M

Multilevel Modeling

Bernd Neumayr and Christoph G. Schuetz
Department for Business Informatics – Data & Knowledge Engineering, Johannes Kepler University Linz, Linz, Austria

Synonyms

Deep metamodeling; Deep modeling; Multilevel metamodeling

Definition

Multilevel modeling extends object-oriented modeling with multiple levels of instantiation as well as deep characterization. As opposed to traditional two-level modeling, multilevel modeling overcomes the strict separation of class and object. The *clabject*, with class facet and object facet, becomes the central modeling element. Multilevel modeling arranges clabjects in arbitrary-depth hierarchies combining aspects of instantiation and specialization. A clabject not only specifies the schema of its members at the instantiation level immediately below but may also specify the schema of the members of its members, and so forth, at arbitrary instantiation levels below, which is referred to as *deep characterization*.

Historical Background

In object-oriented modeling, a *class* describes the common attributes of its many instances. An instance of a class is also referred to as *object*. A class, however, may itself be seen as an object with a distinct set of attributes that describe the class rather than its instances. The class then becomes instance of another class, the *metaclass*. A self-describing object, on the other hand, defines its class itself. The appellation of such a class varies between modeling approaches (e.g., singleton class, eigenclass, own type).

In multilevel modeling, the class-object or *clabject* – a term coined by Atkinson for a modeling primitive otherwise called "two-faceted construct" [13] with class facet and object facet alike – allows for unbounded meta-modeling with metaclasses, metaclasses of metaclasses, and so forth. The distinction between object, class, metaclass, meta-metaclass, and so forth becomes relative. To its instances an object is a class. To instances of its instances an object is a metaclass. *Deep characterization* allows an object to define schema elements of members at arbitrary instantiation levels below. An object may specify distinct sets of attributes to be instantiated by instances, instances of instances, and so forth.

In the late 1980s, Telos (see cross-reference) and its implementation ConceptBase, which employs the O-Telos dialect of Telos, were among the first approaches to support unbounded metamodeling for data and knowledge engineering, with an arbitrary number of instantiation levels.

L. Liu, M.T. Özsu (eds.), *Encyclopedia of Database Systems*,
DOI 10.1007/978-1-4899-7993-3_80807-1

In Telos, everything is an object and an object may be instance of any other object. Thus, the clabject effectively becomes the main modeling primitive in Telos. Telos, however, offers no direct support for deep characterization.

Among the first systems to introduce a kind of deep characterization was VODAK [7] in the early 1990s, albeit limited to three levels. The VODAK system realizes deep characterization through the organization of objects into types and, in parallel, into classes and metaclasses. Metaclasses, classes, and individual objects are uniformly treated as objects. Every object specifies an own type that describes its specific structure and behavior. Metaclasses and classes additionally specify an instance type describing structure and behavior of its instances. The own type of an object specializes the instance type of the object's class. Metaclasses additionally specify an instance-instance type describing structure and behavior of the instances of its instances. The instance type of a class specializes the instance-instance type of its class, which is a metaclass. In this regard, the instance-of relationship between individual object and class, as well as between class and metaclass, combines aspects of instantiation and specialization. In addition to instance-of relationships, an individual object may generalize another individual object and a class may generalize another class.

Potency-based deep instantiation [1], from the late 1990s onwards, has gone beyond the VODAK approach by allowing arbitrary levels of instantiation. A clabject may define properties with different potencies. Relating potency-based deep instantiation to the VODAK approach, potency-0 properties of a clabject define its own type, potency-1 properties define its instance type, potency-2 properties define its instance-instance type, potency-3 properties define the instance-instance-instance type, and so forth.

The distinction between linguistic and ontological metamodeling [1, 2], also referred to as orthogonal classification architecture (OCA), has since become a central aspect of multilevel modeling. Linguistic metamodeling refers to instantiation of primitives from the modeling language, e.g., CarModel and Porsche911GT3 are linguistic instances of Element. Ontological metamodeling, on the other hand, refers to instantiation relationships based on semantic criteria, e.g., Porsche911GT3 is ontological instance of CarModel. Multilevel modeling approaches commonly are multilevel with respect to ontological metamodeling but remain two-level with respect to linguistic metamodeling.

In the middle of the 1990s, the introduction of materialization [13] as an abstraction pattern brought a new type of relationship with both specialization and instantiation semantics. A materialization relationship associates a class of more abstract objects, e.g., product categories, with a class of more concrete objects, e.g., product models. Clabjects (then referred to as "two-faceted constructs") instantiate the former and specialize the latter. Cascaded use of materialization allows for unbounded levels of classification. Materialization relationships propagate attribute values from abstract to concrete objects either as attribute values or, depending on the propagation mechanism, promoting attribute values from the object facet to attributes of the class facet of a clabject.

The 1990s and 2000s also saw the emergence of power types [6, 12], an expressive pattern for grouping objects. A power type is "a type whose instances are subtypes of another type" [12]. The power type, e.g., CarModel, is always applied in conjunction with a partitioned type, e.g., CarIndividual. The instances of the power type are specializations of the partitioned type. Furthermore, the instances of the power type are typically modeled as clabjects.

More recently, m-objects and m-relationships [10] have combined aspects of deep instantiation, materialization, and power types. An m-object, e.g., Product, has a set of hierarchically ordered, named levels, e.g., Catalog, Category, Model, and Individual. For each level, an m-object defines a class; an m-object instantiates its top-level class, i.e., its own type. The relationships between these classes are kind of materialization. A class at one level is power type of the class at the level immediately below and partitioned type of the class at the level immediately above. An m-object, e.g., Car, may concretize another

m-object, e.g., Product. The concretization relationship between m-objects has an instantiation and specialization facet. The concretizing m-object instantiates the concretized m-object's second-level class and specializes the concretized m-object's other classes. M-relationships relate m-objects at various levels.

A recent study [9] has shown the practical relevance of multilevel modeling by studying the occurrences of multilevel modeling patterns (independent of using a multilevel modeling approach) in existing meta-models from different domains. These multilevel modeling patterns are ubiquitous especially in software architecture and enterprise/process modeling, leading to the conclusion that many modeling problems are intrinsically multilevel. Extensions to deep instantiation and implementations of multilevel modeling environments (e.g., [8]) further improve applicability of multilevel modeling to practical problems.

The discussion [3,5] about the ontological and pragmatic adequacy of modeling with clabjects is ongoing. It has been argued [4, 15] that the compactness gained from representing different semantic relationships (such as instantiation and specialization) by a single relationship – referred to as ontological instance-of [2], materialization [13], or concretization [10] – leads to models that are more difficult to understand since "semantic clarity is traded for reduction of model size" [4]. The MLT multilevel theory [4] fosters semantic clarity by relating multilevel modeling with the ontological foundations of conceptual modeling.

Scientific Fundamentals

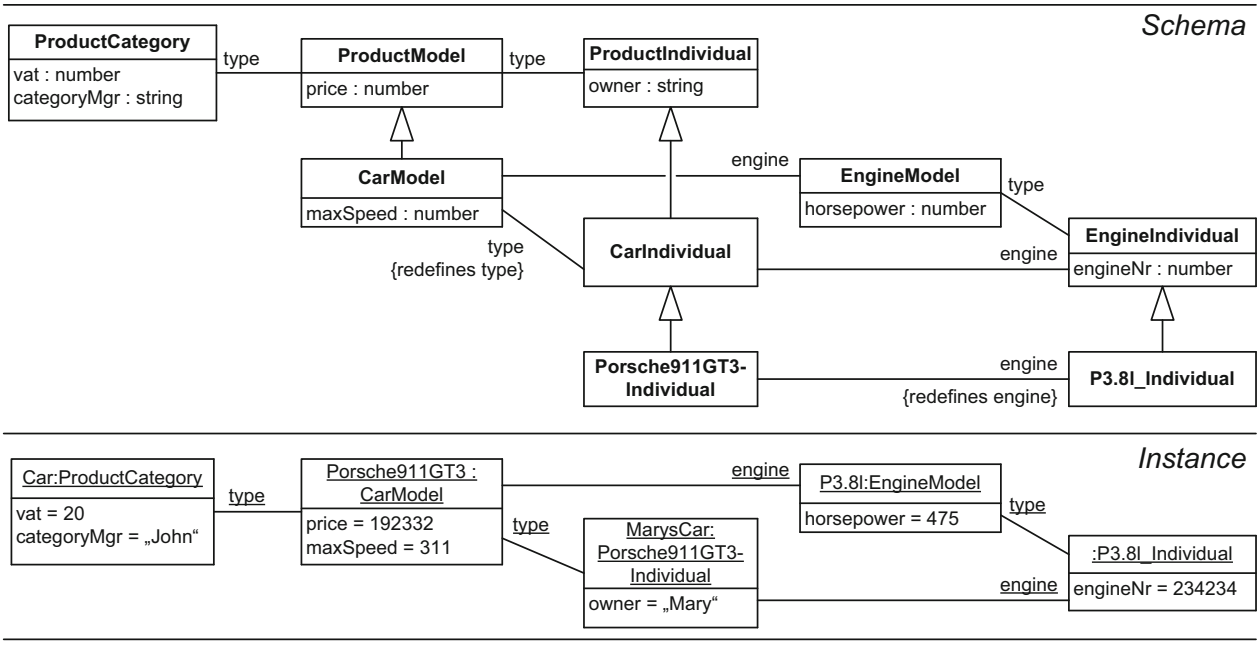
Multilevel modeling is an extension of traditional two-level modeling. The core patterns of multilevel modeling comprise class/object, property range/value, extension, and extension by auxiliary class. Among the additional patterns are clabject generalization, property specialization, heterogeneous level hierarchies, and level-crossing relationships. Multilevel modeling approaches aim at the creation of concise models with clear semantics as well as support for flexible querying.

Background from Two-Level Modeling

A database typically consists of database schema and database instance. A conceptual data model or semantic data model (see Cross-Reference) consists of an intensional definition of the database schema as well as an extensional definition of the database instance. Using terms from UML (see Cross-Reference), a conceptual data model describes the database schema using classes, associations, and attributes; attributes and associations are also referred to as properties. A conceptual data model describes the database instance using objects, links, and attribute values. Links and attribute values, also referred to as property values, are collected into slots of objects.

In a database schema, classes serve a dual purpose. First, classes collect objects into sets, i.e., each class is the set of its member objects, thereby providing an entry point for queries. Second, classes provide the structure of the data, i.e., each class defines the common attributes of its member objects. Whereas the database instance typically is considered dynamic, i.e., updated by users and applications, the database schema typically is considered static, i.e., fixed a priori and not changed by users or applications.

In many application domains, e.g., enterprise databases, the strong assumption of fixed database schema with a static set of classes versus dynamic database instance with changing objects is problematic. For example, a company may organize its product catalog in a database with product categories collected into the ProductCategory class, product models collected into the ProductModel class, and individual products collected into the ProductIndividual class (Fig. 1). The addition of a new product category, e.g., "Car," would see an instance of ProductCategory, e.g., an object Car, added to the database instance. In order to represent the particularities of a newly added product category, the database schema may be extended with subclasses of ProductModel and ProductIndividual. For example, the CarModel class introduces the maximum speed as attribute and an association to the EngineModel class with the specified horsepower as attribute; each



Multilevel Modeling, Fig. 1 Two-level representation (in UML) of a multilevel product catalog

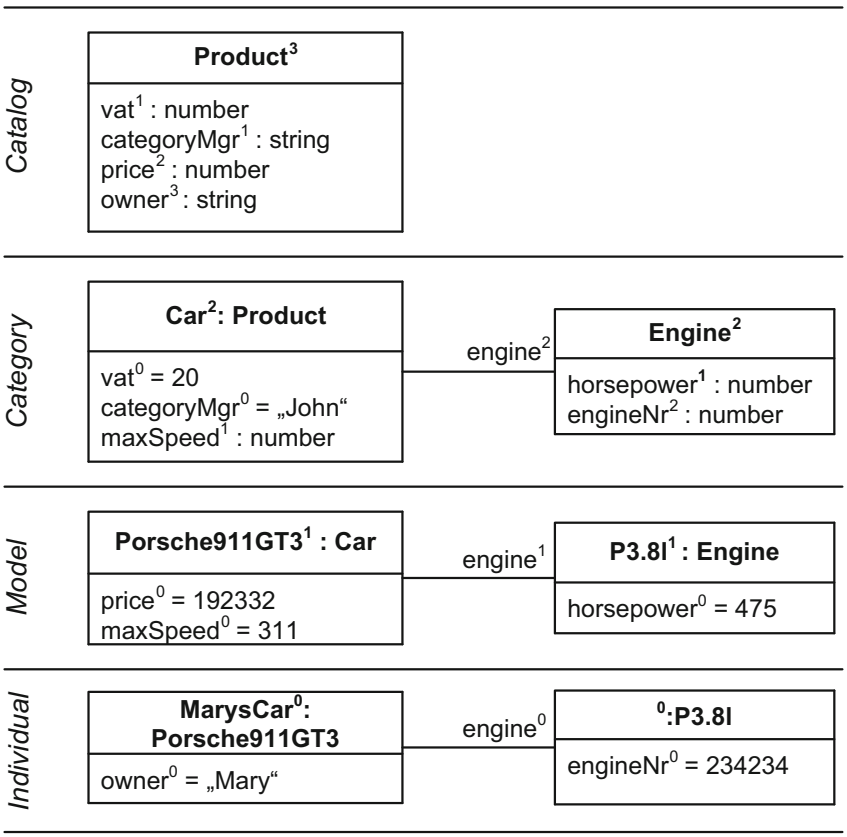
CarModel object links to an EngineModel object. The CarIndividual class introduces an association to the EngineIndividual class with the engine identity number as attribute; each CarIndividual object links to an EngineIndividual object. Likewise, the addition of a new product model, e.g., "Porsche 911 GT3," would see an instance of ProductModel (or one of its subclasses), e.g., Porsche911GT3, added to the database instance. Furthermore, individual physical entities of the newly added product model may have their properties restricted, e.g., Porsche911GT3Individual objects have a P3.8l_Individual engine. Thus, the manipulation of the database instance may also entail introduction of classes in the database schema.

Multilevel modeling approaches aim for a better support of modeling situations with fluid schema/instance boundaries as well as avoidance of problems associated with the redundant representation of classes and objects in database schema and instance. Different approaches to multilevel modeling all have in common their reliance on a modeling primitive that embodies characteristics of class and object alike, usually referred to as *clabject*. For example, a product category "Car" may be represented by the Car clabject that instantiates – also referred to as concretization, ontological instantiation, or materialization in other multilevel modeling approaches – a ProductCategory class and serves as class of car models such as "Porsche 911 GT3." The clabject is believed to avoid accidental complexity in modeling situations lacking clear separation of schema and instance data.

Core Multilevel Modeling Patterns

In the remainder of this section, Fig. 2 serves to illustrate frequent patterns of multilevel modeling, based on the patterns identified by de Lara et al. [9]. The example shows a multilevel representation of a product catalog with different product categories, product models, and product individuals. The multilevel representation relies on deep instantiation as modeling approach together with named levels to increase its semantic clarity. The core patterns also translate to other multilevel modeling approaches.

The *class/object pattern* (cf. type-object [9]) is arguably the central pattern of multilevel modeling. The class/object pattern consists of two model elements, a class and an object. Many multilevel modeling approaches allow representation of the class/object pattern as a single model



Multilevel Modeling, Fig. 2 Multilevel representation (using deep instantiation and named levels) of a multilevel product catalog

element with class facet and object facet. Consider, for example, the Porsche911GT3 clabject in Fig. 2 which is first an object instantiating the Car clabject with a value of 311 for the maxSpeed attribute. The Porsche911GT3 clabject also acts as class collecting instances such as MarysCar.

The class/object pattern may be cascaded and combined with deep characterization. A clabject then not only characterizes its immediate instances but also instances of its instances (which we refer to as deep instances) at arbitrary instantiation levels below. For example, in Fig. 2, Product represents the class of product categories (including the *Car* category) with properties vat and categoryMgr (of potency 1), the class of product models (including the *Porsche 911 GT3* model) with a price property (of potency 2) as well as the class of individual physical entities (including *Mary's Car*) with the owner property (of potency 3). The Car clabject is immediate instance of Product assigning a value to the vat and categoryMgr properties, the Porsche911GT3 clabject is deep instance of Product two levels below assigning a value to the price property, and the MarysCar clabject is deep instance of Product three instantiation levels below assigning a value to the owner property.

The *property range/value pattern* (cf. relation configuration [9]) allows for the redefinition of the range of a property according to property values of a clabject. For example, clabject Porsche911GT3 has P3.8l as value of the engine property. This property value acts as refinement of the range of the engine property, i.e., instances of Porsche911GT3 may only be related via the engine property to instances of P3.8l.

The class/object pattern and the property range/value pattern are two forms of the *schema/instance duality* which is central to multilevel modeling. Objects at higher levels act as specialized classes at lower levels. Property values at higher levels act as refined property ranges at lower levels.

The *extension* (cf. dynamic feature [9]) pattern allows for dynamic additions to the database schema. For example, the Car clabject in Fig. 2 extends the schema of its instances with respect to the 2-potency properties defined by the Product

clabject. Consider the Product clabject which defines the price attribute (of potency 2) for deep instances two levels below. The Car clabject extends this schema with a maxSpeed attribute which is to be instantiated by instances of Car, i.e., deep instances of Product two instantiation levels below.

The *extension by auxiliary class* (cf. dynamic auxiliary domain concept [9]) pattern, in combination with the extension pattern, permits the dynamic addition of auxiliary classes that act as range of dynamically added properties. For example, Car extends the schema of its instances and of the instances of its instances with the engine property and introduces the Engine auxiliary class with the horsepower attribute. The P3.8l clabject instantiates Engine, defining a value of 475 for the horsepower property and being linked to Porsche911GT3, an instance of Car.

Class and Metaclass Facets of Clabjects

In multilevel clabject hierarchies with deep characterization, a clabject becomes a multifaceted construct with multiple class and metaclass facets. Likewise the instantiation relationship between two clabjects becomes a multifaceted relationship combining multiple instantiation and specialization relationships. Level names together with procedures for the construction of qualified names from clabject names and level names help to make more explicit these different facets and allow to directly refer to them and to describe the different relationships between facets of clabjects.

A qualified name combining a clabject name and a level name refers to a class facet of a clabject (see Def. 5 in [10], with clabjects referred to as m-objects). Qualified names act as substitutes for explicitly modeled classes, e.g., ProductModel and CarModel of Fig. 1, and serve as predefined entry points for querying. For example, clabject names Product and Car are combined with level name Model to construct qualified names Product⟨Model⟩ and Car⟨Model⟩ to refer to class facets of clabjects Product and Car that collect clabjects at level Model, such as Porsche911GT3, as members. The Porsche911GT3 clabject is member of Product⟨Model⟩ and Car⟨Model⟩. The Car⟨Model⟩ class facet is a subclass of the Product⟨Model⟩ class facet.

Level names and clabject names can further be combined to qualified names that refer to different metaclass facets of clabjects [10]. For example, Product⟨Category⟨Model⟩⟩ refers to the metaclass facet of clabject Product that has class facets such as Car⟨Model⟩ and Motorcycle⟨Model⟩ as members. Likewise, Product⟨Category⟨Individual⟩⟩ refers to the metaclass facet of the Product clabject which has class facets such as Car⟨Individual⟩ and Motorcycle⟨Individual⟩ as members. The Car⟨Model⟨Individual⟩⟩ metaclass facet, with members such as the Porsche911GT3 ⟨Individual⟩ class facet, is a subclass of the Product⟨Model⟨Individual⟩⟩ metaclass facet.

Additional Multilevel Modeling Patterns

Clabject generalization (cf. element classification [9]) allows for the arrangement in inheritance hierarchies of clabjects at the same level. For example, after introducing the additional Motorcycle product category, which also has a maxSpeed property, one could generalize the Motorcycle and Car clabjects to the generalized Vehicle clabject and move the maxSpeed property to Vehicle.

Multilevel property specialization allows for the specialization of properties to more specific properties. For example, a component property introduced at the Product clabject may be specialized as engine, transmission, and others by the Car clabject. Property specialization may be dependent or independent from the stepwise instantiation of clabjects. Materialization [13] comes with a slightly different approach, referred to as type 3 mechanism, where the values of a clabject's type 3 property become properties of the instance of the clabject.

Heterogeneous level hierarchies allow for relating clabjects in hierarchies with different numbers of levels. For example, in the product catalog example in Fig. 2, products are described at three levels of instantiation, namely, product category, product model, and product individual. Other

kinds of entities may come with a different number of instantiation levels. For example, persons may be organized at a single level of instantiation, i.e., a Person class may be specified as a simple class with individual persons such as John and Mary as instances.

Heterogeneous level hierarchies are naturally supported by powertype-based approaches including materialization. The deep instantiation approach has been extended in this direction by the proposal of m-objects and m-relationships [10], dual deep instantiation [11], and by de Lara et al. [9]. M-objects additionally come with the possibility to dynamically introduce additional levels in sub-hierarchies, for example, an additional *car brand* level may be introduced for the *car* product category above the *car model* level.

Level-crossing relationships transcend strict metamodeling. The original approach to deep modeling [3] adheres to the rules of strict metamodeling, i.e., every element at some level is an instance of an element at the next higher level, and the instance-of relationship is the only type of relationship that crosses level boundaries. It has been observed by many authors that especially the latter restriction can hardly be maintained in many situations. For example, when modeling persons by a simple class Person which then acts as range of the categoryMgr and owner properties of the Product clabject, both the categoryMgr and the owner property crosses level boundaries. Level-crossing relationships are naturally supported by powertype-based approaches. M-objects and m-relationships [10] and dual deep instantiation [11] support level-crossing relationships, the latter using dual potencies. The approach by de Lara et al. [8] supports level-crossing relationships using deep references.

Key Applications

In model-driven software engineering, multilevel modeling is applied to the specification of domain-specific languages [9]. In multilevel business process modeling [14], process instances (i.e., object life cycles) at higher levels impose dynamic constraints on the life cycles of their members at lower levels. In application integration, a multilevel model acts as the conceptual model of an integrated enterprise database [11] or a heterogeneous IT ecosystem [15].

Cross-References

- ▸ Deep Instantiation
- ▸ Metamodel
- ▸ Semantic Data Model
- ▸ Telos
- ▸ Unified Modeling Language

Recommended Reading

1. Atkinson C, Kühne T. The essence of multilevel metamodeling. In: Gogolla M, Kobryn C, editors. UML 2001. LNCS, vol. 2185. Springer; 2001. p. 19–33.
2. Atkinson C, Kühne T. Model-driven development: a metamodeling foundation. IEEE Softw. 2003;20(5):36–41.
3. Atkinson C, Kühne T. In defence of deep modelling. Inf Softw Technol. 2015;64:36–51.
4. Carvalho VA, Almeida JPA, Fonseca CM, Guizzardi G. Extending the foundations of ontology-based conceptual modeling with a multi-level theory. In: Johannesson P, Lee M, Liddle SW, Opdahl AL, López OP, editors. ER 2015. LNCS, vol. 9381. Springer; 2015. p. 119–33.
5. Eriksson O, Henderson-Sellers B, Ågerfalk PJ. Ontological and linguistic metamodelling revisited: a language use approach. Inf Softw Technol. 2013;55(12):2099–124.
6. Gonzalez-Perez C, Henderson-Sellers B. A powertype-based metamodelling framework. Softw Syst Model. 2006;5(1):72–90.
7. Klas W, Neuhold EJ, Schrefl M. Metaclasses in VODAK and their application in database integration. GMD Technical Report (Arbeitspapiere der GMD); 1990.
8. de Lara J, Guerra E, Cobos R, Moreno-Llorena J. Extending deep meta-modelling for practical model-driven engineering. Comput J. 2014;57(1):36–58.
9. de Lara J, Guerra E, Cuadrado JS. When and how to use multilevel modelling. ACM Trans Softw Eng Methodol. 2014;24(2):12:1–12:46.
10. Neumayr B, Grün K, Schrefl M. Multi-level domain modeling with m-objects and m-relationships. In: Link S, Kirchberg M, editors. APCCM 2009. CRPIT, vol. 96. Australian Computer Society; 2009. p. 107–16.

11. Neumayr B, Jeusfeld MA, 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, editors. CAiSE 2014. LNCS, vol. 8484. Springer; 2014. p. 503–17.
12. Odell J. Power types. J Object-Oriented Prog. 1994;7(2):8–12.
13. Pirotte A, Zimányi E, Massart D, Yakusheva T. Materialization: a powerful and ubiquitous abstraction pattern. In: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB); 1994. p. 630–641.
14. Schuetz CG. Multilevel business processes – modeling and data analysis. Springer Vieweg Wiesbaden; 2015.
15. Selway M, Stumptner M, Mayer W, Jordan A, Grossmann G, Schrefl M. A conceptual framework for large-scale ecosystem interoperability. In: Johannesson P, Lee M, Liddle SW, Opdahl AL, López OP, editors. ER 2015. LNCS, vol. 9381. Springer; 2015. p. 287–301.