Abstract vs Concrete Clabjects in Dual Deep Instantiation

Bernd Neumayr and Michael Schrefl

Department of Business Informatics – Data & Knowledge Engineering
Johannes Kepler University Linz, Austria
firstname.lastname@jku.at

Abstract. Deep Instantiation allows for a compact representation of models with multiple instantiation levels where clabjects combine object and class facets and allow to characterize the schema of model elements several instantiation levels below. Clabjects with common properties may be generalized to superclabjects. In order to clarify the exact nature of superclabjects, Dual Deep Instantiation, a variation of Deep Instantiation, distinguishes between abstract and concrete clabjects and demands that superclabjects are abstract. An abstract clabject combines the notion of abstract class, i.e., it may not be instantiated by concrete objects, and of abstract object, i.e., is does not represent a single concrete object but properties common to a set of concrete objects. This paper clarifies the distinction between abstract and concrete clabjects and discusses the role of concrete clabjects for mandatory constraints at multiple levels and for coping with dual inheritance introduced with the combination of generalization and deep instantiation. The reflections in this paper are formalized based on a simplified form of dual deep instantiation but should be relevant to deep characterization in general.

1 Introduction

What is represented as instance data in one application may be represented as schema data in another application. For example, in an application for managing a product catalog, product category *car* is represented by a class Car which is instantiated by objects representing particular car models, such as BMW Z4. In a customer service application, the same car model may be represented by a class which is instantiated by objects representing individual cars, such as PetersZ4.

Potency-based *Deep Instantiation* [1] allows for a compact and integrated representation of such scenarios. For example, clabject BMW Z4 in our running example (see Fig. 1), which makes use of a simplified and relaxed version of Dual Deep Instantiation [9], represents both: an object, namely the car model *BMW Z4*, and a class, namely the class of individual cars of model *BMW Z4*. Further up in the product hierarchy, the clabject Car with potency 2 represents product category *car* as well as the classes of individual cars and of car models. Finally, the whole product hierarchy is represented by clabject Product with potency 3.

Clabjects with common properties may be generalized to a superclabject. In Dual Deep Instantiation, superclabjects are abstract and are not considered as objects in their own right. For example, concrete clabjects Car and Motorcycle are generalized to an abstract superclabject Vehicle which defines properties shared by Car and Motorcycle, e.g., MsBlack is the category manager of the two product categories represented by Car and Motorcycle.

In the remainder of the paper we introduce, in Sect. 2, a simplified form of Dual Deep Instantiation along which we discuss, in Sect. 3, the distinction between abstract and concrete clabjects. In Sect. 4, we define an inheritance mechanism and describe the role of concrete clabjects in dealing with dual inheritance stemming from the combination of generalization and deep instantiation. Sect. 5 introduces support for defining mandatory constraints over concrete clabjects at multiple levels in order to control the stepwise instantiation process. Sect. 6 gives a brief overview of related work and Sect. 7 concludes the paper.

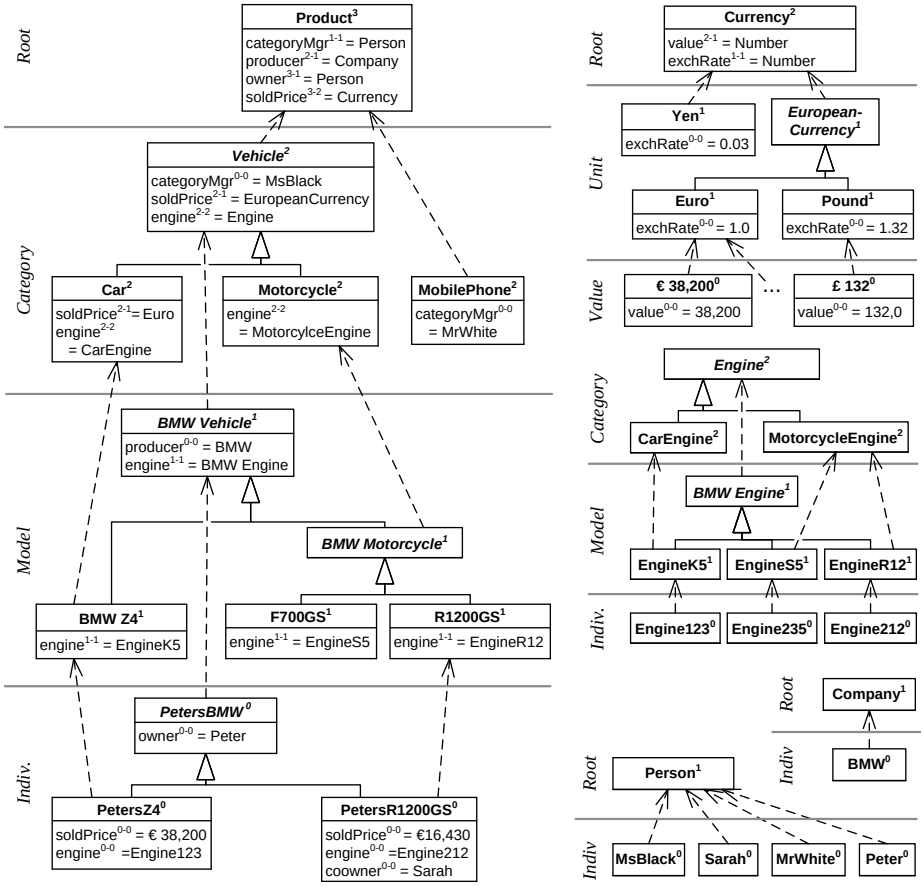

Fig. 1. Multi-level Hierarchies of Abstract and Concrete Clabjects

2 Dual Deep Instantiation and Generalization – Simplified

Dual Deep Instantiation (DDI) [9] allows to relate clabjects at different instantiation levels by *bi-directional and multi-valued relationships* and allows to separately indicate the depth of characterization for the source and target of a relationship (not assuming a strict separation of classification levels). By restricting clabjects to have only one parent, either related by instantiation or by generalization, it only allows *single inheritance*. For discussing the distinction between concrete and abstract clabjects and their role in multi-level models we introduce, in this section, a simplified and relaxed variant of DDI. It is *simpflified by only considering uni-directional and single-valued property references*. It is *relaxed by allowing dual inheritance*: a clabject may now have two parents, one at the same level and connected by a generalization relationship and the other at the next higher level and related by an instantiation relationship.

A DDI model contains a set of clabjects (see No. 1 in Table 2). Clabjects are organized in instantiation hierarchies with an arbitrary number of instantiation levels, where $in(x, y)$ expresses that clabject x is an instantiation of clabject y (No. 2); we also say y is the *class-clabject* of x. Clabjects at the same instantiation level may be organized in specialization hierarchies, where $spec(x, y)$ expresses that clabject x is a *direct specialization* of clabject y (No. 3); we also refer to y as *the superclabject* of x.

We use the term *clabject* in a wide sense. It covers what is traditionally modeled as individual objects (tokens, instance specifications), classes, simple values, and primitive datatypes. In DDI, even individuals may refine or extend their own schema, e.g., property coowner is introduced at individual PetersR1200GS, and we assume, in this regard, that every individual comes with its own class facet. So, in DDI *everything is a clabject*. Note, in the original DDI approach [9] we used the terms *object* or *DDI object* for what we now call *clabject*.

Each clabject comes with a clabject potency (No. 4). A potency of a clabject is given by a natural number (including 0) and indicates the number of instantiation levels of the clabject, where $ptcy(x) = n$ expresses that x has descendants at the next n instantiation levels beneath. Note, in our previous work [9], clabjects did not have an asserted potency.

In order to simplify discussion and formalization of the approach, we introduce auxiliary terms, predicates and shorthand notations. We say x *isa* y if x is either a specialization or an instantiation of y (No. 5), we also say y is a parent of x. We say x is a *member* of y if x relates to y by a chain of *isa* with exactly one instantiation step (No. 6). We say x is an n-member of y if x relates to y by a chain of *isa* with n instantiation steps (No. 7). We use $.^+$ and $.^*$ to denote the transitive and transitive-reflexive closure, respectively, of a binary relation. We say, x is a *descendant* of y and y is an *ancestor* of x if $isa^+(x, y)$. We say, x is a *specialization* of y and y is a *generalization* (or is *a superclabject*) of x if $spec^+(x, y)$.

When a clabject x instantiates a clabject c then the potency of x is the potency of c decremented by one (No. 8). Clabjects in generalization hierarchies all have the same clabject potency (No. 9).

Table 1. Instantiation and Generalization Hierarchies of Clabjects

Sorts & Asserted Predicates:	
(1)	C: *clabjects* (representing individuals, classes, datatypes and values)
(2)	$in \subseteq C \times C$
(3)	$spec \subseteq C \times C$
(4)	$ptcy : C \to \mathbb{N}$ ($\mathbb{N}$ is the set of natural numbers including 0)
Auxiliary Predicates:	
(5)	$isa(x, y) :\Leftrightarrow in(x, y) \vee spec(x, y)$
(6)	$member(x, c) :\Leftrightarrow \exists s \exists d : spec^*(x, s) \wedge in(s, d) \wedge spec^*(d, c)$
(7)	$nmember(x, c, n) :\Leftrightarrow (n = 0 \wedge spec^*(x, c)) \vee$ $\exists m \exists d : ((n = m + 1) \wedge nmember(x, d, m) \wedge member(d, c))$
Well-formedness Criteria and Syntactic Restrictions:	
(8)	$in(x, y) \to ptcy(x) = ptcy(y) - 1$
(9)	$spec(x, y) \to ptcy(x) = ptcy(y)$
(10)	$isa^+(x, c) \to x \neq c$
(11)	$in(x, c) \wedge in(x, d) \to c = d$
(12)	$spec(x, s) \wedge spec(x, z) \to s = z$
(13)	$spec(x, s) \wedge in(x, c) \to \exists y : isa^*(s, y) \wedge isa^*(c, y)$
(14)	$spec^*(x, y) \wedge in(x, c) \wedge in(y, d) \to spec^*(c, d)$

In DDI every clabject hierarchy comes with its own set of instantiation levels which is introduced by the potency of the root clabject (a root clabject is a clabject without parent). For example, root clabject Product with potency 3 introduces a clabject hierarchy with three instantiation levels (not counting the instantiation level of the root clabject). These instantiation levels may be given labels. For example, root clabject Product has three instantiation levels, labelled Category, Model, and Individual. Instead of saying *"BMW Z4 is 2-member of Product"* one may now say *"BMW Z4 is a Product Model"*.

We assume acyclic clabject hierarchies (No. 10) with single classification, i.e., every clabject has at most one class-clabject (No. 11), and single generalization, i.e., every clabject has at most one direct superclabject (No. 12).

In the original formalization of DDI [9], in order to avoid multiple inheritance, every clabject had at most one parent, either a class-clabject or a superclabject. We now relax this global restriction and allow clabjects to have both a class-clabject and a superclabject. Class-clabject and superclabject of a clabject must have a common ancestor (No. 13). Further, if a clabject x, which is an instantiation of c, specializes a clabject y, which is an instantiation of d, then c needs to be a specialization of d (No. 14). In the forthcoming sections we will introduce further constraints on the combined use of generalization and instantiation.

Two clabjects may be related via property references. A quintuple $R(x, i, p, j, y)$ is an asserted property reference of source clabject x via a property p to target clabject y with source potency i and target potency j (No. 16), we also say there is a p^{i-j} reference from x to y. The source potency and the target potency

Table 2. Deep Characterization of Clabjects via Property References

Additional Sorts & Asserted Predicates:
(15) P: properties
(16) $R \subseteq C \times \mathbb{N} \times P \times \mathbb{N} \times C$
Additional Well-formedness Criteria and Syntactic Restrictions:
(17) $R(x,i,p,j,y) \wedge R(s,k,p,l,t) \rightarrow \exists c \exists m \exists n \exists d : R(c,m,p,n,d) \wedge isa^*(x,c) \wedge isa^*(s,c)$
(18) $R(x,i,p,j,y) \wedge R(x,k,p,l,d) \rightarrow i = k \wedge j = l \wedge y = d$
(19) $R(x,i,p,j,y) \rightarrow ptcy(x) \geq i \wedge ptcy(y) \geq j$
(20) $R(x,i,p,j,y) \wedge R(c,k,p,l,d) \wedge nmember(x,c,n) \rightarrow n = k - i$
(21) $R(x,i,p,j,y) \wedge R(c,k,p,l,d) \wedge nmember(y,d,n) \rightarrow n = l - j$
(22) $R(x,i,p,j,y) \wedge R(c,k,p,l,d) \wedge isa^+(x,c) \rightarrow isa^*(y,d)$

indicate how many instantiation levels below the source clabject and the target clabject, respectively, the property is to be ultimately instantiated. For example, the $\mathsf{soldPrice}^{2-3}$ reference from Product to Currency is ultimately instantiated by the $\mathsf{soldPrice}^{0-0}$ reference from PetersZ4 to 38,200.

Well-formed property references obey the following constraints. Every property is introduced with a single clabject, i.e., if two clabjects have the same property, then they must have a common ancestor which introduced that property (No. 17). For simplicity (and space limitations) we only consider single-valued properties, that is, there may only be a single property reference per source clabject and property (No. 18). The source and target potency must be lower or equal to the potency of the source and target clabject, respectively (No. 19). When instantiating and refining properties, source and target potencies must be reduced according to the number of instantiation steps between the source clabjects (No. 20) and target clabjects (No. 21), respectively. A clabject c with a reference to clabject d via property p with a target potency higher than 0 or if d is abstract introduces a range referential integrity constraint for all descendants of c: descendants of c may only refer via p to descendants of d (No. 22). This is akin to co-variant refinement and, in terms of the UML, to redefinition of association ends.

3 The Abstract Superclass Rule in the Context of Abstract and Concrete Clabjects

In this section we clarify the distinction between abstract and concrete clabject. We revisit the *abstract superclass rule* and adapt it to the setting of multi-level modeling with abstract and concrete clabjects.

Abstract clabjects combine aspects of abstract classes and abstract objects. *Concrete clabjects* combine aspects of concrete classes and concrete objects. The distinction between abstract and concrete class is described as: 'A class that has the ability to create instances is referred to as *instantiable* or *concrete*, otherwise

it is called *abstract.*' [3] The distinction between *abstract objects* and *concrete objects* is heavily discussed in Philosophy [13] and there are many different ways to explain it. In this paper we follow 'the way of abstraction' which is also followed by Kühne [4]: 'An abstract object represents all instances that are considered to be equivalent to other for a certain purpose [...] An abstract object captures what is universal about a set of instances but resides at the same logical level as the instances'. In DDI, clabjects are either asserted as abstract (No. 23 in Table 3) or derived to be *concrete* (No. 24).

Table 3. Abstract and Concrete Clabjects and the Abstract Superclabject Rule

(23)	$abstract \subseteq C$
(24)	$concrete(x) :\Leftrightarrow x \in C \wedge \neg abstract(x)$
(25)	$spec(x, y) \rightarrow abstract(y)$
(26)	$abstract(c) \wedge in(x, c) \rightarrow abstract(x)$
(27)	$concrete(x) \wedge member(x, y) \rightarrow \exists z : concrete(z) \wedge member(x, z)$

In the literature it has been proposed that only abstract classes may be specialized:

Abstract Superclass Rule: All superclasses are abstract [3] in that they have no direct instances.

Obeying this rule improves the clarity of object-oriented models, especially when the extension (set of instances) of classes is of interest. Despite the trade-off of additional classes to be modeled and maintained, we feel that obeying the abstract superclass rule in multi-level modeling is beneficial because of the increased clarity. That is why in the original DDI approach [9] only concrete clabjects could be instantiated. We now relax this restriction as follows:

Abstract Superclabject Rule: All superclabjects are abstract in that they have no direct concrete instances (but they may have abstract instances).

This is formalized as: All superclabjects are abstract (No. 25). If an abstract clabject acts as class in an instantiation relationship, then the clabject playing the instance role must be abstract as well (No. 26). Every concrete clabject that is member of some clabject must be member of a concrete clabject (No. 27).

The meaning of the allowed kinds of instantiation relationships depends on the abstractness of the related clabjects. An instantiation relationship between a clabject x in the role of the instance and a clabject c in the role of the class, denoted as $in(x, c)$, can be classified as one of the following:

- An *immediate concrete instantiation* relationship is between a concrete clabject in the instance role and a concrete clabject in the class role. For example, the relationship between BMW Z4 and Car is an immediate concrete instantiation relationship, meaning that BMW Z4 is an instance of Car, or, more

exactly, that the object facet of BMW Z4 is an instance of a class-facet of Car.

- A *shared concrete instantiation* relationship is between an abstract clabject x in the instance role and a concrete class c in the class role. For example, the relationship between BMW Motorcycle and Motorcycle is a shared concrete instantiation, meaning that all concrete specializations of BMW Motorcycle, such as F700GS and R1200GS, are instances of Motorcycle.
- A *shared abstract instantiation* relationship is between an abstract clabject x in the instance role and an abstract clabject c in the class role. For example, the relationship between BMW Vehicle and Vehicle is a shared abstract instantiation relationship, meaning that all concrete specializations of BMW Vehicle are instances of a concrete specialization of Vehicle, e.g., BMW Z4 is an instance of Car.

4 Coping with Dual Inheritance

In this section we define the mechanism for inheritance of property references along generalization and instantiation relationships. A clabject inherits both from its class-clabject and from its superclabject. We refer to this specific form of multiple inheritance as *dual inheritance*. Dual inheritance leads to potential conflicts. We propose one way to guarantee that concrete clabjects are conflict-free and, thus, satisfiable.

Table 4. Dual Inheritance

(28) $R^\circ(x,i,p,j,y) :\Leftrightarrow R(x,i,p,j,y) \vee (\exists s : spec(x,s) \wedge R^\star(s,i,p,j,y))$ $\vee (\exists c : in(x,c) \wedge R^\star(c,i+1,p,j,y) \wedge i \geq 0)$
(29) $R^\star(x,i,p,j,y) :\Leftrightarrow R^\circ(x,i,p,j,y) \wedge \neg(\exists j' \exists y' : isa^+(y',y) \wedge R^\circ(x,i,p,j',y'))$
(30) $concrete(x) \wedge spec(x,s) \wedge R^\star(s,i,p,j,y) \wedge in(x,c) \wedge R^\star(c,i+1,p,j',y')$ $\rightarrow isa^*(y,y') \vee isa^*(y',y) \vee (\exists j'' \exists y'' : R(x,i,p,j'',y''))$

Inheritance of property references is defined using the following predicates: predicate R (No. 16 in Table 2) holds *asserted property references* of all clabjects. Auxiliary predicate R° (No. 28 in Table 4) holds *asserted and inherited property references*. Derived predicate $R^\star$ (No. 29) holds *effective property references* which are the most-specific property references out of the asserted and inherited property references.

We first look at asserted and inherited property references (No. 28). From its superclabject a clabject inherits all effective property references. From its class-clabject it inherits all effective property references with a source potency of 1 or above. When inheriting property references from the class-clabject, the source potency is decremented by 1. For example, clabject BMW Z4 inherits from its superclabject BMW Vehicle a soldPrice reference to Euro and from its class-clabject Car a soldPrice reference to EuropeanCurrency.

An inherited or asserted property reference of a clabject x at source potency i for property p to target object y is effective if x has no inherited or asserted property reference for property p to a target object which is a descendant of y (No. 29). For example, the reference to Euro is the effective soldPrice reference for clabject BMW Z4 since it is more specific than the reference to EuropeanCurrency.

We now discuss the role of concrete clabjects in resolving or detecting conflicts introduced by dual inheritance along generalization and instantiation relationships. If a concrete clabject inherits from its superclabject and from its class-clabject references for property p to y and y', respectively, we demand that one of the references is a descendant of the other. If this is not the case the modeler needs to resolve the potential conflict by asserting a reference of property p to some clabject y'' (No. 30) which needs to be, due to the previously introduced constraints, a descendant of both y and y'. If this is not possible, the modeler detects a conflict. This guarantees that concrete clabjects that obey this constraint are satisfiable at the instantiation levels beneath. Conflicts are resolved or detected at concrete objects. For example, BMW Z4 inherits for property engine via specialization from BMW Vehicle and via instantiation from Car references to BMW Engine and CarEngine, respectively. To avoid a potential conflict, BMW Z4 asserts for property engine a reference to EngineK5.

Other approaches to detecting and resolving conflicts are (1) to ignore the problem and accept the possiblity of unsatisfiable properties, (2) to use more sophisticated techniques to decide whether two conflicting property references are satisfiable or not, or (3) to make the above check not only for concrete clabjects but also for abstract clabjects, for example it would make necessary to add a property reference from BMW Motorcycle to a to-be created clabject, e.g., called BMW MotorcycleEngine, that specializes BMW Engine and instantiates MotorcycleEngine and which is a generalization of BMW F700GS and BMW R1200GS. With regard to the effort associated with such an immediate conflict resolution, delaying conflict resolution to concrete clabjects, as introduced above, seems to be a good compromise.

5 Mandatory Constraints at Multiple Levels

By now, it is up to the modeler to decide whether properties are to be instantiated and at which levels they are to be instantiated. In this short section we introduce support for defining mandatory constraints at multiple levels. Mandatory constraints allow to control the stepwise instantiation process by demanding that concrete source clabjects at a given level of the domain of the property need to refer to a concrete clabject at a given level of the range of the property or to a clabject that is a descendant of a concrete clabject at the given level.

In more formal terms, a mandatory constraint $total(i, p, j)$ (No. 31 in Table 5) expresses that property p, which is introduced between clabject c and d with potencies n and m, is mandatory for potencies i and j, with potencies i and j being lower or equal to potencies n and m, respectively (No. 32). This means

Table 5. Mandatory constraints

(31) $total \subseteq \mathbb{N} \times P \times \mathbb{N}$
(32) $total(i,p,j) \rightarrow \exists c \exists n \exists m \exists d : R(c,n,p,m,d) \wedge i \leq n \wedge j \leq m$
(33) $R(c,n,p,m,d) \wedge total(i,p,j) \wedge j \leq m \wedge nmember(x,c,n-i) \wedge concrete(x)$ $\rightarrow \exists j' \exists y \exists y' : R^{\star}(x,i,p,j',y') \wedge isa^{*}(y',y) \wedge concrete(y) \wedge nmember(y,d,m-j)$

that concrete $(n-i)$-members of c must refer via property p to a clabject that is a (descendant of a) $(m-j)$-member of d (No. 33).

For example, property engine is introduced at clabject Vehicle by a reference with source potency 2 and target potency 2 to clabject Engine. The multi-level domain of property engine is given by the 0-, 1-, and 2-members of Vehicle and its multi-level range is given by the 0-, 1-, and 2-members of Engine. Mandatory constraint $total(2, \mathsf{engine}, 2)$ demands that every concrete 0-member of Vehicle refers to some concrete 0-member of Engine or to a descendant of Engine.

6 Related Work

DDI is heavily influenced by the classical work on deep instantiation of Atkinson and Kühne [1]. Kühne and Schreiber [5] introduced the notion of *superclabject* and propose the use of metaclass compatibility rules and represent the abstractness of clabjects by giving them potency 0. De Lara et al. [7] propose to declare abstract clabjects as such. In both approaches, abstractness of clabjects only refers to the inability to create instances; abstract clabjects in the dual sense of abstract object and abstract class are not discussed. Kühne [4] provides an in-depth discussion of the distinction between generalization and classification together with a discussion of abstract objects.

From *M-Objects and M-Relationships* [8], DDI takes the idea that instantiation (then called concretization) levels have a label and that every instantiation (or concretization) hierarchy has its own set of instantiation levels. M-Objects do not come with the possibility to model generalization hierarchies of m-objects at one instantiation level. A comparison with different techniques for deep characterization, then called 'multi-level abstraction', is given in [10]. DDI is further influenced by Pirotte et al's work on Materialization [12]. Similar to M-Objects, materialization does not come with support for generalizing objects at the same abstraction level and does not come with the distinction between concrete and abstract classes. Many aspects of clabject hierarchies with deep instantiation may be alternatively modeled using powertypes [11] or the powertype pattern [2] (see [9]). It is, however, unclear how generalization of clabjects may be modeled using powertypes or the powertype pattern.

The most important related work is that of de Lara et al. [6] on the uniform handling of inheritance at every meta-level, which however does not come with the simplicity and conceptual clarity provided by the abstract superclabject rule. It is open to future work to analyze the trade-offs of both approaches and to combine the advantages of both approaches.

7 Conclusion

We have discussed the distinction between abstract and concrete clabjects as one way of clarifying the meaning of superclabjects in multi-level models. In a simplified setting (only considering single-valued and uni-directional property references) we have introduced and relaxed the abstract superclabject rule, showed how the distinction between abstract and concrete clabjects helps to cope with dual inheritance, and introduced support for mandatory constraints over concrete clabjects at multiple levels. We currently work on extending the full DDI approach (also considering bi-directional and multi-valued relationships) and its ConceptBase implementation [9] along the lines discussed in this paper, especially on extending DDI with full-fledged multi-level multiplicity constraints.

References

1. Atkinson, C., Kühne, T.: The Essence of Multilevel Metamodeling. In: Gogolla, M., Kobryn, C. (eds.) Proceedings of the 4^{th} International Conference on the UML 2001, Toronto, Canada. LNCS, vol. 2185, pp. 19–33. Springer Verlag (Oct 2001)
2. Eriksson, O., Henderson-Sellers, B., Ågerfalk, P.J.: Ontological and linguistic metamodelling revisited: A language use approach. Information & Software Technology 55(12), 2099–2124 (2013)
3. Hürsch, W.L.: Should superclasses be abstract? In: Tokoro, M., Pareschi, R. (eds.) ECOOP. LNCS, vol. 821, pp. 12–31. Springer (1994)
4. Kühne, T.: Contrasting classification with generalisation. In: Kirchberg, M., Link, S. (eds.) APCCM. CRPIT, vol. 96, pp. 71–78. Australian Computer Society (2009)
5. Kühne, T., Schreiber, D.: Can programming be liberated from the two-level style: multi-level programming with deepjava. In: Gabriel, R.P., Bacon, D.F., Lopes, C.V., Jr., G.L.S. (eds.) OOPSLA. pp. 229–244. ACM (2007)
6. de Lara, J., Guerra, E., Cobos, R., Moreno-Llorena, J.: Extending deep metamodelling for practical model-driven engineering. Comput. J. 57(1), 36–58 (2014)
7. de Lara, J., Guerra, E., Cuadrado, J.S.: Model-driven engineering with domain-specific meta-modelling languages. Software & Systems Modeling pp. 1–31 (2013)
8. Neumayr, B., Grün, K., Schrefl, M.: Multi-Level Domain Modeling with M-Objects and M-Relationships. In: Link, S., Kirchberg, M. (eds.) APCCM. CRPIT, vol. 96, pp. 107–116. ACS, Wellington, New Zealand (2009)
9. Neumayr, B., Jeusfeld, M.A., Schrefl, M., Schütz, C.: Dual deep instantiation and its conceptbase implementation. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE. Lecture Notes in Computer Science, vol. 8484, pp. 503–517. Springer (2014)
10. Neumayr, B., Schrefl, M., Thalheim, B.: Modeling techniques for multi-level abstraction. In: Kaschek, R., Delcambre, L.M.L. (eds.) The Evolution of Conceptual Modeling. LNCS, vol. 6520, pp. 68–92. Springer (2008)
11. Odell, J.: Power types. JOOP 7(2), 8–12 (1994)
12. Pirotte, A., Zimányi, E., Massart, D., Yakusheva, T.: Materialization: A Powerful and Ubiquitous Abstraction Pattern. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) VLDB. pp. 630–641. Morgan Kaufmann (1994), 0605
13. Rosen, G.: Abstract objects. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Fall 2014 edn. (2014)