Under consideration for publication in Formal Aspects of Computing

A Formalisation of Deep Metamodelling

Alessandro Rossini[1], Juan de Lara[2], Esther Guerra[2], Adrian Rutle[3] and Uwe Wolter[4]

[1]Department of Networked Systems and Services, SINTEF, Oslo, Norway
[2]Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
[3]Faculty of Engineering and Natural Sciences, Aalesund University College, Norway
[4]Department of Informatics, University of Bergen, Norway

Abstract. Metamodelling is one of the pillars of model-driven engineering, used for language engineering and domain modelling. Even though metamodelling is traditionally based on a two-metalevel approach, several researchers have pointed out limitations of this solution and proposed an alternative *deep* (also called *multi-level*) approach to obtain simpler system specifications. However, this approach currently lacks a formalisation that can be used to explain fundamental concepts such as deep characterisation, double linguistic/ontological typing and linguistic extension. This paper provides such a formalisation based on the Diagram Predicate Framework, and discusses its practical realisation in the METADEPTH tool.

Keywords: model-driven engineering, multi-level metamodelling, deep metamodelling, deep characterisation, potency, double linguistic/ontological typing, linguistic extension, category theory, graph transformation, Diagram Predicate Framework, METADEPTH.

1. Introduction

Model-driven engineering (MDE) promotes the use of models as the primary assets in software development, where they are used to specify, simulate, generate and maintain software systems. Models can be specified using general-purpose languages like the Unified Modeling Language (UML) [Obj10b]. However, to fully unfold the potential of MDE, models are frequently specified using domain-specific languages (DSLs) which are tailored to a specific domain of concern. One way to define DSLs in MDE is by specifying metamodels, which are models that describe the concepts and define the (abstract) syntax of a DSL.

The Object Management Group (OMG) [Obj] has proposed the Meta-Object Facility (MOF) [Obj06] as the standard language to specify metamodels, and some popular implementations exist, most notably the Eclipse Modeling Framework (EMF) [SBPM08]. In this approach, a system is specified using models at two metalevels: a metamodel defining allowed types and a model instantiating these types. However, this approach may have limitations [AK02b, AK08, GPHS06], in particular when the metamodel includes the *type-object* pattern [AK02b, AK08, GPHS06], which requires an explicit modelling of types and their instances at the same metalevel. In this case, *deep metamodelling* (also called *multi-level metamodelling*) using more than two metalevels yields simpler models [AK08].

Correspondence and offprint requests to: alessandro.rossini@sintef.no, {Juan.deLara,Esther.Guerra}@uam.es, adru@hials.no, wolter@ii.uib.no

Deep metamodelling was proposed in the seminal works of Atkinson and Kühne [AK02b], and several researchers and tools have subsequently adopted this approach [ADP09, AM09, dG10, dGCML13]. However, there is still a lack of formalisation of the main concepts of deep metamodelling such as deep characterisation through potency, double linguistic/ontological typing and linguistic extension [dG10]. Such formalisation is needed in order to explain the main aspects of the approach, study the different semantic variation points and their consequences, as well as to classify the different semantics found in the tools implementing them [KS07, ADP09, AM09, AGK09, dG10, AGK12].

In this paper, we present a formal approach to deep metamodelling based on the Diagram Predicate Framework (DPF) [RRLW09a, RRLW09b, RRLW10a, RRLW10b, Rut10, RRM$^+$11, Ros11, RRLW12], a diagrammatic specification framework founded on category theory and graph transformation. DPF has been adopted up to now to formalise several concepts in MDE, such as (MOF-based) metamodelling, model transformation and model versioning. The proposed formalisation helps in reasoning about the different semantic variation points in the realisation of deep metamodelling, in classifying the existing tools according to these options, in expressing correctness constraints regarding deep instantiation, as well as in understanding the equivalences and relations between systems with and without deep characterisation.

This paper further develops the formalisation of deep metamodelling published in [RdG$^+$12]. Compared to the previous work, we extend it with a presentation of linguistic extensions. Moreover, we provide a declarative semantics of deep metamodelling (i.e., deep characterisation through potency, double linguistic/ontological typing and linguistic extension). Finally, we discuss an implementation of the proposed formalisation within the METADEPTH [dG10] tool.

The remainder of the paper is structured as follows. Section 2 illustrates the limitations of traditional metamodelling through an example in the domain of component-based web applications. Section 3 introduces deep metamodelling. Section 4 outlines DPF. Section 5 explains different concepts of deep metamodelling through its formalisation in DPF. Section 6 shows how deep metamodelling relates to traditional metamodelling by means of flattening constructions. Section 7 shows a practical implementation of deep metamodelling, discussing how the findings of the proposed formalisation affect this tool. In Section 8, the current research in deep metamodelling is summarised. In Section 9, some concluding remarks and ideas for future work are presented.

2. Metamodelling

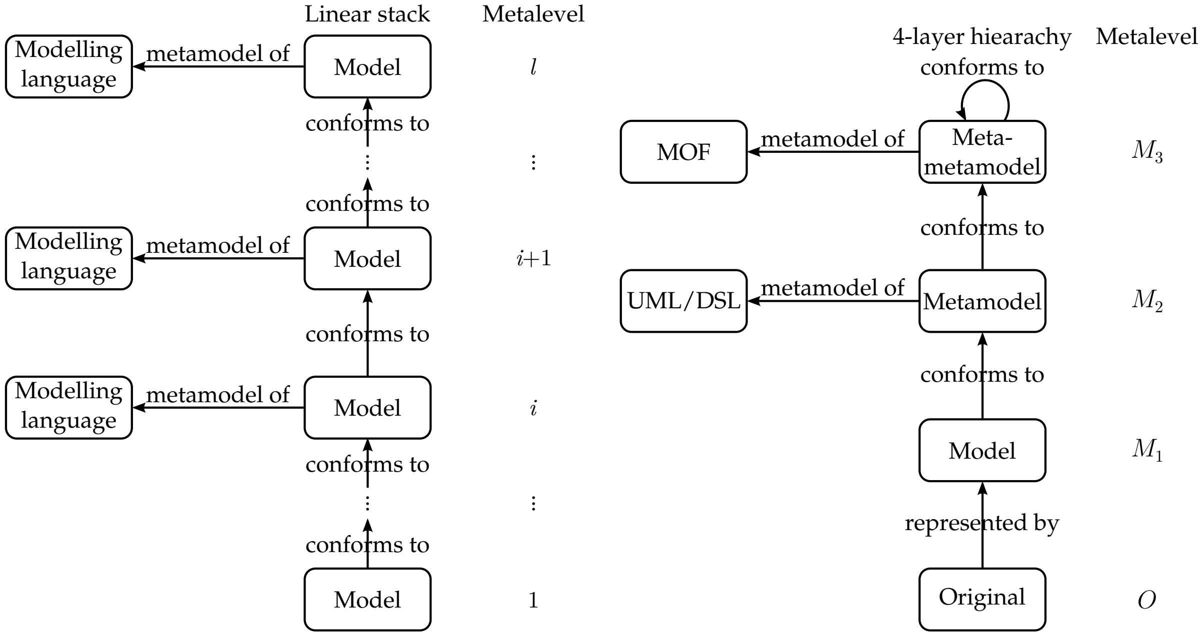
Metamodels are frequently used to define the (abstract) syntax of a modelling language, i.e., the set of modelling concepts, their attributes and their relationships, as well as the rules for combining these concepts to specify valid models [Obj10b]. Metamodels are specified using structural metamodelling languages such as the MOF. MOF-like metamodelling languages allow for the specification of simple constraints such as multiplicity and uniqueness constraints, hereafter called *structural constraints*. However, these structural constraints may not be sufficient to specify complex system requirements. Hence, metamodels are often complemented with textual constraint languages such as the Object Constraint Language (OCL) [Obj10a] to specify more complex constraints, hereafter called *attached constraints*.

A model is said to be *typed by* a metamodel if each element in the model is typed by an element in the metamodel, while a model is said to *conform to* a metamodel if it is typed by the metamodel and, in addition, satisfies all (structural and attached) constraints of the metamodel.

In a traditional *metamodelling stack* (or hierarchy), models at each metalevel *conform* to the corresponding metamodel of the modelling language at the adjacent metalevel above (see Figure 1(a)). This pattern is often referred to as *linear* metamodelling in the literature [AK02a]. Moreover, in *strict* metamodelling, a model element at each metalevel has exactly one type at the adjacent metalevel above. The top-most model of a traditional metamodelling stack may not conform to any model or may be a reflexive model, i.e., a model which conforms to itself. The length (or depth) of a traditional metamodelling stack is fixed (i.e., it cannot change depending on the requirements) and the metalevels are conventionally numbered from 1 onwards starting from the bottom-most.

For instance, in the 4-layer hierarchy [BG01] developed by the OMG, models conform to the metamodel of UML (see Figure 1(b)). The metamodel of UML, in turn, conforms to the metamodel of MOF [Obj06], and the latter is reflexive. Please note that *meta-* is a relative term, so that the UML metamodel is a model as well, while the MOF metamodel is a meta-metamodel with respect to the models.

The OMG's 4-layer hierarchy is the one most widely adopted in practice, but the designer is restricted to working with models at two metalevels only: a metamodel at metalevel M_2 corresponding to the modelling language (e.g., UML or a DSL), and a model at metalevel M_1 conforming to this metamodel. The following example illustrates that, on some occasions, the restriction to two metalevels leads to the introduction of accidental complexity, which could be avoided if the models were organised using more than two metalevels.



(a) Pattern in a linear metamodelling stack (b) OMG's 4-layer hierarchy

Figure 1. Linear metamodelling stack

Example 1 (A DSL for component-based web applications). One of the aims of the "Go Lite" project is the model-driven engineering of web applications. In the following, we describe a small excerpt of one of the modelling problems encountered in this project. Note that we use sans-serif fonts to denote model elements.

In "Go Lite", a DSL is adopted to define the mash-up of components (like Google Maps and Google Fusion Tables) to provide the functionality of a web application. A simplified version of this language can be defined using the metalevels M_2 and M_1 of the OMG's 4-layer hierarchy (see Figure 2).

The metamodel at metalevel M_2 corresponds to the DSL for component-based web applications. In this metamodel, the metaclass Component defines *component types* having a type identifier, whereas the metaclass CInstance defines *component instances* having a variable name and a flag indicating whether the instance should be visually rendered. Moreover, the metaassociation datalink defines the *data link types* between component types, whereas the metaassociation dlinstance defines the *data link instances* between component instances. Finally, the metaassociation type defines the typing of each component instance.

The model at metalevel M_1 represents a component-based web application which shows the position of professors' offices on a map. In this model, the classes Map and Table are instances of the metaclass Component and represent component types, whereas the classes UAMCamp and UAMProfs are instances of the metaclass CInstance and represent component instances of Map and Table, respectively. The association geopos is an instance of the metaassociation datalink and represents the allowed data link between the component types Map and Table, whereas the association offices is an instance of the metaassociation dlinstance and represents the actual data link between the component instances UAMCamp and UAMProfs. Finally, the associations camptype and profstype are instances of the metaassociation type and represent the typing of the component instances UAMCamp and UAMProfs, respectively.

The type-object relation between component types and instances is represented explicitly in the metamodel by the metaassociation type between the metaclasses Component and CInstance. However, the type-object relation between data link types and instances is implicit since there is no explicit relation between the metaassociations datalink and dlinstance, and this may lead to several problems. Firstly, it is not possible to define that the data link instance offices is typed by the data link type geopos, which could be particularly ambiguous if the model contained multiple data link types between the component types Map and Table. Moreover, it could be possible to specify a reflexive data link instance from the component instance UAMProfs to itself, which should not be allowed since the component type Table does not have any reflexive data link type. Although these errors could be detected by complementing the metamodel with attached OCL constraints, these constraints would not be enough to *guide*

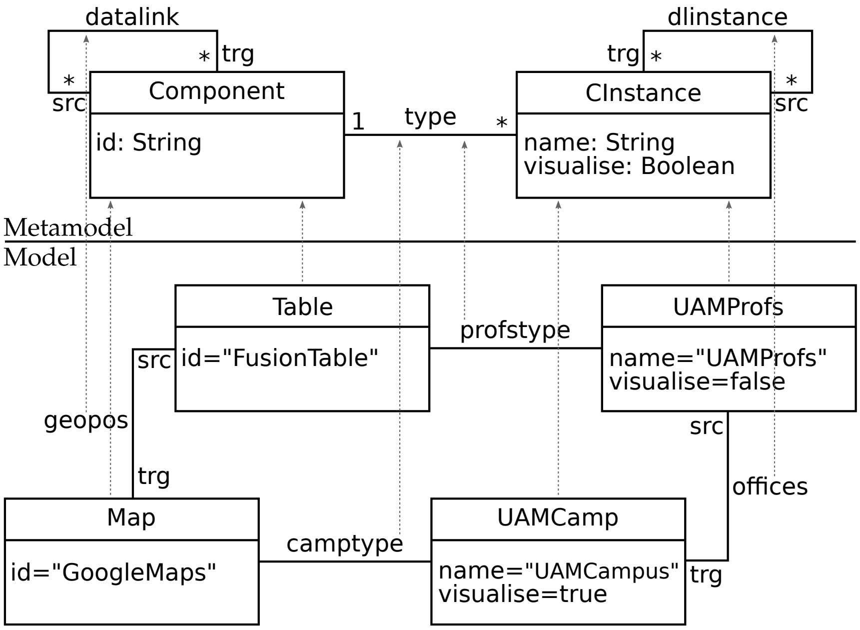


Figure 2. A two-metalevel DSL for component-based web applications

the correct instantiation of each data link, in the same way as a built-in type system would do if the data link types and instances belonged to two different metalevels. This is because while violation of the attached OCL constraints would be detected in a reactive manner, the built-in type system would hamper the violation of typing constraints in a proactive manner.

In the complete definition of the DSL, the component types can define features which need to be correctly instantiated in the component instances. This leads to even more cluttered models (see Figure 3). In the model, the class Scroll is associated to the class Map and represents the zooming capabilities of the map component. The definition of the class UAMScroll and its association to both the classes UAMCamp as well as Scroll has to be done manually. Moreover, the conformance check that the value "true" assigned to the attribute value is actually a boolean has to be done manually as well. Hence, either one builds manually the needed machinery to emulate the existence of two metalevels within the same one, or this two-metalevel solution eventually becomes convoluted and hardly usable.

In the following, we show that organising the models in three metalevels results in a simpler and more usable DSL.

3. Deep metamodelling

This section introduces the main concepts of deep metamodelling, illustrating how they overcome the problems of the two-metalevel approach when defining DSLs which incorporate the type-object pattern.

3.1. Deep characterisation

The first ingredient of deep metamodelling is *deep characterisation*: the ability to describe structure and express constraints for metalevels below the adjacent one. In this work, we adopt the deep characterisation approach described in [AK02b]. In this approach, each element has a *potency*. In the original proposal of [AK02b], the potency is a natural number which is attached to a model element to describe at how many subsequent metalevels this element can be instantiated. Moreover, the potency decreases in one unit at each instantiation at a deeper metalevel. When it reaches zero, a pure instance that cannot be instantiated further is obtained. In Section 5, we provide a more precise definition for potency.

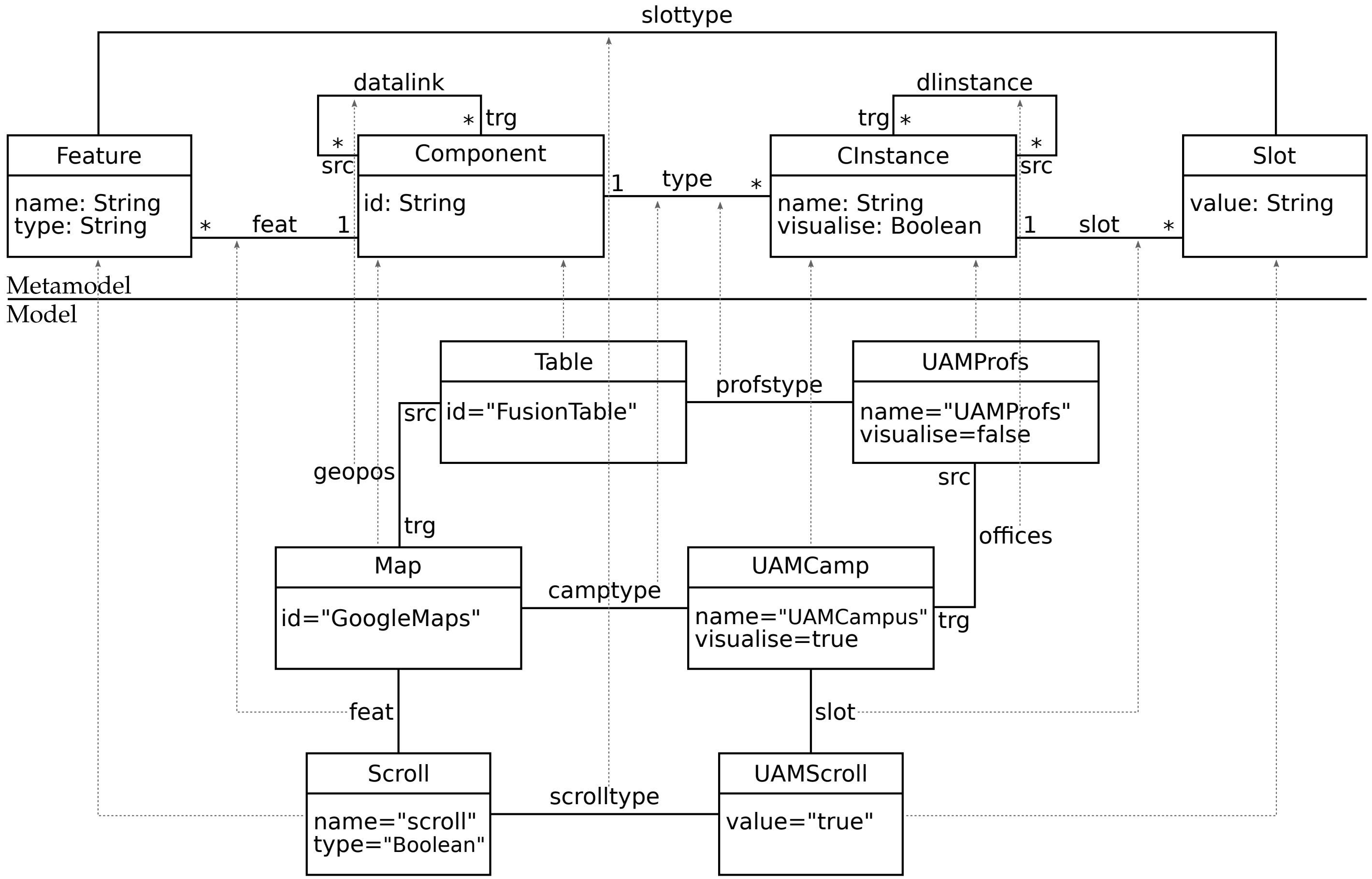


Figure 3. Extension of the two-metalevel DSL adding component features

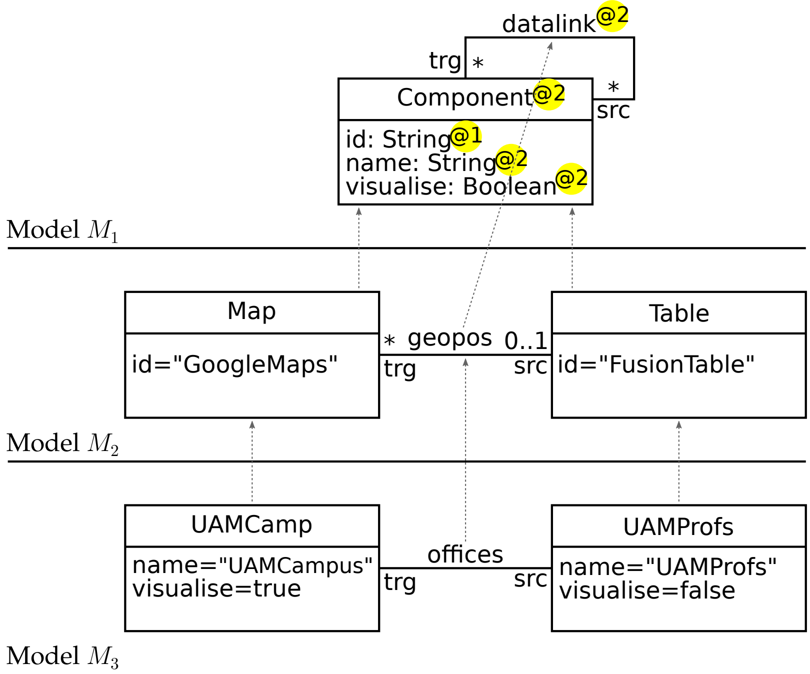


Figure 4. A three-metalevel DSL for component-based web applications corresponding to the DSL in Figure 2

In deep metamodelling, the elements at the top metalevel are pure types, the elements at the bottom metalevel are pure instances, and the elements at intermediate metalevels retain both a type and an instance facet. Because of that, they are all called *clabjects*, which is the merge of the words class and object [AK08]. Since in deep metamodelling the number of metalevels may change depending on the requirements, we find it more convenient to number the metalevels from 1 onwards starting from the top-most, in contrast to the traditional metamodelling stack (see Figure 1(a)).

The following example illustrates the usage of deep characterisation.

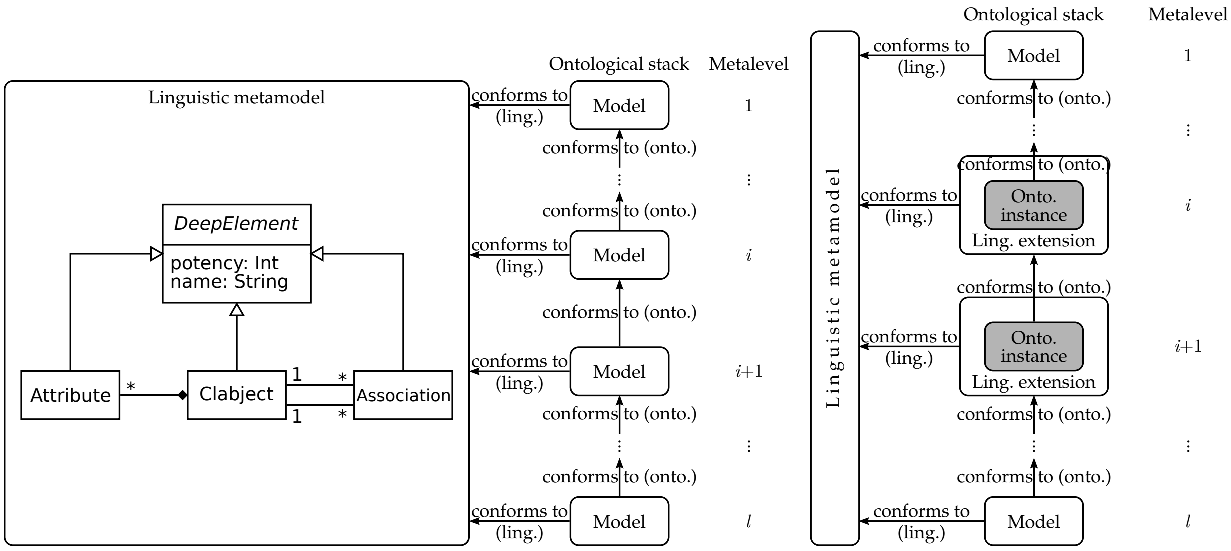
Example 2 (A DSL for component-based web applications in three metalevels). Compared to Example 1, the DSL for component-based web applications can be defined in a simpler way using deep metamodelling (see Figure 4).

The model M_1 contains the definition of the DSL. In this model, the clabject Component has potency 2, which denotes that it can be instantiated at the two subsequent metalevels. Its attribute id has potency 1, which denotes that it can be assigned a value when Component is instantiated at the adjacent metalevel below. Its other two attributes name and visualise have potency 2, which denotes that they can be assigned a value only two metalevels below. The association datalink also has potency 2, which denotes that it can be instantiated at the two subsequent metalevels. Please note that, at the intermediate metalevel, association geopos retains a type facet and hence its ends can be decorated with cardinalities to control the multiplicities of its instances. Altogether, the DSL in Figure 4 is simpler than the one in Figure 2, as it contains less model elements to define the same DSL.

In this example, the deep characterisation enabled us to specify the attributes name and visualise in M_1, which should be assigned values in indirect instances of Component, i.e., UAMCamp and UAMProfs. Moreover, we did not need to include the clabject CInstance or the association dlinstance in the model M_1 in order to emulate the instantiation of instances of Component and Datalink since this could be taken care of by the built-in type system.

3.2. Double typing and linguistic extension

The dashed grey arrows in Fig. 4 denote the *ontological typing*, which represents an instantiation within a domain; e.g., the clabjects Map and Table are ontologically typed by the clabject Component. In addition, deep metamodelling frameworks usually support an orthogonal *linguistic typing* [AK08, dG10], which represents an instantiation within a linguistic modelling language used to specify the models at all metalevels of the ontological stack.



(a) Metamodelling stack with double linguistic/ontological typing

(b) Metamodelling stack with double linguistic/ontological typing and linguistic extension

Figure 5. Double linguistic/ontological typing and linguistic extension

Figure 5(a) shows the scheme of this double linguistic/ontological typing. Moreover, it shows a simplified linguistic metamodel, which contains some of the metaclasses needed to specify models, e.g., clabjects, attributes and associations.

In Figure 4, the clabjects Component, Map and UAMCamp are linguistically typed by the metaclass Clabject, whereas the attributes id, name and visualise are linguistically typed by the metaclass Attribute. The availability of a double linguistic/ontological typing has the advantage that one can uniformly treat all clabjects independently of their ontological type and metalevel. This enables the specification of *generic* model manipulations typed by the linguistic metamodel, which then become applicable to models at any metalevel.

The double linguistic/ontological typing also enables so-called *linguistic extensions* [dG10]. These extensions are a useful mechanism to extend existing metamodelling stacks by adding at intermediate metalevels new clabjects and/or attributes (to existing clabjects) which are only linguistically typed. This solution improves the scalability of deep metamodelling since it facilitates addressing new requirements at intermediate metalevels which could not be foreseen or addressed at the top-most metalevel. Figure 5(b) shows the scheme of linguistic extensions. All models in the ontological stack conform linguistically to the linguistic metamodel, but only portions of them conform ontologically to the model at the adjacent metalevel above.

The following example illustrates the usage of linguistic extensions.

Example 3 (Extended DSL for component-based web applications in three metalevels). As discussed in Example 1, the component types can define features which need to be correctly instantiated in the component instances. These new features can be naturally expressed as linguistic extensions in the model M_2 (see Figure 6). In particular, the clabject Map is extended with an attribute scroll of type Boolean. This linguistic extension reflects the fact that the clabject Map retains both a type and an instance facet. The attribute scroll has potency 1, which denotes that it can be assigned a value in the model M_3.

Figure 6 also shows that potency can be attached to constraints as well. The attached OCL constraint in the model M_1 forbids to reflexively connect indirect instances of Component (see Figure 6). This constraint has potency 2, which denotes that it has to be evaluated in the model M_3 only.

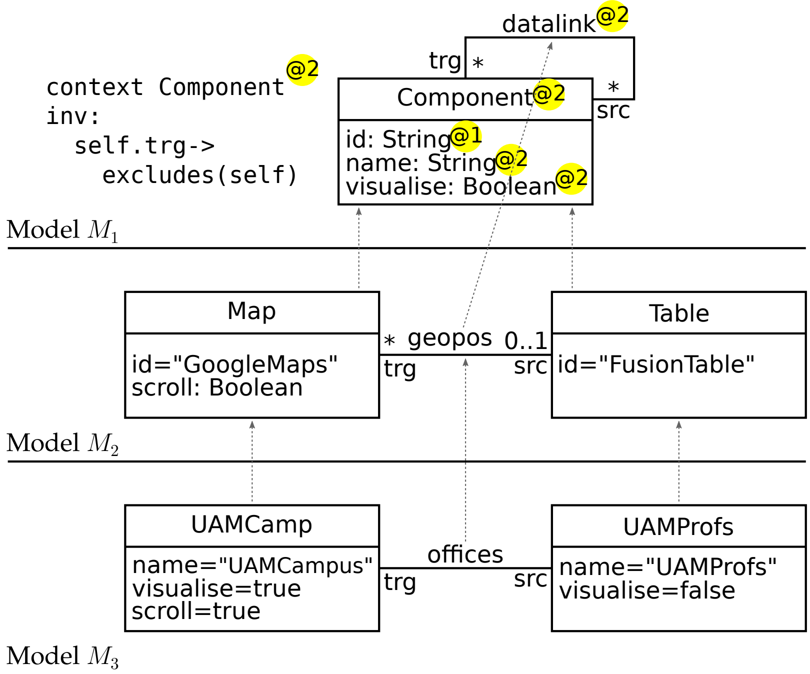


Figure 6. Linguistic extension of the three-metalevel DSL adding component features

Regarding the handling of features of component types, the solution presented in Example 3 has two main advantages with respect to the solution in Example 1. Firstly, linguistic extensions enable the use of a built-in type system to check the conformance of feature types and instances; e.g., the conformance check that the value true assigned to the attribute scroll is actually a boolean. Secondly, the built-in type system is used to guide the instantiation of clabjects; e.g., when the clabject Map is instantiated, all its attributes are instantiated as well. In Example 1, the correct instantiation was done either manually or by additional machinery needed to emulate the existence of two metalevels within the same one.

In the following, we discuss some open questions in deep metamodelling.

3.3. Some open questions in deep metamodelling

Deep metamodelling allows a more flexible approach to metamodelling by introducing richer modelling mechanisms. However, their semantics have to be precisely defined in order to obtain sound, robust models. Even if the literature (and this section) permits grasping an intuition of how these modelling mechanisms work, there are still open questions which require clarification.

Some works in the literature give different semantics to the potency of associations. In Example 3, the associations are instantiated like clabjects. In this case, the association datalink with potency 2 in the model M_1 is first instantiated as the association geopos with potency 1 in the model M_2, and then instantiated as the association offices with potency 0 in the model M_3 (see Figure 6); i.e., the instantiation of offices is mediated by geopos. This means that one cannot create an indirect instance of datalink with potency 0 in the model M_3 if there is not an instance with potency 1 in the model M_2. In contrast, the attributes name and visualise with potency 2 in the model M_1 are assigned a value directly in the model M_3 (see Figure 6); i.e., the instantiation of name and visualise is not mediated. Some frameworks such as EMF [Ecl, SBPM08] represent associations as Java references, so the associations could also be instantiated like attributes. In this case, the association datalink would not need to be instantiated in the model M_2 in order to be able to instantiate it in the model M_3. This would have the effect that one could add an association between any two component instances in the model M_3, not necessarily between instances of Table and instances of Map. Hence, the question is whether the instantiation of associations should be mediated or not.

Another ambiguity concerns constraints, since some works in the literature support potency on constraints [dG10] but others do not [AGK09]. In Example 3, the attached OCL constraint in the model M_1 is evaluated in the model M_3 only; i.e., it is not evaluated in the model M_2. In other cases, it might be useful to have a potency which denotes that a constraint has to be evaluated at every metalevel. In Example 3, none of the multiplicity constraints has potency and they are all evaluated at the adjacent metalevel below. In other cases, it might be useful to attach a potency to multiplicity constraints. For instance, a potency 2 on the multiplicity constraints of the association datalink in the model M_1 would have the effect that one could control the number of data link instances in the model M_3.

Finally, another research question concerns the relation between metamodelling stacks with and without deep characterisation. One could define constructions to *flatten* deep characterisation; e.g., given the three-metalevel stack of Example 3, one could obtain another three-metalevel stack without potencies but with some elements replicated along metalevels, making explicit the semantics of potency. This would allow the migration of deeply characterised systems into tools that do not support deep characterisation. One could also define further constructions to flatten multiple metalevels into two or to eliminate the double typing.

Altogether, we observe a lack of consensus and precise semantics for some of the aspects of deep metamodelling. The contribution of this work is the use of DPF to provide a neat semantics for the different aspects of deep metamodelling: double linguistic/ontological typing (see Section 5.1), linguistic extension (see Section 5.2) and deep characterisation through potency (see Section 5.3). As a distinguishing note, we propose two possible semantics of potency for each model element, i.e., clabjects, attributes, associations and constraints. This proposal recognises the different instantiation semantics described in the literature ("clabject-like" and "attribute-like", see Section 5.3), generalising them to enable their application to every model element. To the best of our knowledge, this is the first time that the two semantics have been recognised and formalised.

4. Diagram Predicate Framework

DPF is a generalisation and adaptation of the categorical sketch formalism [BW95], where the constraining constructs of modelling languages are represented by user-defined predicates in a more intuitive and adequate way. In particular, DPF is an extension of the Generalized Sketches Framework originally developed by Diskin et al. in [Dis97, Dis96, DKPJ00, Dis02, DK03, Dis03, DK05, Dis05]. This section presents the basic concepts of DPF that are used in the formalisation of deep metamodelling. The interested reader can consult [DW08, RRLW10a, Rut10, Ros11, RRLW12] for a more detailed presentation of the framework.

4.1. Graph and graph homomorphism

In a first approximation, diagrammatic models can be represented by graphs of different kinds, e.g., simple graphs, bipartite graphs, directed graphs, directed multi-graphs, attributed graphs, hypergraphs, etc. Graphs are a well-known and well-understood means to represent structural and behavioural properties of software systems [EEPT06]. In this paper, we adopt directed multi-graphs.

A directed multi-graph consists of a set of nodes together with a set of edges, where multiple edges between the same source and target nodes are permitted. Graphs are related by graph homomorphisms. A graph homomorphism consists of a pair of maps from the nodes and edges of a graph to those of another graph, where the maps preserve the source and target of each edge.

Definition 1 (Graph). A graph $G = (G_N, G_A, src^G, trg^G)$ consists of a set G_N of nodes (or vertices), a set G_A of edges (or arrows) and two maps $src^G, trg^G : G_A \to G_N$ assigning the source and target to each edge, respectively. $f : X \to Y$ denotes that $src(f) = X$ and $trg(f) = Y$.

Definition 2 (Subgraph). A graph $G = (G_N, G_A, src^G, trg^G)$ is subgraph of a graph $H = (H_N, H_A, src^H, trg^H)$, written $G \sqsubseteq H$, if and only if $G_N \subseteq H_N$, $G_A \subseteq H_A$ and $src^G(f) = src^H(f)$, $trg^G(f) = trg^H(f)$, for all $f \in G_A$.

Definition 3 (Graph homomorphism). A graph homomorphism $\phi : G \to H$ consists of a pair of maps $\phi_N : G_N \to H_N$, $\phi_A : G_A \to H_A$ which preserve the sources and targets, i.e., for each edge $f : X \to Y$ in G we have $\phi_A(f) : \phi_N(X) \to \phi_N(Y)$ in H.

Remark 1 (Inclusion graph homomorphism). $G \sqsubseteq H$ if and only if the inclusion maps $inc_N : G_N \hookrightarrow H_N$ and $inc_A : G_A \hookrightarrow H_A$ define a graph homomorphism $inc : G \hookrightarrow H$.

Having defined graphs and graph homomorphisms, it is natural to consider all graphs and graph homomorphisms as objects and morphisms, respectively, of a category [BW95, Fia04]. The category of graphs is defined as follows:

Definition 4 (Category of graphs). The category **Graph** has all graphs G as objects and all graph homomorphisms $\phi : G \to H$ as morphisms between graphs G and H.

The composition $\phi; \psi : G \to K$ of two graph homomorphisms $\phi : G \to H$ and $\psi : H \to K$ is defined component-wise $\phi; \psi = (\phi_N, \phi_A); (\psi_N, \psi_A) := (\phi_N; \psi_N, \phi_A; \psi_A)$. The identity graph homomorphisms $id^G : G \to G$ are also defined component-wise $id^G = (id^{G_N}, id^{G_A})$. This ensures that the composition of graph homomorphisms is associative and that identity graph homomorphisms are left and right neutral with respect to composition.

The semantics of nodes and edges of a graph has to be chosen in a way which is appropriate for the corresponding modelling environment [RRLW12]. In object-oriented structural modelling, each object may be related to a set of other objects. Hence, it is appropriate to interpret nodes as sets and edges $\mathsf{X} \xrightarrow{\mathsf{f}} \mathsf{Y}$ as multi-valued functions $f : X \to \wp(Y)$. The powerset $\wp(Y)$ of Y is the set of all subsets of Y, i.e., $\wp(Y) = \{A \mid A \subseteq Y\}$. Moreover, the composition of two multi-valued functions $f : X \to \wp(Y)$, $g : Y \to \wp(Z)$ is defined by $(f; g)(x) := \bigcup\{g(y) \mid y \in f(x)\}$.

The semantics of a graph can be formally defined in either an *indexed* or a *fibred* way [Dis05, DW08]. In the indexed version, the semantics of a graph is given by all graph homomorphisms $sem : G \to \mathbf{U}$ from the graph G into a category $\mathbf{U}$, e.g., **Set** (sets as objects and functions as morphisms) or **Mult** (sets as objects and multi-valued functions as morphisms as described above). In the fibred version, the semantics of a graph is given by the set of its instances. An instance (I, ι) of a graph G consists of a graph I together with a graph homomorphism $\iota : I \to G$.

Although the usage of graphs for the representation of diagrammatic models is a success story, an enhancement of the formal basis is needed to specify diagrammatic constraints and define a conformance relation between models which takes into account these constraints.

4.2. Signature and specification

In DPF, a model is represented by a *specification* $\mathfrak{S}$. A specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ consists of an *underlying graph* S together with a set of *atomic constraints* $C^{\mathfrak{S}}$ which are specified by means of a *signature* Σ. A signature $\Sigma = (\Pi^\Sigma, \alpha^\Sigma)$ consists of a set of *predicates* $\pi \in \Pi^\Sigma$, each having an arity (or shape graph) $\alpha^\Sigma(\pi)$. An atomic constraint (π, δ) consists of a predicate $\pi \in \Pi^\Sigma$ together with a graph homomorphism $\delta : \alpha^\Sigma(\pi) \to S$ from the arity of the predicate to the underlying graph of the specification.

Definition 5 (Signature). A signature $\Sigma = (\Pi^\Sigma, \alpha^\Sigma)$ consists of a set of predicate symbols Π^Σ and a map α^Σ which assigns a graph to each predicate symbol $\pi \in \Pi^\Sigma$. $\alpha^\Sigma(\pi)$ is called the *arity* of the predicate symbol π.

Definition 6 (Atomic constraint). Given a signature $\Sigma = (\Pi^\Sigma, \alpha^\Sigma)$, an atomic constraint (π, δ) on a graph S consists of a predicate symbol $\pi \in \Pi^\Sigma$ and a graph homomorphism $\delta : \alpha^\Sigma(\pi) \to S$.

Definition 7 (Specification). Given a signature $\Sigma = (\Pi^\Sigma, \alpha^\Sigma)$, a specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ consists of a graph S and a set $C^{\mathfrak{S}}$ of atomic constraints (π, δ) on S with $\pi \in \Pi^\Sigma$.

The following example illustrates the usage of signatures and specifications to represent object-oriented structural models.

Example 4 (Signature and specification). Let us consider the system introduced in Examples 1 and 2. For the sake of illustration, assume that this system should satisfy the following requirements:

1. A component must have exactly one identifier.
2. A component may be connected to other components.
3. A component can not be connected to itself.

Figure 7 shows a specification $\mathfrak{T} = (T, C^{\mathfrak{T}} : \Sigma)$ which is compliant with the requirements above. Moreover, Figure 7 shows a signature $\Sigma = (\Pi^\Sigma, \alpha^\Sigma)$. The first column of the table shows the predicate symbols. The second and the third columns show the arities of predicates and a proposed visualisation of the corresponding atomic constraints, respectively. Finally, the fourth column presents the semantic interpretation of each predicate.

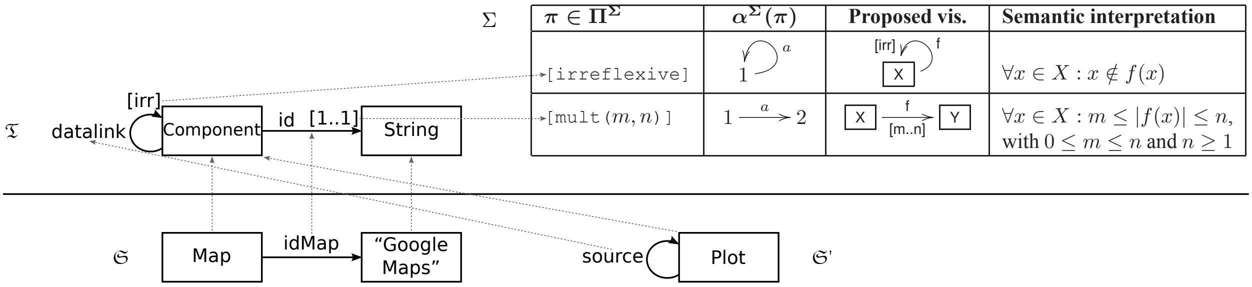


Figure 7. A signature Σ and specifications $\mathfrak{T}$, $\mathfrak{S}$ and $\mathfrak{S}'$, where only $\mathfrak{S}$ conforms to $\mathfrak{T}$

Table 1. The atomic constraints $(\pi, \delta) \in C^{\mathfrak{T}}$ and their graph homomorphisms

(π, δ)	$\alpha^{\Sigma}(\pi)$	$\delta(\alpha^{\Sigma}(\pi))$
$(\texttt{[mult(}1,1\texttt{)]}, \delta_1)$	$1 \xrightarrow{a} 2$	Component $\xrightarrow{\text{id}}$ String
$(\texttt{[irreflexive]}, \delta_2)$	$1 \circlearrowleft a$	Component $\circlearrowleft$ datalink

In $\mathfrak{T}$, the system requirements are enforced by the atomic constraints $(\texttt{[mult(}1,1\texttt{)]}, \delta_1)$ and $(\texttt{[irreflexive]}, \delta_2)$. The graph homomorphisms δ_1 and δ_2 are defined as follows (see Table 1):

$$\delta_1(1) = \mathsf{Component}, \quad \delta_1(2) = \mathsf{String}, \quad \delta_1(a) = \mathsf{id}$$
$$\delta_2(1) = \mathsf{Component}, \quad \delta_2(a) = \mathsf{datalink}$$

Remark 2 (Predicate symbols). Some of the predicate symbols in Σ (see Figure 7) refer to single predicates, e.g., [irreflexive], while some others refer to a family of predicates, e.g., $\texttt{[mult(}m,n\texttt{)]}$. In the case of $\texttt{[mult(}m,n\texttt{)]}$, the predicate is parametrised by the (non-negative) integers m and n, which represent the lower and upper bounds, respectively, of the cardinality of the function which is constrained by this predicate.

The semantics of predicates of the signature Σ (see Figure 7) is described using the mathematical language of set theory. In an implementation, the semantics of a predicate is typically given by code for a so-called validator where both the mathematical and the validator semantics should coincide; i.e., the validator accepts a given instance of a predicate if and only if the instance is accepted according to the mathematical semantics. However, it is not necessary to choose between the above mentioned possibilities; it is sufficient to know that any of these possibilities defines valid instances of predicates.

Definition 8 (Semantics of predicates). Given a signature $\Sigma = (\Pi^{\Sigma}, \alpha^{\Sigma})$, a semantic interpretation $[\![..]\!]^{\Sigma}$ of Σ consists of a mapping that assigns to each predicate symbol $\pi \in \Pi^{\Sigma}$ a set $[\![\pi]\!]^{\Sigma}$ of graph homomorphisms $\iota : O \to \alpha^{\Sigma}(\pi)$, called valid instances of π, where O may vary over all graphs. $[\![\pi]\!]^{\Sigma}$ is assumed to be closed under isomorphisms.

The semantics of a specification is defined in the fibred way [Dis05, DW08]; i.e., the semantics of a specification $\mathfrak{S} = (S, C^{\mathfrak{S}}\!:\!\Sigma)$ is given by the set of its instances (I, ι). An instance (I, ι) of a specification $\mathfrak{S}$ consists of a graph I together with a graph homomorphism $\iota : I \to S$ which satisfies the set of atomic constraints $C^{\mathfrak{S}}$.

To check that an atomic constraint is satisfied in a given instance of a specification $\mathfrak{S}$, it is enough to inspect only the part of $\mathfrak{S}$ which is affected by the atomic constraint. This kind of restriction to a subpart is obtained by the pullback construction [BW95, Fia04], which can be regarded as a generalisation of the inverse image construction.

Definition 9 (Instance of a specification). Given a specification $\mathfrak{S} = (S, C^{\mathfrak{S}}\!:\!\Sigma)$, an instance (I, ι) of $\mathfrak{S}$ consists of a graph I and a graph homomorphism $\iota : I \to S$ such that for each atomic constraint $(\pi, \delta) \in C^{\mathfrak{S}}$ we have $\iota^* \in [\![\pi]\!]^{\Sigma}$, where the graph homomorphism $\iota^* : O^* \to \alpha^{\Sigma}(\pi)$ is given by the following pullback:

$$\begin{tikzcd} \alpha^{\Sigma}(\pi) \arrow[r, "\delta"] & S \\ O^* \arrow[u, "\iota^*"] \arrow[r, "\delta^*"'] \arrow[ur, phantom, "P.B."] & I \arrow[u, "\iota"'] \end{tikzcd}$$

Given a specification $\mathfrak{S}$, the category of instances of $\mathfrak{S}$ is defined as follows:

Definition 10 (Category of instances). Given a specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$, the category $\textsf{Inst}(\mathfrak{S})$ has all instances (I, ι) of $\mathfrak{S}$ as objects and all graph homomorphisms $\phi : I \to I'$ as morphisms between instances (I, ι) and (I', ι'), such that $\iota = \phi; \iota'$.

$$\begin{tikzcd} & S & \\ I \arrow[ur, "\iota"] \arrow[rr, "\phi"'] & = & I' \arrow[ul, "\iota'"'] \end{tikzcd}$$

$\textsf{Inst}(\mathfrak{S})$ is a full subcategory of $\textsf{Inst}(S)$ where $\textsf{Inst}(S) = (\textsf{Graph} \downarrow S)$ is the comma category of all graphs typed by S [BW95]; i.e., we have an inclusion functor $inc^{\mathfrak{S}} : \textsf{Inst}(\mathfrak{S}) \hookrightarrow \textsf{Inst}(S)$.

4.3. Typing and conformance

In DPF, a specification $\mathfrak{S}$ is said to be typed by a graph T if there exists a graph homomorphism $\iota : S \to T$, called the *typing morphism*, between the underlying graph of the specification $\mathfrak{S}$ and the graph T. A specification $\mathfrak{S}$ is said to conform to a specification $\mathfrak{T}$ if there exists a typing morphism $\iota : S \to T$ between the underlying graphs of $\mathfrak{S}$ and $\mathfrak{T}$ such that (S, ι) is a valid instance of $\mathfrak{T}$; i.e., such that ι satisfies the atomic constraints $C^{\mathfrak{T}}$.

Definition 11 (Typed specification). Given a signature $\Sigma = (\Pi^{\Sigma}, \alpha^{\Sigma})$ and a graph T, a specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ typed by T is a specification $\mathfrak{S}$ together with a graph homomorphism $\iota : S \to T$, called the *typing morphism*.

Definition 12 (Conformant specification). Given two signatures $\Theta = (\Pi^{\Theta}, \alpha^{\Theta})$, $\Sigma = (\Pi^{\Sigma}, \alpha^{\Sigma})$ and a specification $\mathfrak{T} = (T, C^{\mathfrak{T}} : \Theta)$, a specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ which conforms to $\mathfrak{T}$ is a specification $\mathfrak{S}$ together with a typing morphism $\iota : S \to T$ such that $(S, \iota) \in \textsf{Inst}(\mathfrak{T})$.

Example 5 (Typing and conformance). Figure 7 shows two specifications $\mathfrak{S}$ and $\mathfrak{S}'$, both typed by $\mathfrak{T}$. However, only $\mathfrak{S}$ conforms to $\mathfrak{T}$, since $\mathfrak{S}'$ violates the atomic constraints $C^{\mathfrak{T}}$: the multiplicity constraint $(\texttt{[mult(1,1)]}, \delta_1)$ is violated since there is no attribute in $\mathfrak{S}'$ which is an instance of id, while the irreflexivity constraint $(\texttt{[irreflexive]}, \delta_2)$ is violated since there is a reflexive reference source which is an instance of datalink.

4.4. Specification morphism

In DPF, the relation between specifications is represented by *specification morphisms*. Specification morphisms are graph homomorphisms between the underlying graphs of specifications. These graph homomorphisms induce a translation of instances of graphs.

Proposition 1 (Translation of instances of graphs). Each graph homomorphism $\phi : S \to S'$ induces a functor $\phi_{\bullet} : \textsf{Inst}(S) \to \textsf{Inst}(S')$ with $\phi_{\bullet}(I, \iota) = (I, \iota; \phi)$ for all $(I, \iota) \in \textsf{Inst}(S)$.

$$\begin{tikzcd} S \arrow[r, "\phi"] & S' \\ I \arrow[u, "\iota"] \arrow[ur, "\iota;\phi"'] & \end{tikzcd}$$

$$\textsf{Inst}(S) \xrightarrow{\phi_{\bullet}} \textsf{Inst}(S')$$

Moreover, each graph homomorphism $\phi : S \to S'$ induces a functor $\phi^\bullet : \mathbf{Inst}(S') \to \mathbf{Inst}(S)$ with $\phi^\bullet(I', \iota')$ given by the pullback $(I^*, \phi^* : I^* \to I', \iota^* : I^* \to S)$ of the span $S \xrightarrow{\phi} S' \xleftarrow{\iota'} I'$ [DW08].

$$\begin{tikzcd} S \arrow[r, "\phi"] & S' \\ I^* \arrow[u, "\iota^*"] \arrow[r, "\phi^*"'] \arrow[ur, phantom, "P.B."] & I' \arrow[u, "\iota'"'] \end{tikzcd}$$

$$\mathbf{Inst}(S) \xleftarrow{\phi^\bullet} \mathbf{Inst}(S')$$

In addition, these graph homomorphisms should preserve atomic constraints.

Definition 13 (Specification morphism). Given two specifications $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ and $\mathfrak{S}' = (S', C^{\mathfrak{S}'} : \Sigma)$, a specification morphism $\phi : \mathfrak{S} \to \mathfrak{S}'$ is a graph homomorphism $\phi : S \to S'$ such that $(\pi, \delta) \in C^{\mathfrak{S}}$ implies $(\pi, \delta; \phi) \in C^{\mathfrak{S}'}$.

$$\begin{tikzcd} \alpha^\Sigma(\pi) \arrow[r, "\delta"] \arrow[rr, bend left, "\delta;\phi"] & S \arrow[r, "\phi"] \arrow[u, phantom, "="] & S' \end{tikzcd}$$

Remark 3 (Subspecification). A specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ is a subspecification of a specification $\mathfrak{S}' = (S', C^{\mathfrak{S}'} : \Sigma)$, written $\mathfrak{S} \sqsubseteq \mathfrak{S}'$, if and only if S is a subgraph of S' and the inclusion graph homomorphism $inc : S \hookrightarrow S'$ defines a specification morphism $inc : \mathfrak{S} \hookrightarrow \mathfrak{S}'$.

Remark 4 (Graph homomorphism and atomic constraints). Any graph homomorphism $\phi : S \to S'$ induces a translation of atomic constraints; i.e., for any specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ we obtain a specification $\phi(\mathfrak{S}) = (S', C^{\phi(S)}{:}\Sigma)$ with $C^{\phi(S)} = \phi(C^{\mathfrak{S}}) = \{(\pi, \delta; \phi) \mid (\pi, \delta) \in C^{\mathfrak{S}}\}$.

Based on this remark, the condition for specification morphisms can be reformulated as follows: a specification morphism $\phi : \mathfrak{S} \to \mathfrak{S}'$ is a graph homomorphism $\phi : S \to S'$ such that $\phi(\mathfrak{S}) \sqsubseteq \mathfrak{S}'$, i.e., $C^{\phi(S)} = \phi(C^{\mathfrak{S}}) \subseteq C^{\mathfrak{S}'}$.

Given a signature Σ, the category of specifications is defined as follows:

Definition 14 (Category of specifications). Given a signature $\Sigma = (\Pi^\Sigma, \alpha^\Sigma)$, the category $\mathbf{Spec}(\Sigma)$ has all specifications $\mathfrak{S} = (S, C^{\mathfrak{S}}{:}\Sigma)$ as objects and all specification morphisms $\phi : \mathfrak{S} \to \mathfrak{S}'$ as morphisms between specifications $\mathfrak{S}$ and $\mathfrak{S}'$.

The associativity of composition of graph homomorphisms ensures that the composition of two specification morphisms is a specification morphism as well and that the composition of specification morphisms is associative. Moreover, the identity graph homomorphisms $id^S : S \to S$ define identity specification morphisms $id^{\mathfrak{S}} : \mathfrak{S} \to \mathfrak{S}$ and ensure that identity specification morphisms are left and right neutral with respect to composition.

Proposition 2 (Specification morphisms and category of instances). For any specification morphism $\phi : \mathfrak{S} \to \mathfrak{S}'$, we have $\phi^\bullet(\mathbf{Inst}(\mathfrak{S}')) \subseteq \mathbf{Inst}(\mathfrak{S})$; i.e., the functor $\phi^\bullet : \mathbf{Inst}(S') \to \mathbf{Inst}(S)$ restricts to a functor $\phi^\bullet : \mathbf{Inst}(\mathfrak{S}') \to \mathbf{Inst}(\mathfrak{S})$.

$$\begin{tikzcd} S \arrow[d, "\phi"'] & \mathbf{Inst}(S) & \mathbf{Inst}(\mathfrak{S}) \arrow[l, hook'] & \mathfrak{S} \arrow[d, "\phi"] \\ S' & \mathbf{Inst}(S') \arrow[u, "\phi^\bullet"] \arrow[ur, phantom, "="] & \mathbf{Inst}(\mathfrak{S}') \arrow[l, hook'] \arrow[u, "\phi^\bullet"'] & \mathfrak{S}' \end{tikzcd}$$

Proof. The proof is given by the result that the composition of two pullbacks is again a pullback [BW95] and by the assumption that $[\![\pi]\!]^{\Sigma}$ is closed under isomorphisms (see Definition 8), as shown in [DW08].

$$
\begin{tikzcd}
\alpha^{\Sigma}(\pi) \arrow[r, "\delta"] \arrow[rr, bend left, "\delta;\phi"] & S \arrow[r, "\phi"] & S' \\
O \arrow[u, "\iota^*"] \arrow[r, "\delta^*"'] \arrow[rr, bend right, "\delta^*;\phi^*"'] \arrow[ur, phantom, "P.B."] & I \arrow[u, "\iota"'] \arrow[r, "\phi^*"'] \arrow[ur, phantom, "P.B."] & I' \arrow[u, "\iota'"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\alpha^{\Sigma}(\pi) \arrow[r, "\delta;\phi"] & S' \\
O^{\bullet} \arrow[u, "\iota^{\bullet}"] \arrow[r, "(\delta;\phi)^*"'] \arrow[ur, phantom, "P.B."] & I' \arrow[u, "\iota'"']
\end{tikzcd}
$$

□

5. Formalisation of deep metamodelling

This section presents a formalisation of deep metamodelling based on DPF. This formalisation is presented stepwise by defining and illustrating double linguistic/ontological conformance, linguistic extension and deep characterisation.

5.1. Double metamodelling stack

A *double metamodelling stack* is a metamodelling stack which supports double linguistic/ontological conformance. Recall that in a double metamodelling stack, models at each metalevel conform linguistically to the corresponding metamodel of a fixed linguistic modelling language, and conform ontologically to the model at the adjacent metalevel above (see Section 3).

The metamodel of the linguistic modelling language of a deep metamodelling stack can be represented in DPF by a specification $\mathfrak{LM} = (LM, C^{\mathfrak{LM}} : \Sigma)$ which consists of an underlying graph LM and a set of atomic constraints $C^{\mathfrak{LM}}$ specified by means of a predicate signature Σ.

A model at metalevel i of a double metamodelling stack can be represented in DPF by a specification $\mathfrak{S}_i = (S_i, C_i : \Omega)$ which consists of an underlying graph S_i and a set of atomic constraints C_i specified by means of a predicate signature Ω. Moreover, $\mathfrak{S}_i$ conforms linguistically to the specification $\mathfrak{LM}$; i.e., there exists a total linguistic typing morphism $\lambda_i : S_i \to LM$ such that (S_i, λ_i) is a valid instance of $\mathfrak{LM}$. Furthermore, $\mathfrak{S}_i$ conforms ontologically to the specification $\mathfrak{S}_{i-1}$; i.e., there exists a total *two-level* ontological typing morphism $\omega_i : S_i \to S_{i-1}$ such that the ontological typing is compatible with the linguistic typing and (S_i, ω_i) is a valid instance of $\mathfrak{S}_{i-1}$.

First, in order to enable reuse later in the paper, the linguistic portion of the double metamodelling stack is defined as follows:

Definition 15 (Linguistic metamodelling stack). Given:

- signatures $\Sigma = (\Pi^{\Sigma}, \alpha^{\Sigma})$, $\Omega = (\Pi^{\Omega}, \alpha^{\Omega})$
- a specification $\mathfrak{LM} = (LM, C^{\mathfrak{LM}}{:}\Sigma)$

A linguistic metamodelling stack with length l consists of:

- specifications $\mathfrak{S}_i = (S_i, C_i{:}\Omega)$, for all $1 \leq i \leq l$
- total linguistic typing morphisms $\lambda_i : S_i \to LM$, for all $1 \leq i \leq l$, such that:
  - $(S_i, \lambda_i) \in \mathbf{Inst}(\mathfrak{LM})$

$$
\begin{tikzcd}
\Sigma \arrow[r, dashed, "C^{\mathfrak{LM}}"] & LM & S_1 \arrow[l, "\lambda_1"'] & \Omega \arrow[l, dashed, "C_1"'] \arrow[ddl, dashed, "C_{i-1}"'] \arrow[dddl, dashed, "C_i"] \arrow[dddddl, dashed, "C_l"] \\
 & & \vdots & \\
 & & S_{i-1} \arrow[uul, "\lambda_{i-1}"'] & \\
 & & S_i \arrow[uuul, "\lambda_i"] & \\
 & & \vdots & \\
 & & S_l \arrow[uuuuul, "\lambda_l"] &
\end{tikzcd}
$$

Note that a linguistic metamodelling stack is similar to a traditional linear metamodelling stack with two metalevels, where each specification $\mathfrak{S}_i$ conforms to the specification $\mathfrak{LM}$. Based on this, the double metamodelling stack is constructed by adding ontological typing morphisms $\omega_i : S_i \to S_{i-1}$ to the linguistic metamodelling stack, as follows:

Definition 16 (Double metamodelling stack). A double metamodelling stack with length l is a linguistic metamodelling stack with length l together with:

- total two-level ontological typing morphisms $\omega_i : S_i \to S_{i-1}$, for all $2 \leq i \leq l$, such that:
  - $\omega_i ; \lambda_{i-1} = \lambda_i$
  - $(S_i, \omega_i) \in \mathsf{Inst}(\mathfrak{S}_{i-1})$

$$
\begin{tikzcd}
\Sigma \arrow[r, dashed, "C^{\mathfrak{LM}}"] & LM & S_1 \arrow[l, "\lambda_1"'] & \Omega \arrow[l, dashed, "C_1"'] \arrow[ddl, dashed, "C_{i-1}"'] \arrow[dddl, dashed, "C_i"] \arrow[dddddl, dashed, "C_l"] \\
 & & \vdots \arrow[u, "\omega_2"'] & \\
 & \arrow[r, phantom, "="] & S_{i-1} \arrow[uul, "\lambda_{i-1}"'] \arrow[u, "\omega_{i-1}"'] & \\
 & & S_i \arrow[uuul, "\lambda_i"] \arrow[u, "\omega_i"'] & \\
 & & \vdots \arrow[u, "\omega_{i+1}"'] & \\
 & & S_l \arrow[uuuuul, "\lambda_l"] \arrow[u, "\omega_l"'] &
\end{tikzcd}
$$

The following example illustrates the usage of a double metamodelling stack.

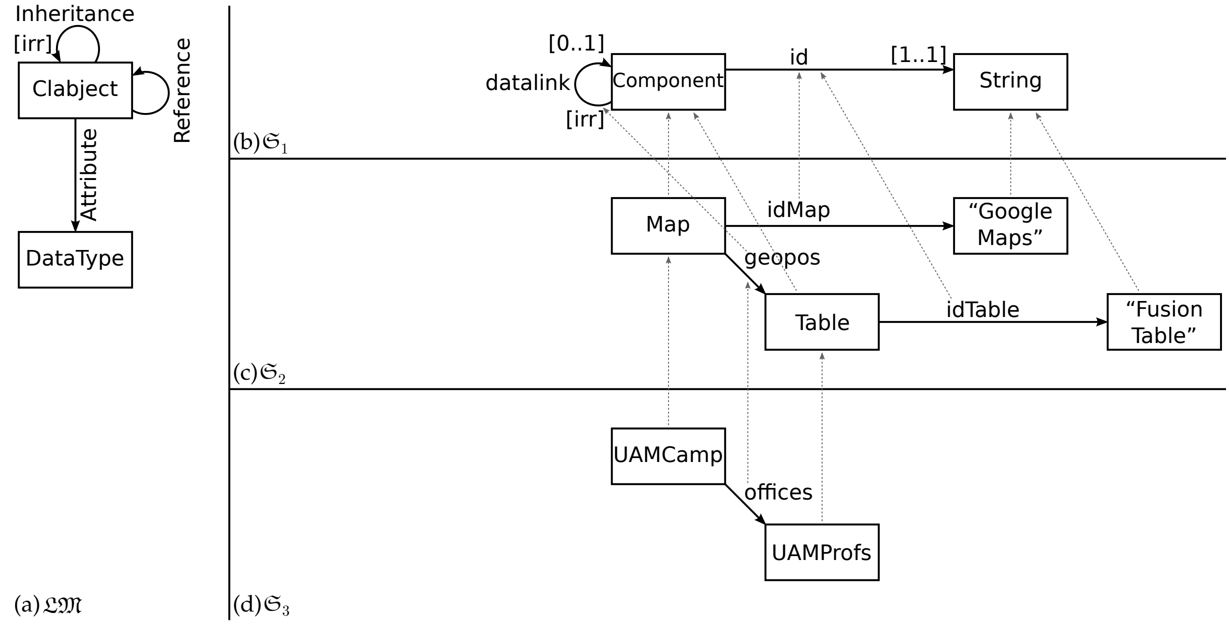


Figure 8. Double metamodelling stack for the example, showing the specifications $\mathfrak{LM}$, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ together with the ontological typing morphisms ω_2 and ω_3

Table 2. The signature Σ

$\pi \in \Pi^{\Sigma}$	$\alpha^{\Sigma}(\pi)$	**Proposed vis.**	**Semantic interpretation**
`[irreflexive]`	$1 \circlearrowleft a$	[irr] f $\boxed{X}$	$\forall x \in X : x \notin f(x)$

Example 6 (Double metamodelling stack). Building upon Example 2, Figure 8(a) shows the specification $\mathfrak{LM}$ and Figures 8(b), (c) and (d) show the specifications $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$, respectively, of a double metamodelling stack. Moreover, Figure 8 shows the ontological typing morphisms ω_2 and ω_3 as dashed grey arrows. Tables 2 and 3 show the signatures Σ and Ω, respectively.

The specification $\mathfrak{LM}$ corresponds to a metamodelling language for object-oriented structural modelling similar to the one in Figure 5(a). The interested reader may consult [Rut10] for details about the semantics of inheritance in DPF.

Table 3. The signature Ω

$\pi \in \Pi^{\Omega}$	$\alpha^{\Omega}(\pi)$	**Proposed vis.**	**Semantic interpretation**
`[mult(`m, n`)]`	$1 \xrightarrow{a} 2$	$\boxed{X} \xrightarrow[\text{[m..n]}]{f} \boxed{Y}$	$\forall x \in X : m \leq \lvert f(x) \rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
`[irreflexive]`	$1 \circlearrowleft a$	[irr] f $\boxed{X}$	$\forall x \in X : x \notin f(x)$

The specifications $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ conform linguistically to $\mathfrak{LM}$; i.e., there exist linguistic typing morphisms $\lambda_1 : S_1 \to LM$, $\lambda_2 : S_2 \to LM$ and $\lambda_3 : S_3 \to LM$ such that (S_1, λ_1), (S_2, λ_2) and (S_3, λ_3) are valid instances of $\mathfrak{LM}$. The linguistic typing morphisms λ_1, λ_2 and λ_3 are defined as follows:

$\lambda_1(\mathsf{Component}) = \mathsf{Clabject}$
$\lambda_1(\mathsf{datalink}) = \mathsf{Reference}$
$\lambda_1(\mathsf{id}) = \mathsf{Attribute}$
$\lambda_1(\mathsf{String}) = \mathsf{DataType}$
$\lambda_2(\mathsf{Map}) = \lambda_2(\mathsf{Table}) = \mathsf{Clabject}$
$\lambda_2(\mathsf{geopos}) = \mathsf{Reference}$
$\lambda_2(\mathsf{idMap}) = \lambda_2(\mathsf{idTable}) = \mathsf{Attribute}$
$\lambda_2(\text{“}\mathsf{GoogleMaps}\text{”}) = \lambda_2(\text{“}\mathsf{FusionTable}\text{”}) = \mathsf{DataType}$
$\lambda_3(\mathsf{UAMCamp}) = \lambda_3(\mathsf{UAMProfs}) = \mathsf{Clabject}$
$\lambda_3(\mathsf{offices}) = \mathsf{Reference}$

Moreover, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ conform ontologically to $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively; i.e., there exist total two-level ontological typing morphisms $\omega_2 : S_2 \to S_1$ and $\omega_3 : S_3 \to S_2$ such that (S_2, ω_2) and (S_3, ω_3) are valid instances of $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively, and commute with the linguistic typing morphisms. The ontological typing morphisms ω_2 and ω_3 are defined as follows:

$\omega_2(\mathsf{Map}) = \omega_2(\mathsf{Table}) = \mathsf{Component}$
$\omega_2(\mathsf{geopos}) = \mathsf{datalink}$
$\omega_2(\mathsf{idMap}) = \omega_2(\mathsf{idTable}) = \mathsf{id}$
$\omega_2(\text{“}\mathsf{GoogleMaps}\text{”}) = \omega_2(\text{“}\mathsf{FusionTable}\text{”}) = \mathsf{String}$
$\omega_3(\mathsf{UAMCamp}) = \mathsf{Map}$
$\omega_3(\mathsf{UAMProfs}) = \mathsf{Table}$
$\omega_3(\mathsf{offices}) = \mathsf{geopos}$

The proposed double metamodelling stack conveniently represents linguistic and ontological typing, but lacks support for linguistic extension and deep characterisation.

Firstly, in Example 3, the attribute scroll constitutes a linguistic extension of the model at metalevel 2 as this element is only typed linguistically. In Example 6, in contrast, $\mathfrak{S}_2$ can not include an attribute scroll which is not ontologically typed by an element in $\mathfrak{S}_1$. This is because the proposed double metamodelling stack has total ontological typing morphisms rather than partial ones.

Moreover, in Example 3, the deep characterisation of the elements Component and datalink at metalevel 1 forbids that these elements are instantiated at metalevel 4 or below. In Example 6, in contrast, one could add a specification $\mathfrak{S}_4$ including elements that are ontologically typed by elements in $\mathfrak{S}_3$.

Furthermore, in Example 3, the deep characterisation of the attribute name at metalevel 1 allows that this element is instantiated (i.e., it is assigned a value) at metalevel 3. In Example 6, in contrast, $\mathfrak{S}_3$ can not include elements which are ontologically typed by a possible attribute name in $\mathfrak{S}_1$ since $\mathfrak{S}_3$ is ontologically typed by $\mathfrak{S}_2$ but not by $\mathfrak{S}_1$.

Finally, in Example 3, the deep characterisation of the OCL constraint ensures that this constraint is evaluated at metalevel 3. In Example 6, in contrast, the atomic constraint $(\texttt{[irreflexive]}, \delta_2)$ corresponding to the OCL constraint above is evaluated in $\mathfrak{S}_2$ but not in $\mathfrak{S}_3$. This is because $\mathfrak{S}_2$ conforms ontologically to $\mathfrak{S}_1$, while $\mathfrak{S}_3$ conforms ontologically to $\mathfrak{S}_2$ but not to $\mathfrak{S}_1$.

In the following, we revise the definition of the double metamodelling stack to support linguistic extension as well as different mechanisms of deep characterisation.

5.2. Partial double metamodelling stack

A *partial double metamodelling stack* is a metamodelling stack which supports double linguistic/ontological conformance and linguistic extension. Recall that in a partial double metamodelling stack, models at each metalevel conform linguistically to the metamodel of a fixed linguistic modelling language, but only a portion of the same models conform ontologically to the model at the adjacent metalevel above (see Section 3); i.e., there can be elements in a model which are only linguistically typed.

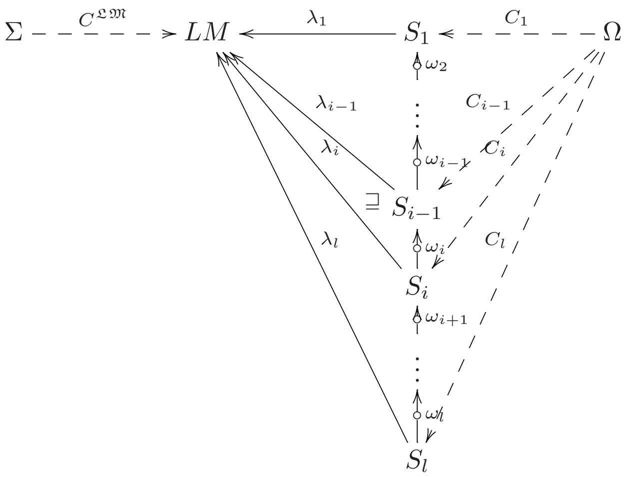
In analogy to the double metamodelling stack, a model at metalevel i of a partial double metamodelling stack can be represented in DPF by a specification $\mathfrak{S}_i = (S_i, C_i \colon \Omega)$ which conforms linguistically to the specification $\mathfrak{LM}$. In contrast to the double metamodelling stack, however, only a subgraph of $\mathfrak{S}_i$ conforms ontologically to the specification $\mathfrak{S}_{i-1}$; i.e., there exists a partial two-level ontological typing morphism $\omega_i : S_i \multimap S_{i-1}$ which is given by a subgraph $I_i \sqsubseteq S_i$ representing the domain of definition of ω_i (see Definition 24) and a total two-level ontological typing morphisms $\omega_i : I_i \to S_{i-1}$, such that the ontological typing is compatible with the linguistic typing and (I_i, ω_i) is a valid instance of $\mathfrak{S}_{i-1}$.

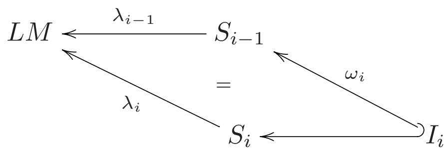
The partial double metamodelling stack is defined as follows:

Definition 17 (Partial double metamodelling stack). A partial double metamodelling stack with length l is a linguistic metamodelling stack with length l together with:

- partial two-level ontological typing morphisms $\omega_i : S_i \multimap S_{i-1}$, for all $2 \leq i \leq l$, which are given by:
  - domain of definition subgraphs $I_i \sqsubseteq S_i$
  - total two-level ontological typing morphisms $\omega_i : I_i \to S_{i-1}$

  such that:
  - $\omega_i ; \lambda_{i-1} \sqsubseteq \lambda_i$
  - $(I_i, \omega_i) \in \mathsf{Inst}(\mathfrak{S}_{i-1})$





Remark 5 (Composition of partial two-level ontological typing morphisms). Note that partial two-level ontological typing morphisms $\omega_k : S_k \multimap S_{k-1}$ can be composed to obtain a partial *multi-level* ontological typing morphism $\omega^i_k : S_k \multimap S_i$, for all $1 \leq i < k \leq l$, which is given by a subgraph $I^i_k \sqsubseteq S_k$ representing the domain of definition of ω^i_k and a total multi-level ontological typing morphism $\omega^i_k : I^i_k \to S_i$, where $\omega^i_k = \omega_k; \ldots; \omega_{i-1}$, $I^i_k = (\omega^i_k)^{-1}(S_i) \sqsubseteq I_k$ and $I^i_k \sqsubseteq \ldots \sqsubseteq I^{k-1}_k = I_k$ (see Definition 24).

Example 7 (Partial double metamodelling stack). Figure 9(a) shows the specification $\mathfrak{LM}$ and Figures 9(b), (c) and (d) show the specifications $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$, respectively, of a partial double metamodelling stack. Moreover, Figure 9 shows the ontological typing morphisms ω_2 and ω_3 as dashed grey arrows.

Compared to Example 6, the specification $\mathfrak{S}_2$ is extended with an attribute scroll with data type Boolean, while the specification $\mathfrak{S}_3$ is extended with a corresponding data value true. The linguistic typing morphisms λ_1, λ_2 and λ_3 are extended with the following mappings:

$\lambda_2(\text{scroll}) = \text{Attribute}$
$\lambda_2(\text{Boolean}) = \text{DataType}$
$\lambda_3(\text{scrollUAM}) = \text{Attribute}$
$\lambda_3(\text{true}) = \text{DataType}$

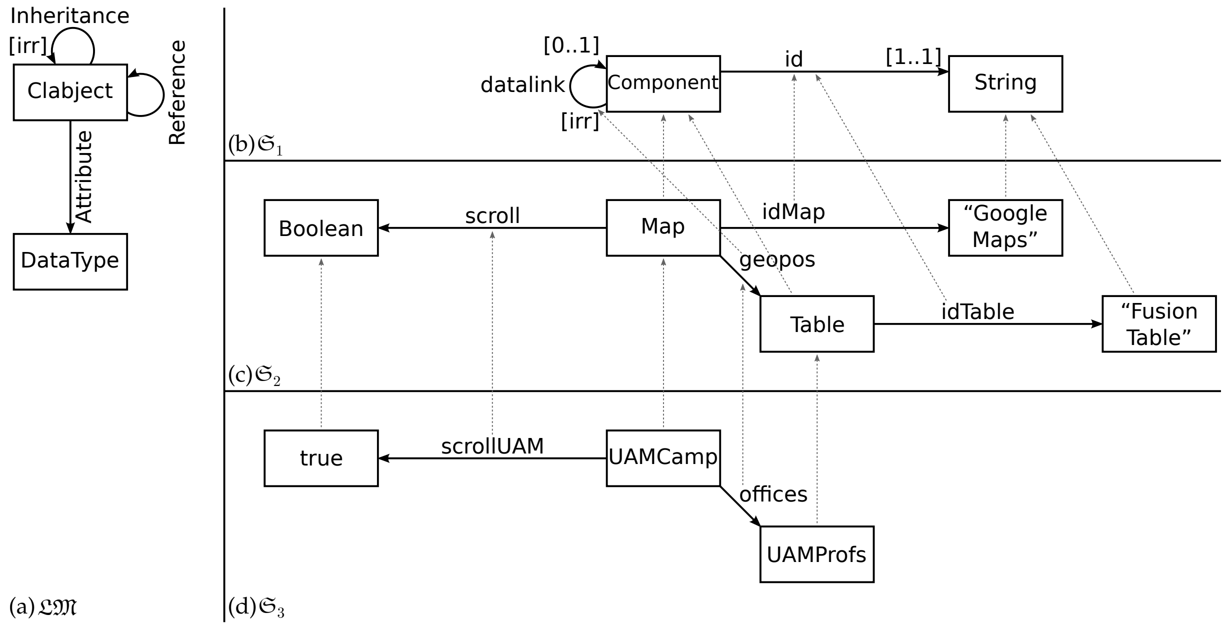


Figure 9. Partial double metamodelling stack for the example, showing the specifications $\mathfrak{LM}$, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ together with the ontological typing morphisms ω_2 and ω_3

Moreover, the subgraphs I_2 and I_3 of the specifications $\mathfrak{S}_2$ and $\mathfrak{S}_3$, respectively, conform ontologically to $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively; i.e., there exist partial two-level ontological typing morphisms $\omega_2 : S_2 \to S_1$ and $\omega_3 : S_3 \to S_2$ such that (I_2, ω_2) and (I_3, ω_3) are valid instances of $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively. Note that in this case, the subgraph I_3 is equal to the underlying graph S_3, meaning that the ontological typing morphism ω_3 is actually total. Compared to Example 6, the ontological typing morphism ω_3 is extended with the following mappings:

$\omega_3(\mathsf{scrollUAM}) = \mathsf{scroll}$

$\omega_3(\mathsf{true}) = \mathsf{Boolean}$

The proposed partial double metamodelling stack adds support for linguistic extension, but still lacks support for deep characterisation.

In the following, we further revise the definition of the partial double metamodelling stack to support different mechanisms of deep characterisation.

5.3. Deep metamodelling stack

A *deep metamodelling stack* is a metamodelling stack which supports double linguistic/ontological conformance, linguistic extension and deep characterisation. Recall that in a deep metamodelling stack, models at each metalevel conform linguistically to the corresponding metamodel of a fixed linguistic modelling language, and a portion of the same models conform ontologically to the models at the metalevels above according to the deep characterisation of elements in these models (see Section 3).

A mechanism for deep characterisation is potency, for which different interpretations are possible. In particular, analysing the existing approaches [dG10, KS07, AK02b] it becomes clear that, implicitly, potency has been given different semantics depending on whether it is attached to clabjects or to attributes. Hence, in this work two kinds of potency are distinguished, namely *multi*-potency ("clabject-like") and *single*-potency ("attribute-like"), denoted by the symbols $\blacktriangle\mathsf{p}$ and $\triangle\mathsf{p}$, respectively. In the following, we define these notions, enabling the attachment of any kind of potency to the different model elements.

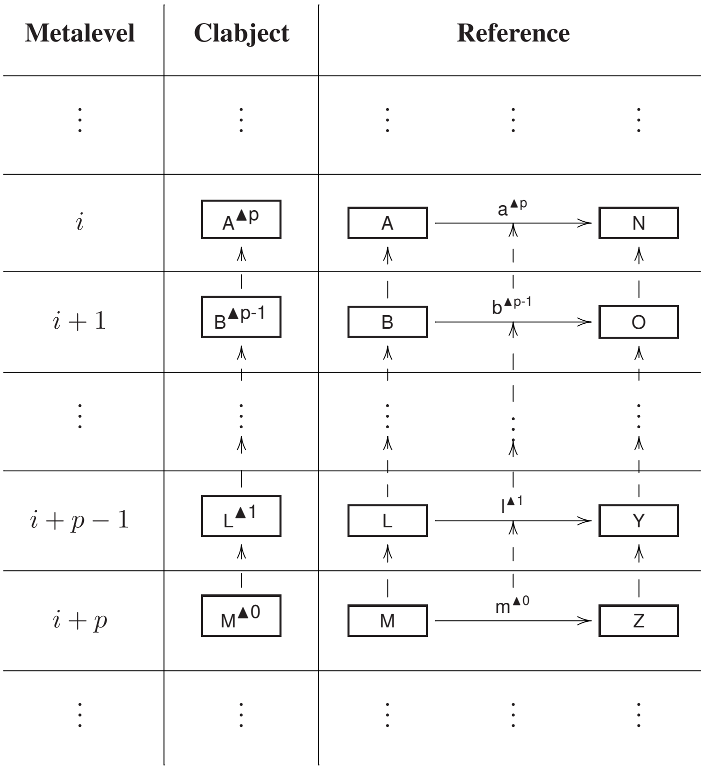


Figure 10. Intuition on the semantics of multi-potency

A multi-potency ▲p on a clabject/reference at metalevel i denotes that this clabject/reference can be instantiated *at all metalevels from* $i + 1$ *to* $i + p$ (see Figure 10), where the instantiation of this clabject/reference has to be *mediated* and the multi-potency has to be *decreased* at each metalevel; e.g., a clabject with multi-potency 0 at metalevel $i + 2$ which is an instance of a clabject with multi-potency 2 at metalevel i must also be an instance of a clabject with multi-potency 1 at metalevel $i + 1$ which in turn is an instance of the considered clabject with multi-potency 2 at metalevel i (see Figures 11 and 12). Most deep metamodelling approaches assume multi-potency semantics for clabjects [ADP09, AM09, AGK09, dG10, KS07]. A multi-potency ▲p on an atomic constraint at metalevel i denotes that this constraint is evaluated at all metalevels from $i + 1$ to $i + p$. Finally, attributes only retain either a type or an instance facet but not both; therefore, the multi-potency on attributes can not be considered.

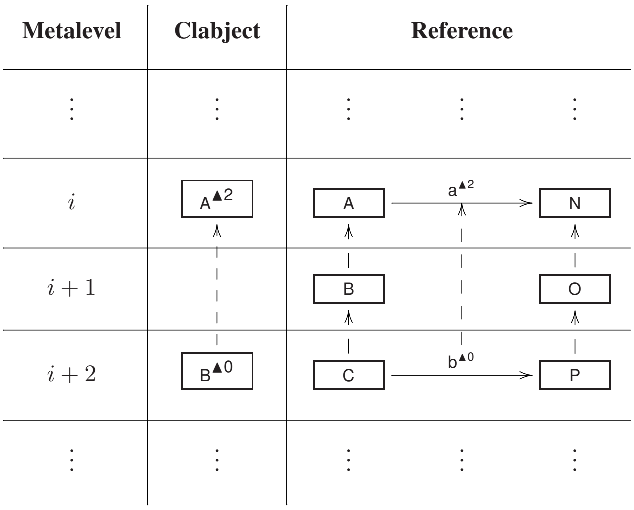


Figure 11. Invalid instantiation: an element with multi-potency 0 at metalevel $i + 2$ can not be a direct instance of an element with multi-potency 2 at metalevel i

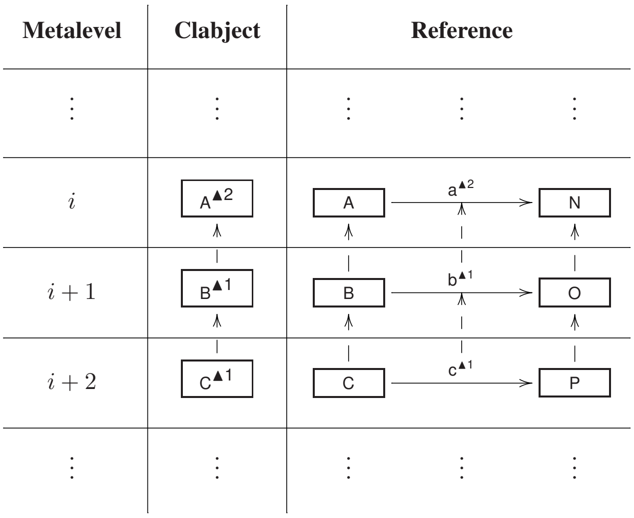


Figure 12. Invalid instantiation: an element with multi-potency 1 at metalevel $i+2$ can not be an instance of an element with the same multi-potency at metalevel $i + 1$

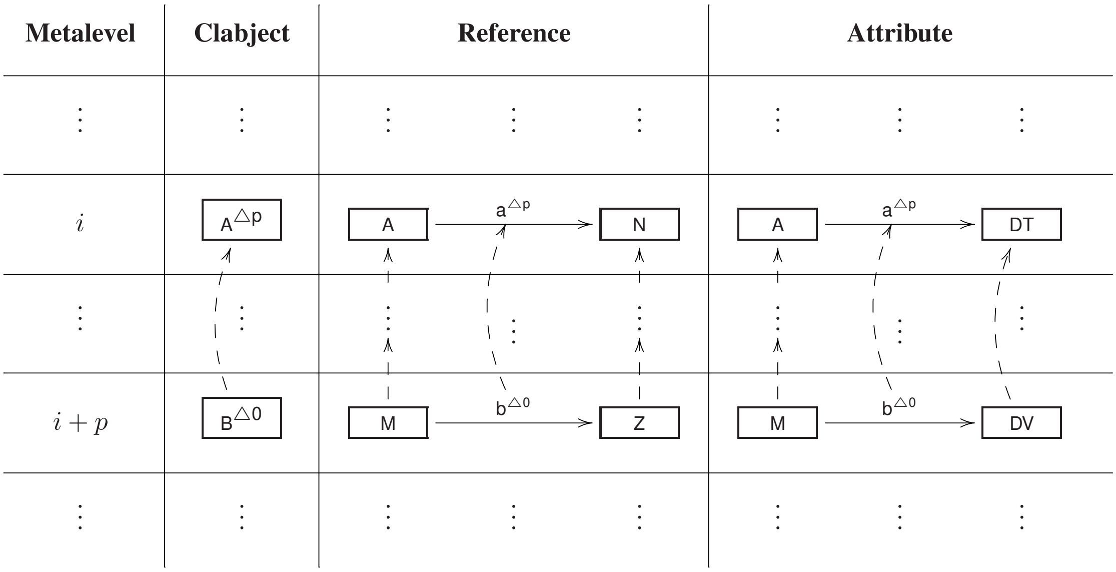


Figure 13. Intuition on the semantics of single-potency

Table 4. Contradictory combinations of multi-potencies on interdependent elements

$\mathsf{A}^{\blacktriangle q} \xrightarrow{\mathsf{a}^{\blacktriangle p}} \mathsf{N}$ $\mathsf{A} \xrightarrow{\mathsf{a}^{\blacktriangle p}} \mathsf{N}^{\blacktriangle q}$	$p > q$: possible to instantiate the reference a only at metalevels $i+1, \ldots, i+q$
$\mathsf{A} \xrightarrow{\mathsf{a}^{\blacktriangle q} \quad \pi^{\blacktriangle p}} \mathsf{N}$	$p > q$: possible to evaluate the atomic constraint π only at metalevels $i + 1, \ldots, i + q$

A single-potency $\triangle\mathsf{p}$ on a clabject/reference at metalevel i, in contrast, denotes that this clabject/reference can be instantiated *at metalevel* $i + p$ *only* (see Figure 13). A single-potency $\triangle\mathsf{p}$ on an attribute at metalevel i denotes that this attribute can be instantiated (i.e., can be assigned a value) at metalevel $i + p$ only. A single-potency $\triangle\mathsf{p}$ on an atomic constraint at metalevel i denotes that this atomic constraint is evaluated at metalevel $i + p$ only.

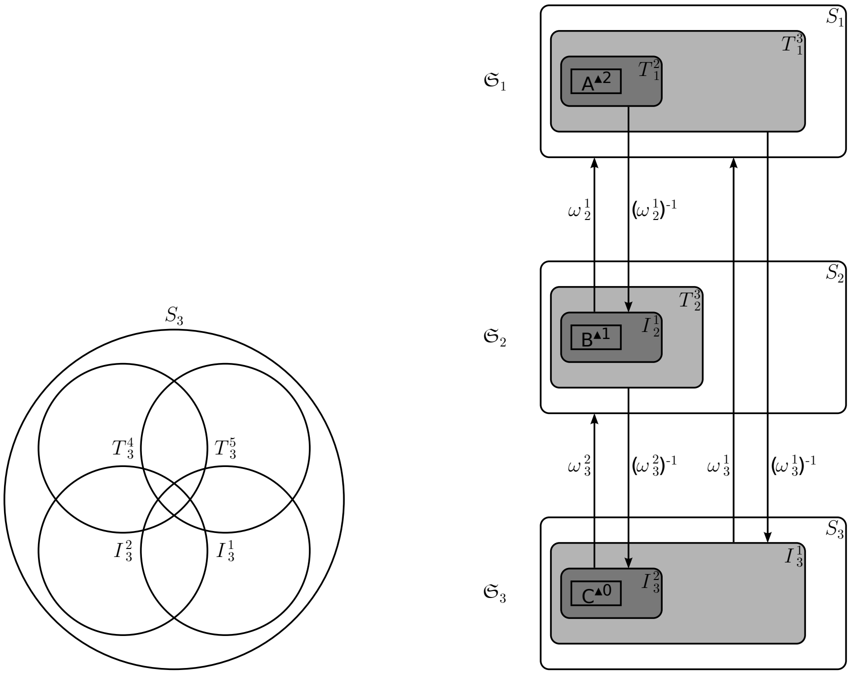
Each element in a model has either a multi-potency or a single-potency. However, some combinations of potencies on interdependent elements may lead to contradictions. Tables 4, 5 and 6 show the contradictory combinations of multi- and single-potencies.

Table 5. Contradictory combinations of single-potencies on interdependent elements

$\boxed{\mathsf{A}^{\triangle q}} \xrightarrow{\mathsf{a}^{\triangle p}} \boxed{\mathsf{N}}$ $\boxed{\mathsf{A}} \xrightarrow{\mathsf{a}^{\triangle p}} \boxed{\mathsf{N}^{\triangle q}}$	$p \neq q$: impossible to instantiate the reference a
$\boxed{\mathsf{A}^{\triangle q}} \xrightarrow{\mathsf{a}^{\triangle p}} \boxed{\mathsf{DT}}$	$p \neq q$: impossible to instantiate the attribute a
$\boxed{\mathsf{A}} \xrightarrow{\mathsf{a}^{\triangle q}\ \pi^{\triangle p}} \boxed{\mathsf{N}}$	$p \neq q$: impossible to evaluate the atomic constraint π

Table 6. Contradictory combinations of multi- and single-potencies on interdependent elements

$\boxed{\mathsf{A}^{\triangle q}} \xrightarrow{\mathsf{a}^{\blacktriangle p}} \boxed{\mathsf{N}}$ $\boxed{\mathsf{A}} \xrightarrow{\mathsf{a}^{\blacktriangle p}} \boxed{\mathsf{N}^{\triangle q}}$	$p \neq q$: impossible to instantiate the reference a $p = q$: possible to instantiate the reference a only if $p = q = 1$
$\boxed{\mathsf{A}^{\blacktriangle q}} \xrightarrow{\mathsf{a}^{\triangle p}} \boxed{\mathsf{N}}$ $\boxed{\mathsf{A}} \xrightarrow{\mathsf{a}^{\triangle p}} \boxed{\mathsf{N}^{\blacktriangle q}}$	$p > q$: impossible to instantiate the reference a
$\boxed{\mathsf{A}^{\blacktriangle q}} \xrightarrow{\mathsf{a}^{\triangle p}} \boxed{\mathsf{DT}}$	$p > q$: impossible to instantiate the attribute a
$\boxed{\mathsf{A}} \xrightarrow{\mathsf{a}^{\triangle q}\ \pi^{\blacktriangle p}} \boxed{\mathsf{N}}$	$p \neq q$: impossible to evaluate the atomic constraint π $p = q$: possible to evaluate the atomic constraint π only if $p = q = 1$
$\boxed{\mathsf{A}} \xrightarrow{\mathsf{a}^{\blacktriangle q}\ \pi^{\triangle p}} \boxed{\mathsf{N}}$	$p > q$: impossible to evaluate the atomic constraint π



(a) A Venn diagram illustrating the partitioning of the underlying graph S_3 of a specification $\mathfrak{S}_3$ into possible type-facet and instance-facet subgraphs

(b) A scheme of a deep metamodelling stack with three metalevels

Figure 14. Illustration of type-facet and instance-facet subgraphs

Next, we provide a structural formalisation of a metamodelling stack with deep characterisation through single- and multi-potency. In analogy to the partial double metamodelling stack, a model at metalevel i of a deep metamodelling stack can be represented in DPF by a specification $\mathfrak{S}_i = (S_i, C_i : \Omega)$ which conforms linguistically to the specification $\mathfrak{LM}$. In contrast to the partial double metamodelling stack, however, the specification $\mathfrak{S}_i$ supports deep characterisation; i.e., it is compliant with the following requirements, for all $1 \leq i < j < k \leq l$, with $o = j - i$ and $p = k - i$:

1. Elements in specifications from $\mathfrak{S}_{i+1}$ to $\mathfrak{S}_k$ can be ontologically typed by elements with multi-potency p in a specification $\mathfrak{S}_i$.
2. Elements in a specification $\mathfrak{S}_k$ can be ontologically typed by elements with single-potency p in a specification $\mathfrak{S}_i$.
3. Elements in specifications from $\mathfrak{S}_{i+1}$ to $\mathfrak{S}_k$ satisfy the atomic constraints with multi-potency p in a specification $\mathfrak{S}_i$.
4. Elements in a specification $\mathfrak{S}_k$ satisfy the atomic constraints with single-potency p in a specification $\mathfrak{S}_i$.

The multi- and single-potency of each clabject, reference and attribute in a specification $\mathfrak{S}_i$ can be represented by considering *type-facet subgraphs* $T_i^k \sqsubseteq S_i$ (see Figure 14). Elements with multi-potency p in a specification $\mathfrak{S}_i$ are included in the type-facet subgraphs from T_i^{i+1} to T_i^k only. Similarly, elements with single-potency p in a specification $\mathfrak{S}_i$ are included in the type-facet subgraph T_i^k only.

Similarly, the multi- and single-potency of each atomic constraint in a specification $\mathfrak{S}_i$ can be represented by considering *subsets of atomic constraints* $C_i^k \sqsubseteq C_i$. Atomic constraints with multi-potency p in a specification $\mathfrak{S}_i$ are included in the subsets from C_i^{i+1} to C_i^k only. Similarly, atomic constraints with single-potency p in a specification $\mathfrak{S}_i$ are included in the subset C_i^k only.

The instantiation in a specification $\mathfrak{S}_k$ of elements with multi- and single-potency p in a specification $\mathfrak{S}_i$ can be represented by considering partial multi-level ontological typing morphisms $\omega_k^i : S_k \multimap S_i$, which are given by *instance-facet subgraphs* $I_k^i \sqsubseteq S_k$ together with total multi-level ontological typing morphisms $\omega_k^i : I_k^i \to S_i$ (see Figures 14(a) and (b)).

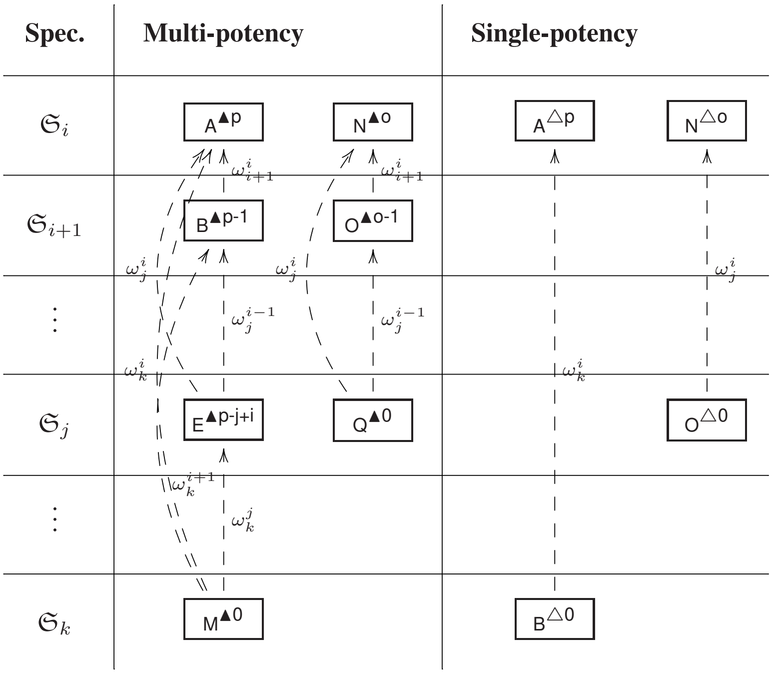


Figure 15. Partial multi-level ontological typing morphisms

The partitioning of a specification into possibly overlapping type-facet subgraphs and instance-facet subgraphs follows the rationale behind the term clabject, namely that elements in a specification of a deep metamodelling stack can retain both a type (class) and instance (object) facet. Figure 14(b) shows a deep metamodelling stack which illustrates this observation. The specification $\mathfrak{S}_1$ contains a single element A with multi-potency 2. The proposed formalisation represents the semantics of the multi-potency 2 on the element A by considering two type-facet subgraphs T_1^2 and T_1^3 and including the element A into them. This reflects the fact that the element A serves as a type for elements in the specifications $\mathfrak{S}_2$ and $\mathfrak{S}_3$. The specification $\mathfrak{S}_2$ contains a single element B with multi-potency 1 which retains both a type and an instance facet. According to the proposed formalisation, the element B is included into the subgraphs T_2^3 and I_2^1. This reflects the fact that the element B serves as a type for elements in the specification $\mathfrak{S}_3$ and, at the same time, it is an instance of the element A in the specification $\mathfrak{S}_1$. Finally, the specification $\mathfrak{S}_3$ contains a single element C with multi-potency 0 which retains an instance facet only. According to the proposed formalisation, the element C is included into the subgraphs I_3^2 and I_3^1 (and no type subgraph T_3^4 is considered). This reflects the fact that the element C is an instance of the element B in the specification $\mathfrak{S}_2$ and, at the same time, an indirect instance of the element A in the specification $\mathfrak{S}_1$. Figure 15 shows another scheme in which the different typings of elements with multi- and single-potency can be observed.

Note that since the instantiation of elements with single-potency can jump over several metalevels, the multi-level ontological typing morphisms ω_k^i and their domains of definition I_k^i can not be obtained by composing the two-level ontological typing morphisms as was the case for partial double metamodelling stacks (see Remark 5); they have to be defined explicitly. Moreover, these jumps mean that the instantiation is no longer monotonic, i.e., $I_k^i \sqsubseteq \ldots \sqsubseteq I_k^{k-1}$ does not hold.

The requirements 1 and 2 that all the elements in a specification $\mathfrak{S}_k$ that are ontologically typed by elements in a specification $\mathfrak{S}_i$ actually have to be ontologically typed by elements in the type-facet subgraph T_i^k can be represented by the condition $(\omega_k^i)^{-1}(T_i^k) = I_k^i$.

The requirements 3 and 4 that all the elements in a specification $\mathfrak{S}_k$ that are ontologically typed by elements in the type-facet subgraph T_i^k also have to satisfy the atomic constraints in the subset C_i^k can be represented by the condition that (I_k^i, ω_k^i) is a valid instance of the *type-facet subspecification* $\mathfrak{T}_i^k = (T_i^k, C_i^k{:}\Omega) \sqsubseteq \mathfrak{S}_i$.

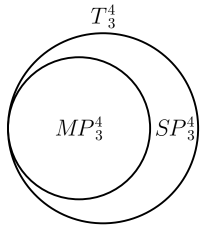


Figure 16. A Venn diagram illustrating the partitioning of the type-facet subgraph T_3^4 of a specification $\mathfrak{S}_3$ into the multi-potency subgraph MP_3^4 and the single-potency subpart SP_3^4, respectively

The partitioning of a specification into type-facet subspecifications ensures that only valid combinations of potencies are allowed. This is because the contradictory combinations of potencies presented in Table 4, 5 and 6 would lead to dangling edges or dangling atomic constraints and hence to invalid type-facet subspecifications.

The requirements above, however, are not sufficient to represent all the aspects of the semantics of deep characterisation. A specification $\mathfrak{S}_i$ of a deep metamodelling stack has to be compliant with the following additional requirements:

5. Elements in specifications from $\mathfrak{S}_{k+1}$ to $\mathfrak{S}_l$ can *not* be ontologically typed by elements with multi-potency p in a specification $\mathfrak{S}_i$; i.e., the instantiation of elements with multi-potencies stops when the multi-potency is zero.
6. Elements in specifications from $\mathfrak{S}_{i+1}$ to $\mathfrak{S}_{k-1}$ and from $\mathfrak{S}_{k+1}$ to $\mathfrak{S}_l$ can *not* be ontologically typed by elements with single-potency p in a specification $\mathfrak{S}_i$.

The multi- and single-potency of each clabject, reference and attribute in a specification $\mathfrak{S}_i$ can be distinguished by considering additional *multi-potency subgraphs* $MP_i^k \sqsubseteq T_i^k$ and *single-potency subparts* $SP_i^k = (T_i^k \setminus MP_i^k) \sqsubseteq T_i^k$ (see Figure 16). These subparts are needed in order to provide a semantics for requirements 5 and 6.

The requirements 5 can be represented by the condition $(\omega_k^i)^{-1}(MP_i^k \setminus MP_i^{k+1}) \sqsubseteq S_k \setminus (\bigcup\limits_{k'>k} T_k^{k'})$ where $S_k \setminus (\bigcup\limits_{k'>k} T_k^{k'})$ includes all the elements in $\mathfrak{S}_k$ which do not retain a type-facet; i.e., which are not instantiated at any metalevel.

The requirement 6 can be represented by the conditions $(\omega_j^i)^{-1}(SP_i^k) = \emptyset$ and $(\omega_k^i)^{-1}(SP_i^k) \sqsubseteq S_k \setminus (\bigcup\limits_{k'>k} T_k^{k'})$.

Furthermore, a specification $\mathfrak{S}_i$ of a deep metamodelling stack has to be compliant with the following additional requirements:

7. Elements in a specification $\mathfrak{S}_k$ which are ontologically typed by elements with multi-potency p in a specification $\mathfrak{S}_i$ must also be ontologically typed by elements with multi-potency $o < p$ in a specification $\mathfrak{S}_j$ which in turn are ontologically typed by the considered elements with multi-potency p in the specification $\mathfrak{S}_i$; i.e., the instantiation of elements with multi-potency is mediated.
8. Elements with multi-potency q in a specification $\mathfrak{S}_k$ can not be ontologically typed by elements with multi-potency $p \leq q$ in a specification $\mathfrak{S}_i$; i.e., the multi-potency of elements is decreased at each instantiation.

The requirement 7 can be represented by the conditions $MP_i^k \sqsubseteq \ldots \sqsubseteq MP_i^{i+1}$, $\omega_k^j; \omega_j^i \sqsubseteq \omega_k^i$ (i.e., $(\omega_k^j)^{-1}(I_j^i) \sqsubseteq I_k^i$) and $(\omega_k^i)^{-1}(MP_i^k) \sqsubseteq I_k^j$.

The requirement 8 can be represented by the condition $(\omega_j^i)^{-1}(MP_i^k \setminus MP_i^{k+1}) \sqsubseteq (MP_j^k \setminus MP_j^{k+1})$.

Finally, a specification $\mathfrak{S}_i$ of a deep metamodelling stack has to be compliant with the following additional requirements:

9. Elements in a specification have either a multi-potency or a single-potency, but not both.
10. The ontological typing is compatible with the linguistic typing.

The requirement 9 can be represented by the condition $SP_i^j \cap T_i^k = \emptyset$.

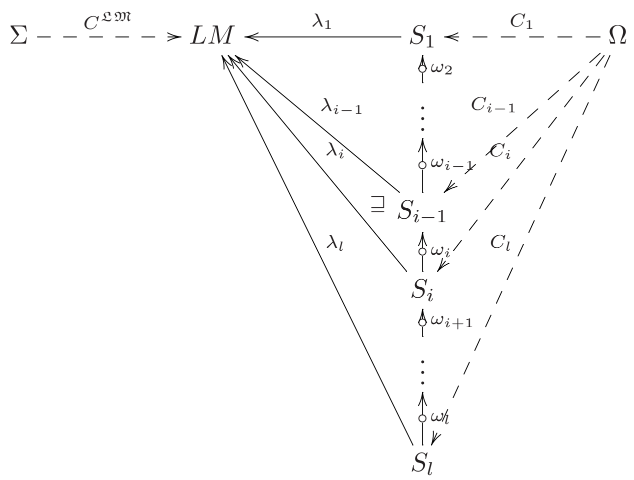
The requirement 10 can be represented as usual by the condition $\omega_k^i; \lambda_i \sqsubseteq \lambda_k$.

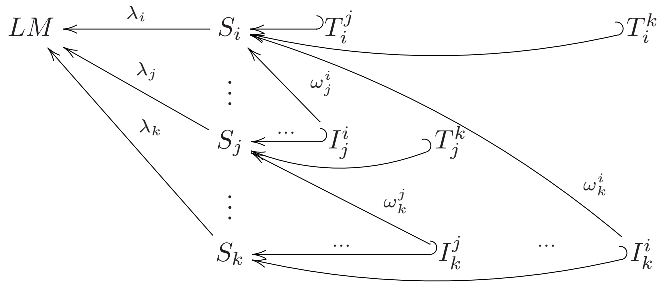
Taking into account all these conditions, the deep metamodelling stack is defined as follows:

Definition 18 (Deep metamodelling stack). A deep metamodelling stack with length l is a linguistic metamodelling stack with length l together with:

- type-facet subspecifications $\mathfrak{T}_i^k = (T_i^k, C_i^k{:}\Omega) \sqsubseteq \mathfrak{S}_i$, for all $1 \leq i < k \leq l$
- multi-potency subgraphs $MP_i^k \sqsubseteq T_i^k$, for all $1 \leq i < k \leq l$, such that:
  - $MP_i^k \sqsubseteq \ldots \sqsubseteq MP_i^{i+1}$ (requirement 7)
- single-potency subparts $SP_i^k = (T_i^k \setminus MP_i^k) \sqsubseteq T_i^k$, for all $1 \leq i < k \leq l$, such that:
  - $SP_i^j \cap T_i^k = \emptyset$, for all $j \neq k$ (requirement 9)
- partial multi-level ontological typing morphisms $\omega_k^i : S_k \multimap S_i$, for all $1 \leq i < k \leq l$, which are given by:
  - instance-facet subgraphs $I_k^i \sqsubseteq S_k$
  - total multi-level ontological typing morphisms $\omega_k^i : I_k^i \to S_i$

  such that for all $1 \leq i < k \leq l$ and all $i < j < k$:
  - $(\omega_k^i)^{-1}(T_i^k) = I_k^i$ (requirements 1 and 2)
  - $(I_k^i, \omega_k^i) \in \textbf{Inst}(\mathfrak{T}_i^k)$ (requirements 3 and 4)
  - $(\omega_k^i)^{-1}(MP_i^k \setminus MP_i^{k+1}) \sqsubseteq S_k \setminus (\bigcup_{k'>k} T_k^{k'})$ (requirement 5)
  - $(\omega_j^i)^{-1}(SP_i^k) = \emptyset$ (requirement 6)
  - $(\omega_k^i)^{-1}(SP_i^k) \sqsubseteq S_k \setminus (\bigcup_{k'>k} T_k^{k'})$ (requirement 6)
  - $\omega_k^j; \omega_j^i \sqsubseteq \omega_k^i$ (i.e., $(\omega_k^j)^{-1}(I_j^i) \sqsubseteq I_k^i$)) (requirement 7)
  - $(\omega_k^i)^{-1}(MP_i^k) \sqsubseteq I_k^j$ (requirement 7)
  - $(\omega_j^i)^{-1}(MP_i^k \setminus MP_i^{k+1}) \sqsubseteq (MP_j^k \setminus MP_j^{k+1})$ (requirement 8)
  - $\omega_k^i; \lambda_i \sqsubseteq \lambda_k$ (requirement 10)





Example 8 (Deep metamodelling stack). Building upon Example 7, Figure 17(a) shows the specification $\mathfrak{LM}$ and Figures 17(b), (c) and (d) show the specifications $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$. Moreover, Figure 17 shows the ontological typing morphisms ω_2^1 and ω_3^2 as dashed grey arrows. Figure 18 shows the same specifications and the ontological typing morphism ω_3^1.

In analogy to Example 7, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ conform linguistically to $\mathfrak{LM}$.

In contrast to Example 7, however, the multi-potency ▲2 on the clabject Component and the reference datalink denotes that these elements are in both type-facet subgraphs T_1^2 and T_1^3 (as well as the multi-potency subgraphs MP_1^2 and MP_1^2). Moreover, the single-potency △1 on the attribute id denotes that this element is in the type-facet subgraph T_1^2 only (as well as the single-potency subpart SP_1^2), while the single-potency △1 on the atomic constraint $(\texttt{[mult(1,1)]}, \delta_3)$ on the same attribute denotes that this element is in the subset of atomic constraints C_1^2 only. Furthermore, the single-potency △2 on the attribute name denotes that this element is in the type-facet subgraph T_1^3 only (as well as the single-potency subpart SP_1^3), while the single-potency △2 on the atomic constraint $(\texttt{[mult(1,1)]}, \delta_4)$ on the same attribute denotes that this element is in the subset of atomic constraints C_1^3 only.

The specification $\mathfrak{S}_2$ conforms ontologically to $\mathfrak{S}_1$; i.e., there exists a partial multi-level ontological typing morphism $\omega_2^1 : S_2 \multimap S_1$ such that (I_2^1, ω_2^1) is a valid instance of the type-facet subspecification $\mathfrak{T}_1^2 = (T_1^2, C_1^2{:}\Omega)$. The ontological typing morphism ω_2^1 is defined as follows (see Figure 17):

$\omega_2^1(\mathsf{Map}) = \omega_2^1(\mathsf{Table}) = \mathsf{Component}$
$\omega_2^1(\mathsf{geopos}) = \mathsf{datalink}$
$\omega_2^1(\mathsf{idMap}) = \omega_2^1(\mathsf{idTable}) = \mathsf{id}$
$\omega_2^1(\text{“GoogleMaps”}) = \omega_2^1(\text{“FusionTable”}) = \mathsf{String}$

The specification $\mathfrak{S}_3$ conforms ontologically to both $\mathfrak{S}_2$ and $\mathfrak{S}_1$; i.e., there exists partial multi-level ontological typing morphisms $\omega_3^2 : S_3 \multimap S_2$ and $\omega_3^1 : S_3 \multimap S_1$ such that (I_3^2, ω_3^2) and (I_3^1, ω_3^1) are valid instances of the type-facet subspecifications $\mathfrak{T}_2^3 = (T_2^3, C_2^3{:}\Omega)$ and $\mathfrak{T}_1^3 = (T_1^3, C_1^3{:}\Omega)$, respectively. The ontological typing morphisms ω_3^2 and ω_3^1 are defined as follows (see Figures 17 and 18):

$\omega_3^2(\mathsf{UAMCamp}) = \mathsf{Map}$
$\omega_3^2(\mathsf{UAMProfs}) = \mathsf{Table}$
$\omega_3^2(\mathsf{offices}) = \mathsf{geopos}$
$\omega_3^2(\mathsf{scrollUAM}) = \mathsf{scroll}$
$\omega_3^2(\mathsf{true}) = \mathsf{Boolean}$
$\omega_3^1(\mathsf{UAMCamp}) = \omega_3^1(\mathsf{UAMProfs}) = \mathsf{Component}$
$\omega_3^1(\mathsf{offices}) = \mathsf{datalink}$
$\omega_3^1(\mathsf{nameMapUAM}) = \omega_3^1(\mathsf{nameTableUAM}) = \mathsf{name}$
$\omega_3^1(\text{“UAMCampus”}) = \omega_3^1(\text{“UAMProfs”}) = \mathsf{String}$

It is straightforward to show that this sample deep metamodelling stack satisfies all the conditions in Definition 18.

In this section, we presented a formalisation of deep metamodelling based on DPF from a structural point of view.

In the following, we switch to an operational point of view and show how to flatten deep characterisation by transforming a deep metamodelling stack into a partial double metamodelling stack.

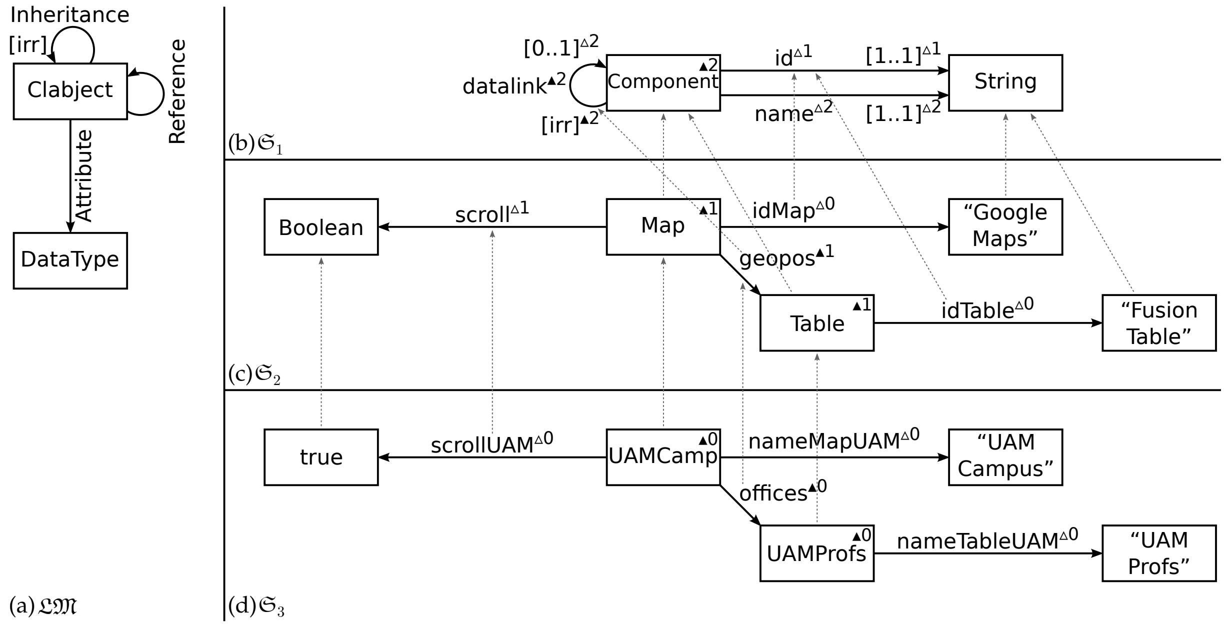


Figure 17. The specifications $\mathfrak{LM}$, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ together with the ontological typing morphisms ω_2^1 and ω_3^2

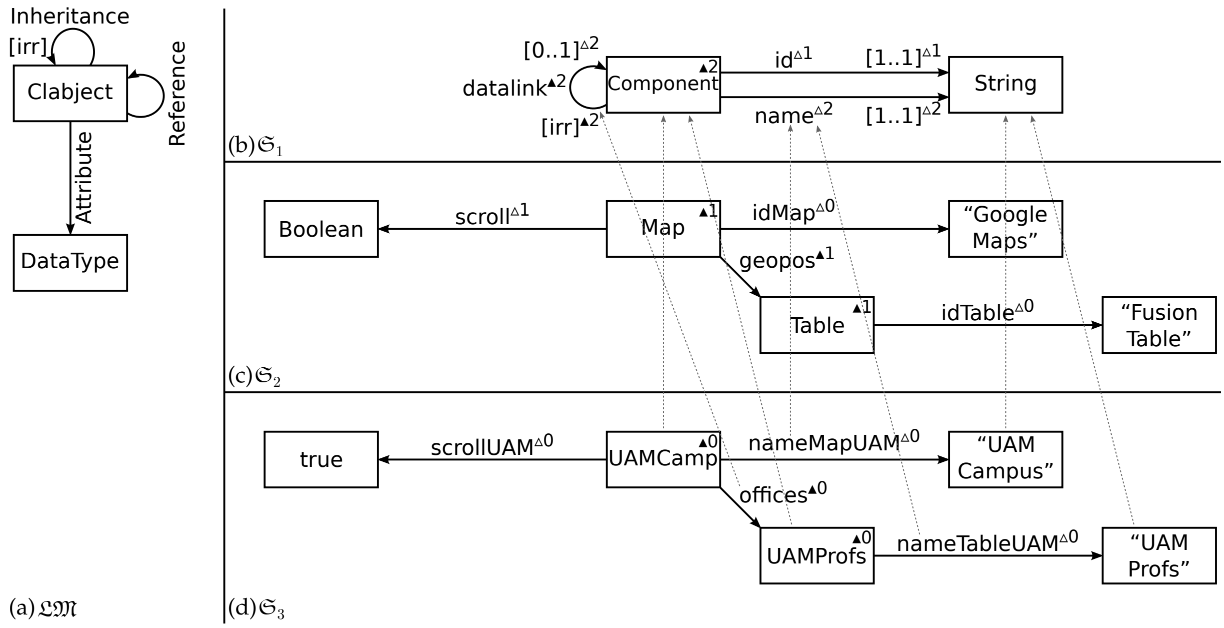


Figure 18. The specifications $\mathfrak{LM}$, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ together with the ontological typing morphism ω_3^1

6. Flattening of a deep metamodelling stack

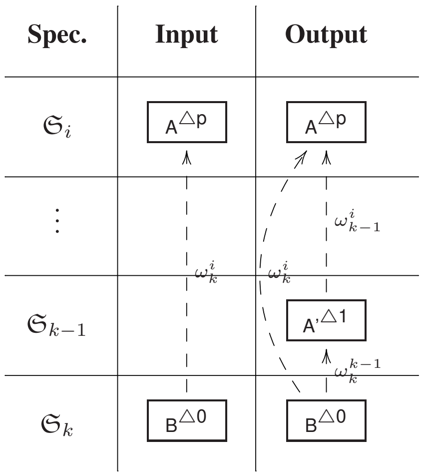
Recall that in a deep metamodelling stack, an element with single-potency 0 at metalevel k may be ontologically typed by an element with single-potency $p = k - i$ at metalevel i; i.e., there may be p metalevels between an instance and its type. In a double metamodelling stack, in contrast, an element at metalevel k can only be ontologically typed by an element at metalevel $k - 1$. In order to better illustrate the semantics of deep characterisation, we show how to flatten deep characterisation by transforming a deep metamodelling stack into a partial double metamodelling stack. This flattening is defined by multiple *replication rules* and an *extraction rule*.

The replication rules rc_0, rr_1, ra_1 and rac_2 follow a general pattern which, for each element with single-potency $p \geq 2$ at metalevel i, adds to metalevel $k - 1$ a replica of the considered element with single-potency decreased to 1. Similar to the layering of transformation rules in specification transformation [EEPT06], the subscripts from 0 to 2 denote the layer to which a rule belongs, so that rules of layer 0 are applied before rules of layer 1, etc.

The replication rule rc_0 adds to metalevel $k - 1$ a replica with single-potency 1 of a clabject with single-potency p at metalevel i, as follows[1]:

Definition 19 (Replication rule rc_0 for clabjects). Given a deep metamodelling stack with length l, for all $1 \leq i < k \leq l$ and $k \geq i + 2$:

- for each $\mathsf{A} \in SP_i^k$
  - $T'^k_{k-1} = T^k_{k-1} \cup \mathsf{A'}$
  - $I'^i_{k-1} = I^i_{k-1} \cup \mathsf{A'}$ and $\omega^i_{k-1}(\mathsf{A'}) = \mathsf{A}$
- for each $\mathsf{B} \in I^i_k$ such that $\omega^i_k(\mathsf{B}) = \mathsf{A}$
  - $I'^{k-1}_k = I^{k-1}_k \cup \mathsf{B}$ and $\omega^{k-1}_k(\mathsf{B}) = \mathsf{A'}$

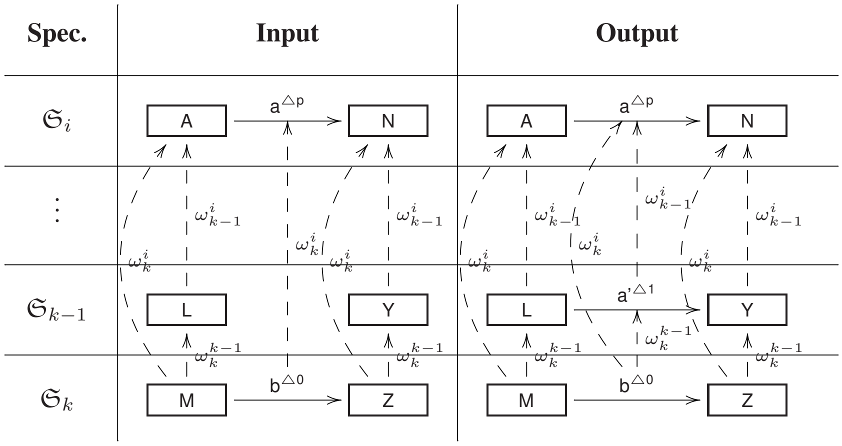


The replication rule rr_1 adds to metalevel $k - 1$ a replica with single-potency 1 of a reference with single-potency p at metalevel i, as follows:

[1] T'^k_{k-1} and I'^i_{k-1} denote the state of the type- and instance-facet subgraphs T^k_{k-1} and I^i_{k-1}, respectively, after the application of the rule.

Definition 20 (Replication rule rr_1 for references). Given a deep metamodelling stack with length l, for all $1 \leq i < k \leq l$ and $k \geq i+2$:

- for each $(\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{N}) \in SP_i^k$
  - for each $\mathsf{L}, \mathsf{Y} \in I_{k-1}^i$ such that $\omega_{k-1}^i(\mathsf{L}) = \mathsf{A}$ and $\omega_{k-1}^i(\mathsf{Y}) = \mathsf{N}$
    - $T'^k_{k-1} = T^k_{k-1} \cup (\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{Y})$
    - $I'^i_{k-1} = I^i_{k-1} \cup (\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{Y})$ and $\omega_{k-1}^i(\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{Y}) = (\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{N})$
- for each $(\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{Z}) \in I_k^i$ such that $\omega_k^i(\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{Z}) = (\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{N})$, $\omega_k^{k-1}(\mathsf{M}) = \mathsf{L}$ and $\omega_k^{k-1}(\mathsf{Z}) = \mathsf{Y}$
  - $I'^{k-1}_k = I^{k-1}_k \cup (\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{Z})$ and $\omega_k^{k-1}(\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{Z}) = (\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{Y})$

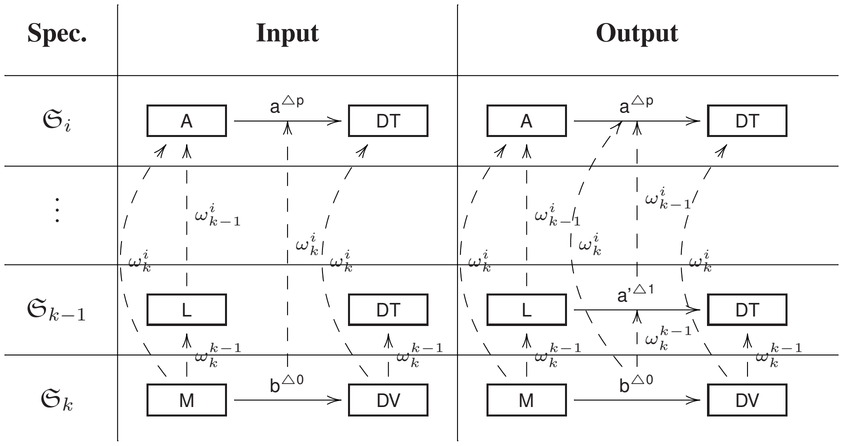


Remark 6 (Identity of data types). Recall that, similar to E-graphs [EPT04, EEPT06], attributes of nodes can be represented in DPF by edges from these nodes to nodes representing data types. Nodes representing data types can be regarded as having a "global identity" in a deep metamodelling stack. Therefore, we assume that all nodes representing data types are implicitly available in each specification $\mathfrak{S}_i$ of the deep metamodelling stack.

The replication rule ra_1 adds to metalevel $k-1$ a replica with single-potency 1 of an attribute with single-potency p at metalevel i, as follows:

Definition 21 (Replication rule ra_1 for attributes). Given a deep metamodelling stack with length l, for all $1 \leq i < k \leq l$ and $k \geq i+2$:

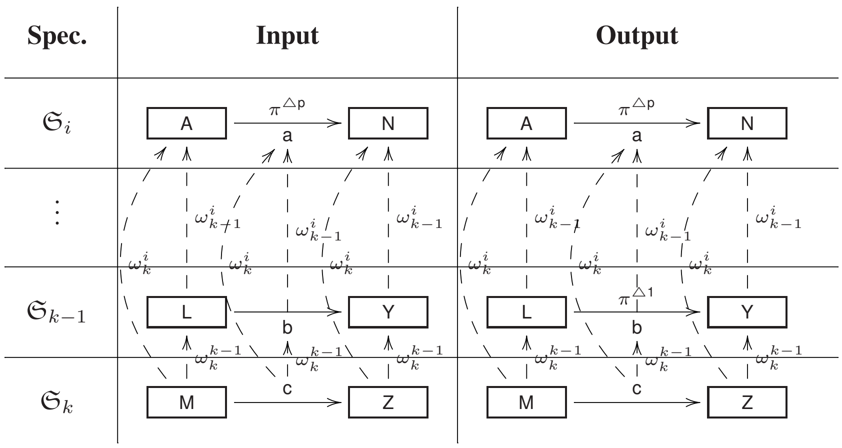
- for each $(\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{DT}) \in SP_i^k$
  - for each $\mathsf{L}, \mathsf{DT} \in I_{k-1}^i$ such that $\omega_{k-1}^i(\mathsf{L}) = \mathsf{A}$
    - $T'^k_{k-1} = T^k_{k-1} \cup (\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{DT})$
    - $I'^i_{k-1} = I^i_{k-1} \cup (\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{DT})$ and $\omega_{k-1}^i(\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{DT}) = (\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{DT})$
- for each $(\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{DV}) \in I_k^i$ such that $\omega_k^i(\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{DV}) = (\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{DT})$, $\omega_k^{k-1}(\mathsf{M}) = \mathsf{L}$ and $\omega_k^{k-1}(\mathsf{DV}) = \mathsf{DT}$
  - $I'^{k-1}_k = I^{k-1}_k \cup (\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{DV})$ and $\omega_k^{k-1}(\mathsf{M} \xrightarrow{\mathsf{b}} \mathsf{DV}) = (\mathsf{L} \xrightarrow{\mathsf{a'}} \mathsf{DT})$



The replication rule rac_2 adds to metalevel $k-1$ a replica with single-potency 1 of an atomic constraint with single-potency p at metalevel i, as follows:

Definition 22 (Replication rule rac_2 for atomic constraints). Given a deep metamodelling stack with length l, for all $1 \leq i < k \leq l$ and $k \geq i+2$:

- for each $(\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{N}) \in T_i^k$ and $(\pi, \delta) \in C_i^k$ where $\delta(\alpha^\Omega(\pi)) = (\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{N})$
  - for each $(\mathsf{L} \xrightarrow{\mathsf{b}} \mathsf{Y}) \in (T_{k-1}^k)$ such that $\omega_{k-1}^i(\mathsf{L} \xrightarrow{\mathsf{b}} \mathsf{Y}) = (\mathsf{A} \xrightarrow{\mathsf{a}} \mathsf{N})$
    - $C'^k_{k-1} = C^k_{k-1} \cup (\pi, \delta')$ where $\delta'(\alpha^\Omega(\pi)) = (\mathsf{L} \xrightarrow{\mathsf{b}} \mathsf{Y})$



Note that the rule rac_2 for the replication of atomic constraints is proposed as a proof-of-concept only. This is because this rule is designed to work with the predicates having arities $1 \circlearrowleft^{a}$ and $1 \xrightarrow{a} 2$, e.g., `[irreflexive]` and `[mult(`m, n`)]` from the signature Ω (see Table 3). However, predicates may have arbitrary arities and semantics which may not enable replication of atomic constraints at all. The conditions under which a predicate enables replication of atomic constraints is outside the scope of this work and will be investigated in future work (see Section 9).

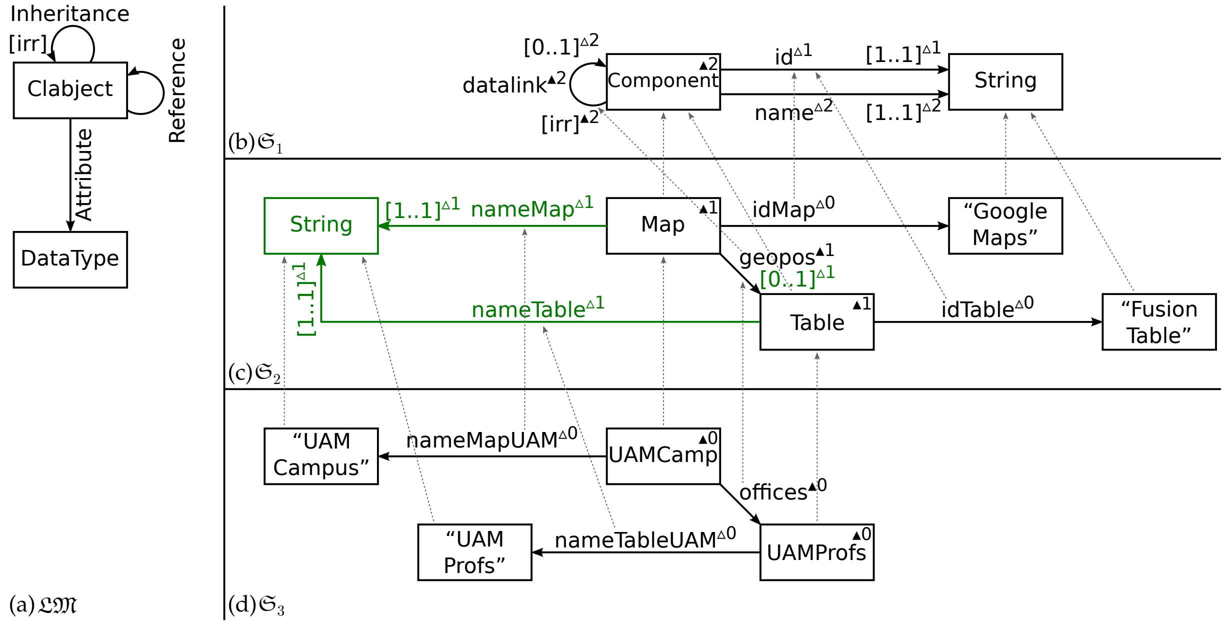


Figure 19. The specifications $\mathfrak{LM}$, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ together with the ontological typing morphisms ω_2^1 and ω_3^2, after the application of the replication rules

According to this layering, the application of the rules adds a replica of a reference only *after* it adds a replica of a clabject. This ensures that the rule which adds a replica of a reference matches both clabjects with multi-potency and their instances as well as clabjects with single-potency and their replicas. Moreover, this ensures that the replica of the reference has as source and target an instance of the considered clabject with multi-potency or a replica of the considered clabject with single-potency. The layering of rules for attributes and atomic constraints follow the same rationale.

The extraction rule e_3 projects out the types at each metalevel i and the corresponding instances at metalevel $i+1$ as the elements in each specification of the target partial double metamodelling stack, as follows:

Definition 23 (Extraction rule e_3). Given a deep metamodelling stack with length l, a double metamodelling stack with length l is extracted as follows:

- $\mathfrak{S}_1 = (T_1^2, C_1^2, \lambda_1)$
- for all $2 \le i \le l-1$, $\mathfrak{S}_i = (T_i^{i+1} \cup I_i^{i-1}, C_i^{i+1}, \lambda_i, \omega_i^{i-1})$
- $\mathfrak{S}_l = (I_l^{l-1}, \lambda_l, \omega_l^{l-1})$

Example 9 (Flattening of a deep metamodelling stack). Building upon Example 8, Figures 19(b), (c) and (d) show the specifications $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ of the deep metamodelling stack, after the application of the replication rules. Moreover, Figure 19(c) shows the replicated elements in green colour. Note that the attribute scroll, the data type Boolean and the corresponding instances are omitted from Figure 19 due to space constraints.

Firstly, the application of ra_1 adds to $\mathfrak{T}_2^3$ the attributes nameMap and nameTable with single-potency $\triangle 1$. Moreover, it adds the following mappings to the ontological typing morphism ω_3^2:

$\omega_3^2(\text{nameMapUAM}) = \text{nameMap}$
$\omega_3^2(\text{nameTableUAM}) = \text{nameTable}$
$\omega_3^2(\text{"UAMCampus"}) = \omega_3^2(\text{"UAMProfs"}) = \text{String}$

Secondly, the application of rac_2 adds to $\mathfrak{T}_2^3$ the atomic constraints $(\texttt{[mult (0,1)]}, \delta_1)$, $(\texttt{[mult (1,1)]}, \delta_2)$ and $(\texttt{[mult (1,1)]}, \delta_3)$ with single-potency $\triangle 1$ on the reference geopos and the attributes nameMap and nameTable, respectively.

Figures 21(b), (c) and (d) show the specifications $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ of the partial double metamodelling stack resulting from the application of the extraction rule. Moreover, Figure 20(b) shows the discarded elements in red colour.

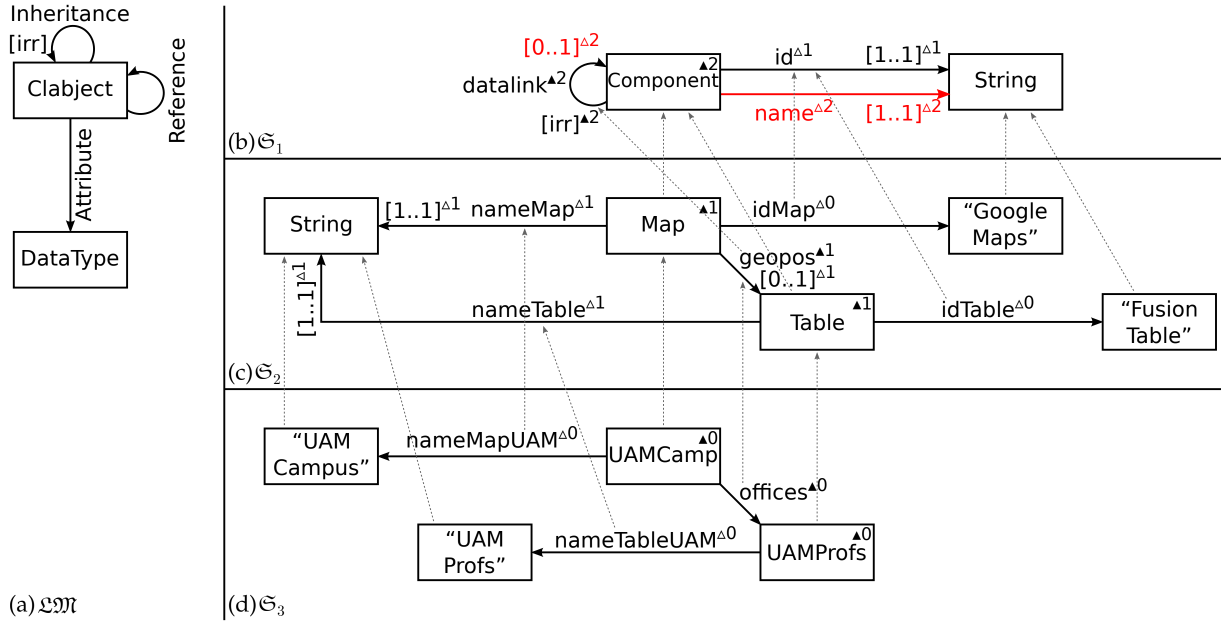


Figure 20. The specifications $\mathfrak{LM}$, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ together with the ontological typing morphisms ω_2^1 and ω_3^2, before the application of the extraction rule

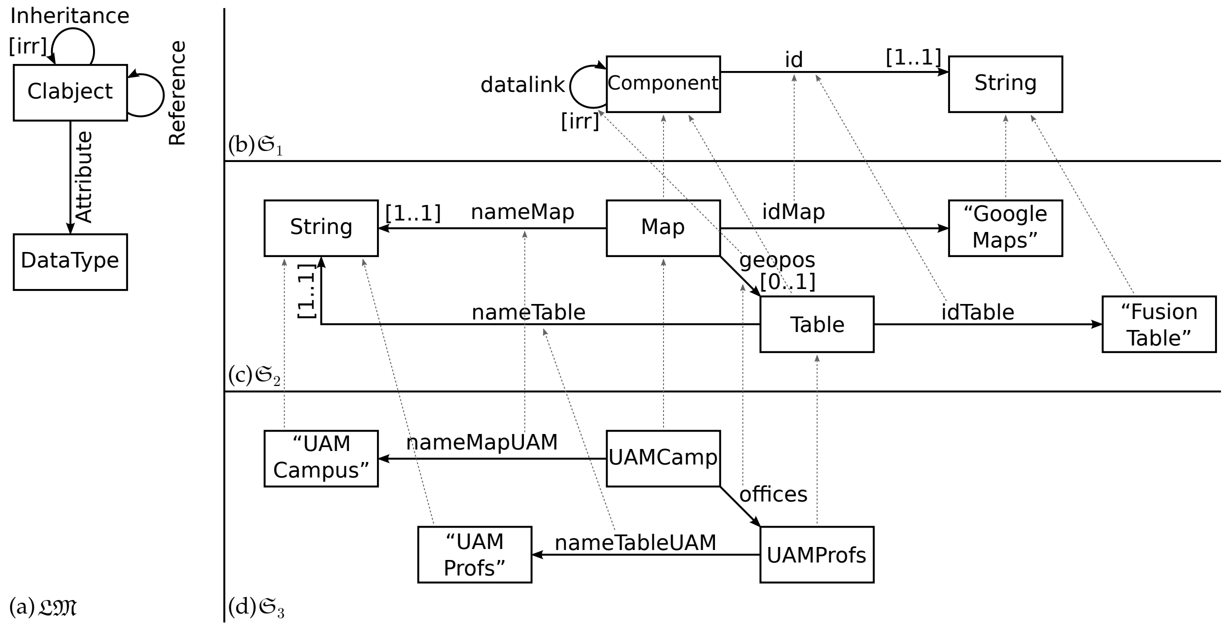


Figure 21. The specifications $\mathfrak{LM}$, $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ together with the ontological typing morphisms ω_2 and ω_3, after the application of the extraction rule

The application of e_3 discards from $\mathfrak{S}_1$ the atomic constraints $([\texttt{mult}(0,1)], \delta_1)$ and $([\texttt{mult}(1,1)], \delta_4)$ on datalink and name, respectively. In this way, these atomic constraints are not evaluated at metalevel 2. Moreover, it discards from $\mathfrak{S}_1$ the attribute name. In this way, it is not possible to instantiate name at metalevel 2.

The presented flattening of the deep characterisation enables the transformation of a deep metamodelling stack into a partial double metamodelling stack. Obviously, part of the deep characterisation information is lost in the transformation. For instance, in Example 8, the multi-potency ▲2 on the elements Component and datalink in $\mathfrak{S}_1$ forbids that these elements are ontologically typed by elements in a possible specification $\mathfrak{S}_4$ or below. In Example 9, in contrast, a possible specification $\mathfrak{S}_4$ may include elements which are ontologically typed by elements in $\mathfrak{S}_3$.

In addition to the flattening of the deep characterisation, it is possible to define the flattening of the double linguistic/ontological conformance which enables the transformation of a partial double metamodelling stack into a traditional metamodelling stack. This could be done by adding the specification $\mathfrak{LM}$ on top of the ontological stack, and adding a replica of all elements in $\mathfrak{LM}$ in all the specifications $\mathfrak{S}_i$, for all $i \leq l - 2$.

7. From theory to practice

METADEPTH [dG10] is a multi-level metamodelling tool that supports deep characterisation through potency, and a textual syntax for modelling. The tool integrates languages for model manipulation and code generation, which makes it suitable for MDE. Listing 1 shows the encoding of the example of Figure 6 using METADEPTH's textual syntax.

```
Model ComponentView@2 {
 Node Component {
  ident@1 : String;
  name : String {id};
  visualise : boolean;
  src : Component[*];
  trg : Component[*];

  irreflexive : $self.trg.excludes(self)$
 }
 Edge Datalink (Component.src, Component.trg){}
}

ComponentView RepositoryComponents {
 Component Map {
  ident = "GoogleMaps";
  srcTable : Table[0..1]{src};
  scroll : boolean = false;
 }
 Component Table {
  ident = "FusionTable";
  trgMap : Map[*]{trg};
 }
 Datalink Geopos(Map.srcTable, Table.trgMap){}
}

RepositoryComponents myApplication {
 Map UAMCamp {
  name = "UAMCampus";
  visualise = true;
  scroll = true;
 }
 Table UAMProfs {
  name = "UAMProfs";
  visualise = false;
 }
 Geopos offices(UAMCamp, UAMProfs){}
}
```

Listing 1: Definition of the language for components using METADEPTH.

Line `1` defines a model named `ComponentView` having potency 2 (specified after the `@` symbol). All elements (clabjects, edges) inside the model receive this potency if it is not explicitly overriden. This can be considered a short-cut to avoid specifying this potency in every element. Clabjects are declared with the keyword `Node`, as shown in line `2`. Note that, being `ComponentView` a top-level model, it is not ontologically typed. All attributes inside `Component` have potency 2, except `ident`, which has potency 1.

Attributes have a name and a type, and optionally a potency (to override the one received from their container clabjects), a multiplicity (one is assumed if none is specified), an initial value and a modifier. The latter are predefined constraints, like `id` (makes the attribute value unique among all objects of same type in the model), `ordered` (makes the collection a sequence) and `unique` (forbids repeated elements in a collection).

Constraints can be specified using an OCL dialect permitting side effects called Epsilon Object Language (EOL) [KPP06]. Line `9` shows a constraint, which receives potency 2 from its container. Constraints can be attached to nodes, edges and models. Attaching a constraint to an element can be done in two ways: by its definition in the context of the element (as shown in the listing), or by declaring it in an outer context and explicitly attaching it to a node or edge. The latter enables the definition of libraries of constraints, following similar ideas to the presented DPF formalisation. As a difference with the formalisation, we cannot explicitly define the arity of the constraint, which is always fixed (node or edge), but we permit arbitrary navigation from the element the constraint is attached to.

Edges are declared as shown in line `11`. They model bidirectional associations by making two already declared references one the opposite of the other. In the listing, references `src` and `trg` are made opposites. Just like nodes, edges can declare attributes as well.

Lines `14-25` show the instantiation of the `ComponentView` model. The instantiated model has potency 1, is named `RepositoryComponents` and includes two instances of `Component`. The first one (`Map`) includes a linguistic extension in line `18`, a `scroll` attribute with type `boolean` and initial value `true`. Moreover, references have multi-potency semantics, and hence we explicitly instantiate references `src` and `trg` of `Component` as shown in lines `17` and `22`. The reference `srcTable` is an instance of `src` (shown inside the brackets), declares `Table` as the type of the reference end, and a multiplicity of `[0..1]`.

Finally, the `ComponentView` model can be instantiated as shown in lines `27-38`. All elements in that model have potency 0. In this case, the edge `offices` does not need to detail the names of the references it connects, as it has only instance facet and this information was given in its type.

While the default semantics of potency for nodes, edges and references is multi-potency, the one for constraints and primitive attributes is single-potency. Following the presented DPF formalisation, METADEPTH was enhanced to support both multi- and single-potency for both nodes and edges. Single-potency is depicted by placing the potency value between parenthesis. For example, we can modify the models as shown in Listing 2 to incorporate single potency to the edge and the `src` and `trg` references. This has the effect that these elements can only be instantiated two metalevels below, hence `Datalink` instances can connect any (indirect) instance of `Component` with potency 0.

```
Model ComponentView@2 {
 Node Component {
   ...
   src@(2) : Component[*];
   trg@(2) : Component[*];
 }
 Edge Datalink@(2) (Component.src, Component.trg){}
}
```

Listing 2: Adding single-potency to some model elements.

Hence, altogether, the presented DPF formalisation helped in realising the two possible semantics for potency, as well as the provision of rules to detect their contradictory combination (see Tables 4, 5 and 6). The latter were implemented as well formedness constraints. As a difference with the formalisation, the tool does not support potencies on multiplicities yet, but assumes a potency of 1 for them.

8. Related work

In this section we compare our work with other deep metamodelling frameworks, as well as with other formal approaches to metamodelling.

8.1. Deep metamodelling frameworks

Deep metamodelling is a relatively new technique, and some of its aspects are still debated in the literature. A first strand of research focuses on multi-level metamodelling.

Early forms of multi-level metamodelling can be traced back to knowledge-based systems like Telos [MBJK90] and deductive object base managers like ConceptBase [JGJS95].

More recent forms include the works in [GOS07, AM09, CSW08]. In [GOS07], MOF is extended with multiple metalevels to enable XML-based code generation. Nivel [AM09] is a double metamodelling framework based on the weighted constraint rule language (WCRL). XMF [CSW08] is a language-driven development framework allowing an arbitrary number of metalevels. The cross-layer modeller (XLM) [DLHE11] allows multilevel modelling of arbitrary numbers of metalevels, by using an embedding in UML and modelling instantiation semantics as OCL constraints. In particular, the designer needs to specify templatised OCL constraints to control the instantiation of associations.

Another form of multi-level metamodelling can be achieved through powertypes [Ode94, GPHS06], since instances of powertypes are also subtypes of another type and hence retain both a type and an instance facet. Multi-level metamodelling can also be emulated through stereotypes [Obj10b], although this is not a general modelling technique since it relies on UML to emulate the extension of its metamodel. The interested reader can consult [AK08] for a thorough comparison of potencies, powertypes and stereotypes.

In contrast to our approach, none of the above mentioned works support deep characterisation; i.e., the ability to describe structure and express constraints for metalevels below the adjacent one. Moreover, none of them enable the definition of linguistic extensions, which are useful in the definition of complex deep languages.

A second strand of research focuses on deep characterisation. Deep characterisation through potency is included in the works [KS07, GKA08, ADP09, AGK09, dG10, AGK12]. DeepJava [KS07] is a superset of Java which extends the object-oriented programming paradigm to feature an unbounded number of metalevels. The work in [GKA08] describes the problems arising from the way in which connectors (e.g., associations, links, generalisations, etc.) are supported in mainstream modelling languages such as UML and why they are not suitable for deep metamodelling. The work in [AGK09] presents a prototype implementation of a modelling infrastructure which provides built-in support for multiple ontological as well as linguistic metalevels. This work was continued within the Melanie tool [AGK12], which includes support for suggesting emendations for models at lower metalevels when models at upper metalevels of a metamodelling stack change (e.g., by changing the value of a potency). In Melanie, fields can be decorated with so-called traits, like value *mutability*, which defines over how many metalevels the value may be changed from the default. In addition, model elements should also be decorated with a *level*, which specifies the metalevel at which the owning model resides. In our formalisation, this is not necessary as the metalevel is implicit in each specification, so that all its elements have the same metalevel. The work in [ADP09] proposes a deep metamodelling framework which extends the basic notion of clabject for handling connector inheritance and instantiation. METADEPTH [dG10, dGCML13] is a deep metamodelling framework which supports potency, double linguistic/ontological typing and linguistic extension.

While these works agree on that clabjects are instantiated using the multi-potency semantics, they differ in other design decisions. Firstly, some works are ambiguous about the instantiation semantics for associations. In [KS07], the associations can be represented as Java references; hence we interpret that they are instantiated using the single-potency semantics. In [GKA08], the connectors are explicitly represented as clabjects but their instantiation semantics is not discussed; hence we interpret that they are instantiated using the multi-potency semantics. Secondly, not all works adhere to *strict* ontological metamodelling. In [ADP09], the ontological type of an association does not need to be in the adjacent metalevel above, but several metalevels above. Note that our single-potency semantics makes it possible to retain strict metamodelling for associations through a flattening construction that replicates these associations. Finally, some works differ in how they tackle potency on constraints and methods. Potency on constraints is not explicitly shown in [AGK09] and not considered in [ADP09], whereas potency on methods is only supported by DeepJava and METADEPTH.

Table 7 shows a summary of the particular semantics for deep characterisation implemented by the above mentioned works and compares it with the semantics supported by our formalisation. It is worth noting that no tool recognises the fact that multiplicity constraints are constraints as well and hence can have a potency.

8.2. Formal approaches to metamodelling

The formalisation of diagrammatic modelling has been extensively discussed in the literature.

The work in [EPT04, EEPT06] uses E-graphs to represent models and metamodels. An E-graph is a generalisation of an attributed graph [EEKR99] and consists of two sets of graph and data nodes, respectively, and three sets of graph

Table 7. Comparison of different deep characterisation semantics

	Clabjects	Associations	Strictness	Constraints	Mult. constraints
DeepJava [KS07]	▲	△	yes	△	N.A.
Atkinson et al. [AGK09, AGK12]	▲	▲	yes	▲	▲1
Aschauer et al. [ADP09]	▲	▲	no	N.A.	▲1
METADEPTH [dG10, dGCML13]	△, ▲	△, ▲	yes	△	▲1
DPF formalisation	△, ▲	△, ▲	yes	△, ▲	△, ▲

edges, node attribute edges and edge attribute edges, respectively. The assignment of attributes to nodes is done by adding node attribute edges from the graph nodes to the data nodes. The assignment of attributes to edges is done by adding edge attribute edges from the graph edges to the data nodes. Attributes of nodes and edges are used to describe properties of nodes and edges, which is similar to how attributes of classes in the UML metamodel are used to describe properties of model elements. Attributes of nodes can be represented in DPF by edges from these nodes to nodes representing data types. The adoption of E-graphs rather than directed multi-graphs may represent a natural next step in the development of DPF.

The work in [BM09] proposes an algebraic semantics for MOF to formalise the concepts of models, metamodels and conformance between them. Models are represented by terms while metamodels are represented by specifications in membership equational logic (MEL). This formal semantics is made executable by using Maude [CDE$^+$07], which directly supports MEL specifications.

The work in [Poe06] exploits the higher-order nature of constructive type theory to uniformly treat the syntax of models, metamodels, as well as MOF itself. Models are represented by terms (token models) and can also be represented by types (type models) by means of a reflection mechanism. This formal semantics ensures that correct typing corresponds to provably correct models and metamodels.

9. Conclusion and future work

In this paper, we presented a formal approach to deep metamodelling based on DPF. Firstly, we illustrated the limitations of traditional metamodelling through an example in the domain of component-based web applications. Secondly, we introduced deep metamodelling through the same example. Thirdly, we defined double linguistic/ontological typing and linguistic extension in view of DPF. Fourthly, we formalised deep characterisation and defined two different semantics for potency, namely multi- and single-potency. Fifthly, we showed how to flatten deep characterisation by transforming a deep metamodelling stack into a double metamodelling stack. Finally, we discussed how the findings of the proposed formalisation are ported back to the METADEPTH deep metamodelling tool.

This paper further develops the formalisation of deep metamodelling published in [RdG$^+$12]. Compared to the previous work, we extended the introduction with a presentation of linguistic extension. Moreover, we provided a declarative semantics of deep metamodelling (i.e., deep characterisation through potency, double linguistic/ontological typing and linguistic extension). Finally, we discussed an implementation of the proposed formalisation within the METADEPTH [dG10] tool.

To the best of our knowledge, this work is the first attempt to clarify and formalise some aspects of the semantics of deep metamodelling. In particular, this work explains different semantic variation points available for deep metamodelling, points out new possible semantics, currently unexplored in practice, as well as classifies the existing tools according to these options.

In future work, we will investigate the effects of overriding the potency of a clabject using inheritance, as this may lead to additional contradictory combinations of potencies. On the practical side, we will define further constructions to flatten multiple metalevels into two and to eliminate the double typing. The implementation of such flattenings in METADEPTH will allow the migration of deep metamodelling stacks into two-metalevel frameworks like EMF.

Acknowledgements

This work was partially funded by the Spanish Ministry of Economy and Competitiveness (project "Go Lite" TIN2011-24139).

A. Appendix

Definition 24 (Partial map). A partial map $f : A \multimap B$ between two sets A and B is given by the domain of definition $dom(f) \subseteq A$ and a total map $f : dom(f) \to B$. For any subset $A_0 \subseteq A$, the image of the subset A_0 under f is defined as $f(A_0) = \{f(a) \mid a \in A_0 \text{ and } a \in dom(f)\} \subseteq f(A) \subseteq B$. For any subset $B_0 \subseteq B$, the inverse image of the subset B_0 under f is defined as $f^{-1}(B_0) = \{a \in dom(f) \mid f(a) \in B_0\} \subseteq f^{-1}(B) \subseteq A$. The composition of two partial maps $f : A \multimap B$ and $g : B \multimap C$ is defined by $dom(f;g) = f^{-1}(dom(g)) \subseteq dom(f)$ and $(f;g)(a) = g(f(a))$, for all $a \in dom(f;g)$.

It is straightforward to check that: the composition of partial maps is associative; for any subset $C_0 \subseteq C$ we have $(f;g)^{-1}(C_0) = g^{-1}(f^{-1}(C_0))$; for any subset $B_0 \subseteq B$ we have $f(f^{-1}(B_0)) \subseteq B_0$ and hence $f(dom(f;g)) \subseteq dom(g)$.

Definition 25 (Partial order over partial maps). A partial order $\sqsubseteq$ over the set of all partial maps from the set A to the set B can be defined as: given two partial maps $f, g : A \multimap B$, $f \sqsubseteq g$ if and only if $dom(f) \subseteq dom(g)$ and $f(a) = g(a)$, for all $a \in dom(f)$.

References

[ADP09] Thomas Aschauer, Gerd Dauenhauer, and Wolfgang Pree. Multi-level Modeling for Industrial Automation Systems. In *Proceedings of EUROMICRO 2009: 35th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 490–496. IEEE Computer Society, 2009.

[AGK09] Colin Atkinson, Matthias Gutheil, and Bastian Kennel. A Flexible Infrastructure for Multilevel Language Engineering. *IEEE Transactions on Software Engineering*, 35(6):742–755, 2009.

[AGK12] Colin Atkinson, Ralph Gerbig, and Bastian Kennel. On-the-Fly Emendation of Multi-level Models. In Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle, and Dimitrios S. Kolovos, editors, *Proceedings of ECMFA 2012: 8th European Conference on Modelling Foundations and Applications*, volume 7349 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2012.

[AK02a] Colin Atkinson and Thomas Kühne. Profiles in a strict metamodeling framework. *Science of Computer Programming*, 44(1):5–22, 2002.

[AK02b] Colin Atkinson and Thomas Kühne. Rearchitecting the UML infrastructure. *ACM Transactions on Modeling and Computer Simulation*, 12(4):290–321, 2002.

[AK08] Colin Atkinson and Thomas Kühne. Reducing accidental complexity in domain models. *Software and Systems Modeling*, 7(3):345–359, 2008.

[AM09] Timo Asikainen and Tomi Männistö. Nivel: a metamodelling language with a formal semantics. *Software and Systems Modeling*, 8(4):521–549, 2009.

[BG01] Jean Bézivin and Olivier Gerbé. Towards a Precise Definition of the OMG/MDA Framework. In *Proceedings of ASE 2001: 16th IEEE International Conference on Automated Software Engineering*, pages 273–280, 2001.

[BM09] Artur Boronat and José Meseguer. Algebraic Semantics of OCL-Constrained Metamodel Specifications. In Manuel Oriol, Bertrand Meyer, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Proceedings of TOOLS 2009: 47th International Conference on Objects, Components, Models and Patterns*, volume 33 of *Lecture Notes in Business Information Processing*, pages 96–115. Springer, 2009.

[BW95] Michael Barr and Charles Wells. *Category Theory for Computing Science (2nd Edition)*. Prentice Hall, 1995.

[CDE+07] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[CSW08] Tony Clark, Paul Sammut, and James Willans. *Applied Metamodelling: A Foundation for Language Driven Development (2nd Edition)*. Ceteva, 2008.

[dG10] Juan de Lara and Esther Guerra. Deep Meta-modelling with METADEPTH. In Jan Vitek, editor, *Proceedings of TOOLS 2010: 48th International Conference on Objects, Components, Models and Patterns*, volume 6141 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2010.

[dGCML13] Juan de Lara, Esther Guerra, Ruth Cobos, and Jaime Moreno-Llorena. Extending Deep Meta-Modelling for Practical Model-Driven Engineering. *The Computer Journal*, In press, 2013.

[Dis96] Zinovy Diskin. Databases as Diagram Algebras: Specifying Queries and Views Via the Graph-Based Logic of Sketches. Technical Report 9602, Frame Inform Systems/LDBD, Riga, Latvia, 1996.

[Dis97] Zinovy Diskin. Towards algebraic graph-based model theory for computer science. In *Logic Colloquium 1995: European Summer Meeting of the Association for Symbolic Logic*. *Bulletin of Symbolic Logic*, 3(1):144–145, 1997.

[Dis02] Zinovy Diskin. Visualization vs. Specification in Diagrammatic Notations: A Case Study with the UML. In Mary Hegarty, Bertrand Meyer, and N. Hari Narayanan, editors, *Proceedings of Diagrams 2002: 2^{nd} International Conference on Diagrammatic Representation and Inference*, volume 2317 of *Lecture Notes in Computer Science*, pages 112–115. Springer, 2002.

[Dis03] Zinovy Diskin. *Practical foundations of business system specifications*, chapter Mathematics of UML: Making the Odysseys of UML less dramatic, pages 145–178. Springer, 2003.

[Dis05] Zinovy Diskin. *Encyclopedia of Database Technologies and Applications*, chapter Mathematics of Generic Specifications for Model Management I and II, pages 351–366. Information Science Reference, 2005.

[DK03] Zinovy Diskin and Boris Kadish. Variable set semantics for keyed generalized sketches: formal semantics for object identity and abstract syntax for conceptual modeling. *Data & Knowledge Engineering*, 47(1):1–59, 2003.

[DK05] Zinovy Diskin and Boris Kadish. *Encyclopedia of Database Technologies and Applications*, chapter Generic Model Management, pages 258–265. Information Science Reference, 2005.

[DKPJ00] Zinovy Diskin, Boris Kadish, Frank Piessens, and Michael Johnson. Universal Arrow Foundations for Visual Modeling. In Michael Anderson, Peter Cheng, and Volker Haarslev, editors, *Proceedings of Diagrams 2000: 1^{st} International Conference on Diagrammatic Representation and Inference*, volume 1889 of *Lecture Notes in Computer Science*, pages 345–360. Springer, 2000.

[DLHE11] Andreas Demuth, Roberto E. Lopez-Herrejon, and Alexander Egyed. Cross-layer modeler: a tool for flexible multilevel modeling with consistency checking. In Tibor Gyimóthy and Andreas Zeller, editors, *Proceedings of SIGSOFT/FSE 2011: 19^{th} ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 452–455. ACM, 2011.

[DW08] Zinovy Diskin and Uwe Wolter. A Diagrammatic Logic for Object-Oriented Visual Modeling. In *Proceedings of ACCAT 2007: 2^{nd} Workshop on Applied and Computational Category Theory*, volume 203/6 of *Electronic Notes in Theoretical Computer Science*, pages 19–41. Elsevier, 2008.

[Ecl] Eclipse Modeling Framework. *Project Web Site*. `http://www.eclipse.org/emf/`.

[EEKR99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages, and Tools*. World Scientific Publishing Company, 1999.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, March 2006.

[EPT04] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental Theory for Typed Attributed Graph Transformation. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Proceedings of ICGT 2004: 2^{nd} International Conference on Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 161–177. Springer, 2004.

[Fia04] José Luiz Fiadeiro. *Categories for Software Engineering*. Springer, May 2004.

[GKA08] Matthias Gutheil, Bastian Kennel, and Colin Atkinson. A Systematic Approach to Connectors in a Multi-level Modeling Environment. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *Proceedings of MoDELS 2008: 11^{th} International Conference on Model Driven Engineering Languages and Systems*, volume 5301 of *Lecture Notes in Computer Science*, pages 843–857. Springer, 2008.

[GOS07] Ralf Gitzel, Ingo Ott, and Martin Schader. Ontological Extension to the MOF Metamodel as a Basis for Code Generation. *Computer Journal*, 50(1):93–115, 2007.

[GPHS06] Cesar Gonzalez-Perez and Brian Henderson-Sellers. A powertype-based metamodelling framework. *Software and Systems Modeling*, 5(1):72–90, 2006.

[JGJS95] Matthias Jarke, Rainer Gallersdörfer, Manfred A. Jeusfeld, and Martin Staudt. ConceptBase - A deductive object base for meta data management. *Journal of Intelligent Information Systems*, 4(2):167–192, 1995.

[KPP06] Dimitrios S. Kolovos, Richard F. Paige, and Fiona Polack. The Epsilon Object Language (EOL). In Arend Rensink and Jos Warmer, editors, *Proceedings of ECMDA-FA 2006: 2^{nd} European Conference on Model-Driven Architecture Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2006.

[KS07] Thomas Kühne and Daniel Schreiber. Can Programming be Liberated from the Two-Level Style? Multi-Level Programming with DeepJava. In Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes, and Guy L. Steele Jr., editors, *Proceedings of OOPSLA 2007: 22^{nd} Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications*, pages 229–244. ACM, 2007.

[MBJK90] John Mylopoulos, Alexander Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, 8(4):325–362, 1990.

[Obj] Object Management Group. *Web site*. `http://www.omg.org`.

[Obj06] Object Management Group. *Meta-Object Facility Specification*, January 2006. `http://www.omg.org/spec/MOF/2.0/`.

[Obj10a] Object Management Group. *Object Constraint Language Specification*, February 2010. `http://www.omg.org/spec/OCL/2.2/`.

[Obj10b] Object Management Group. *Unified Modeling Language Specification*, May 2010. `http://www.omg.org/spec/UML/2.3/`.

[Ode94] James Odell. Power Types. *Journal of Object-Oriented Programming*, 7(2):8–12, 1994.

[Poe06] Iman Poernomo. A Type Theoretic Framework for Formal Metamodelling. In *International Seminar on Architecting Systems with Trustworthy Components*, volume 3938 of *Lecture Notes in Computer Science*, pages 262–298. Springer, 2006.

[RdG^{+}12] Alessandro Rossini, Juan de Lara, Esther Guerra, Adrian Rutle, and Yngve Lamo. A Graph Transformation-Based Semantics for Deep Metamodelling. In Andy Schürr, Dániel Varró, and Gergely Varró, editors, *Proceedings of AGTIVE 2011: 4^{th} International Symposium on Applications of Graph Transformations with Industrial Relevance*, volume 7233 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2012.

[Ros11] Alessandro Rossini. *Diagram Predicate Framework meets Model Versioning and Deep Metamodelling*. PhD thesis, Department of Informatics, University of Bergen, Norway, December 2011.

[RRLW09a] Adrian Rutle, Alessandro Rossini, Yngve Lamo, and Uwe Wolter. A Category-Theoretical Approach to the Formalisation of Version Control in MDE. In Marsha Chechik and Martin Wirsing, editors, *Proceedings of FASE 2009: 12^{th} International Conference on Fundamental Approaches to Software Engineering*, volume 5503 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2009.

[RRLW09b] Adrian Rutle, Alessandro Rossini, Yngve Lamo, and Uwe Wolter. A Diagrammatic Formalisation of MOF-Based Modelling Languages. In Manuel Oriol, Bertrand Meyer, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Proceedings of TOOLS 2009: 47th International Conference on Objects, Components, Models and Patterns*, volume 33 of *Lecture Notes in Business Information Processing*, pages 37–56. Springer, 2009.

[RRLW10a] Alessandro Rossini, Adrian Rutle, Yngve Lamo, and Uwe Wolter. A formalisation of the copy-modify-merge approach to version control in MDE. *Journal of Logic and Algebraic Programming*, 79(7):636–658, 2010.

[RRLW10b] Adrian Rutle, Alessandro Rossini, Yngve Lamo, and Uwe Wolter. A Formalisation of Constraint-Aware Model Transformations. In David Rosenblum and Gabriele Taentzer, editors, *Proceedings of FASE 2010: 13th International Conference on Fundamental Approaches to Software Engineering*, volume 6013 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2010.

[RRLW12] Adrian Rutle, Alessandro Rossini, Yngve Lamo, and Uwe Wolter. A formal approach to the specification and transformation of constraints in MDE. *Journal of Logic and Algebraic Programming*, 81(4):422–457, 2012.

[RRM$^+$11] Alessandro Rossini, Adrian Rutle, Khalid A. Mughal, Yngve Lamo, and Uwe Wolter. A Formal Approach to Data Validation Constraints in MDE. In Marcel Kyas, Sun Meng, and Volker Stolz, editors, *Proceedings of TTSS 2011: 5th International Workshop on Harnessing Theories for Tool Support in Software*, pages 65–76, September 2011.

[Rut10] Adrian Rutle. *Diagram Predicate Framework: A Formal Approach to MDE*. PhD thesis, Department of Informatics, University of Bergen, Norway, November 2010.

[SBPM08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0 (2nd Edition)*. Addison-Wesley Professional, 2008.