Ninth Americas Conference on Information Systems. Tampa, FL 2003, pp. 2944-2955

[Fran11] Frank, U.: The MEMO Meta Modelling Language (MML) and Language Architecture. 2nd Ed., ICB Research Report No. 43, University Duisburg-Essen 2011

[Fran10] Frank, U.: Outline of a Method for Designing Domain-Specific Modelling Languages. ICB-Research Report No. 42, Universität Duisburg-Essen 2010

[Fran98] Frank, U.: Evaluating Modelling Languages: Relevant Issues, Epistemological Challenges and a Preliminary Research Framework. Research Report No. 15, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau 1998

[HeRa10] Henderson-Sellers, B.; Ralyté, J.: Situational Method Engineering: State-of-the-Art Review. In: Journal of Universal Computer Science, vol. 16, no. 3, 2010, pp. 424-478

[JaSz06] Jackson, E.; Sztipanovits, J.: Towards A Formal Foundation For Domain-specific Modelling Languages. In: Proceedings of the Sixth ACM International Conference on Embedded Software (EMSOFT'06), 2006, 53-63

[Jeus07] Jeusfeld, M.A.: Partial Evaluation in Meta Modelling. Proceedings of the IFIP 8.1 Working Conference on Situational Method Engineering (ME-2007), Geneva, 2007, Springer: Berlin, Heidelberg etc. 2007, pp. 115-129

[KeTo08] Kelly, S.; Tolvanen, J.-P.: Domain-Specific Modelling: Enabling full code generation. Wiley-IEEE Computer Society Press 2008

[Klep08] Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley 2008

[Odel98] Odell, J.: Power Types. In: Odell, J. (Ed.): Advanced Object-Oriented Analysis and Design Using UML. Cambridge University Press: Cambridge 1998, pp. 23-33 (revised version of: Odell, J.: Power Types. In: Journal of Object-Oriented Programming, Vol. 7, No. 2, 1994, pp. 8-12)

[OpHe99] Opdahl, A.L.; Henderson-Sellers, B.: Evaluating and Improving OO Modelling Languages Using the BWW-Model. In Proceedings of the Information Systems Foundations Workshop (Ontology, Semiotics and Practice), (digital publication), Sydney 1999

[Roll09] Rolland, C.: Method Engineering: Towards Methods as Services. In: Software Process Improvement and Practice. Vol. 14, 2009, pp. 143–164

[WaWe93] Wand, Y.; Weber, R.: On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. In: Journal of Information Systems, 1993, pp. 217-237