Multilevel Modelling for Interoperability

Andreas Jordan, Wolfgang Mayer, and Markus Stumptner

University of South Australia, Australia,
{andreas.jordan}@mymail.unisa.edu.au,
{wolfgang.mayer,mst}@cs.unisa.edu.au

Abstract. Model-driven approaches to establishing interoperability between information systems have recently embraced meta-modelling frameworks spanning multiple levels. However, no consensus has yet been established as to which techniques adequately support situations where heterogeneous domain-specific models must be linked within a common modelling approach. We introduce modelling primitives that support the multilevel modelling paradigm for information integration in heterogeneous information systems. We extend standard specialisation and instantiation mechanisms to enable the propagation of semantic and schema information across model levels and compare our approach using a suite of criteria to show that our approach improves modularity, redundancy, query complexity, and level stratification.

1 Introduction

The core idea of conceptual modelling since the ER-Model [1] is the notion of defining a particular language that can be used to effectively construct concise and clear domain models. Carrying over conceptual model characteristics into object-oriented software models (such as UML), led to rich design methods including meta-modelling frameworks: models that can define the languages used to produce the actual conceptual and design models for information systems. For example, Atkinson and Kühne [2] analyse the way in which design/product relationships cause inconsistencies with the fixed meta-model hierarchy of the UML standard. However, advanced application domains, such as those requiring the in-depth modelling of products, e.g. engineering standards, online catalogues, reference data libraries, continue to present a particular challenge to meta-modelling frameworks. This is further exacerbated when investigating appropriate multilevel modelling abstractions suitable for applications in an interoperability setting.

In this paper we discuss the multi-level modelling extensions that we believe will enable automated, model-driven transformation of data between two of the major data standards in the Oil & Gas industry [3,4]. A key concern is the notion that the same component in a real system can be subject to multiple classifications. This is to allow a multi-dimensional view of the modelled data, from recording specific technical solutions and constraints to be used for technical access as well as to serve business/ERP requirements.

The contribution of this paper is a set of modelling extensions to overcome limitations of existing approaches, such as the "heterogeneous level" issue of [5].

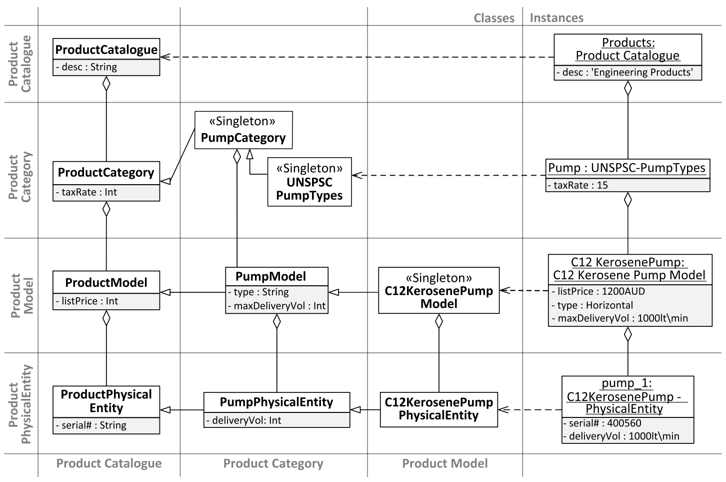

Fig. 1: Product catalogue modelled in plain UML, using generalisation, instantiation and aggregation (adapted from [5])

2 Motivating Example and Multi-Level Modelling

There are several important aspects and complexities to the domain that need to be modelled. We identify three levels of data: business level, specification level, and physical entity level. Firstly, both designs and the physical entities of a product catalogue must be represented and have their own life-cycles which forms the physical entity level. Secondly, a second level of classification provides the specifications of the physical entities. Thirdly, a third level of classification, or categorisation, must be imposed on the designs. This third level of classification also consists of complex taxonomies relevant from the business/ERP perspective. The modelling approaches taken by established related standards demand a flexible approach to support mappings and model transformations between them.

A UML-based approach to modelling this situation is shown in Figure 1, in which ProductCatalogue, ProductCategory, ProductModel, and ProductPhysicalEntity are modelled as an abstraction hierarchy using aggregation. Specialisation is used to distinguish categories (i.e. business classifications), models (i.e. designs), and physical entities of pumps and motors.

Modelling complex domains using UML suffers from what Henderson-Sellers et al. describe as *enactment* (see [6]). Moreover, the model in Figure 1 contains a number of redundant classes as discussed in [5,7]; the misuse of the aggregation relationship to represent a membership and/or classification relation; and the result that the physical entities are not intuitively instances of their product models, but rather are *part-of* their models. Basically, a domain model that seems to naturally have multiple levels of classification is forced into a 2-level modelling framework so that all aspects of the product data exist at the instance level. In an attempt to resolve such issues, multi-level modelling (MLM) techniques have been developed.

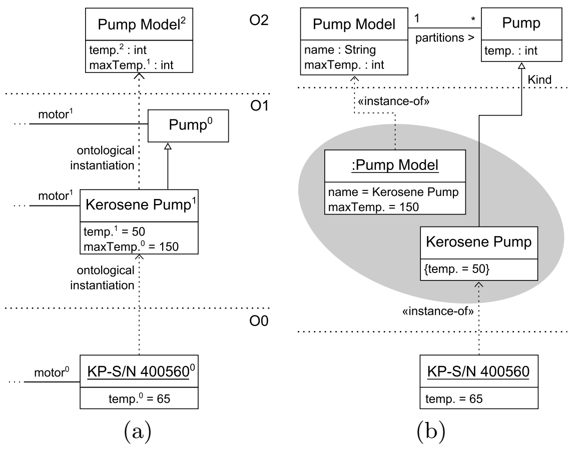

Fig. 2: Deep Instantiation (a) and Power Type (b) models of a product catalogue

The concept of *potency* [7] was originally introduced for *Deep Instantiation (DI)* to support the transfer of information across more than one level of instantiation. DI introduced the concept of *ontological* instantiation, which is a domain specific instantiation relationship (different from standard *linguistic* instantiation in UML meta-modelling).

Using DI techniques, which works within the boundaries of strict meta-modelling, invariably results in relationships (other than instantiation) crossing level boundaries, which is not permitted under *strict* meta-modelling. For example in Figure 2a, if the attributes $temp^2$ and $maxTemp^1$ were modelled as associations to values of the type DegreeCelsius (rather than just integers that must be interpreted as such), no matter at what level DegreeCelsius is placed it would result in an association crossing a level boundary at some point [8]. Moreover, if an additional level of instantiation was introduced into a DI-based model, global changes to the potency values of all concepts in the model (as opposed to just the subhierarchy directly affected) are required.

The concept of *power types* [9] has also been applied in the context of MLM frameworks [10]. Basically, for a power type t of another type u, the instances of t must be subtypes of u. In contrast to potency, power types do not necessarily provide deep instantiation semantics; they provide semantics closer to the conceptual situation being represented by clearly identifying the concepts involved, their relationships, and their properties.

For example, Figure 2b shows one instance of the Power Type pattern, where type and instance facets of a concept are depicted within the ellipse. The power type Pump Model for the type Pump is displayed; the concept ProductCategory would be represented as cascading uses of the power type pattern.

M-objects (multi-level objects) and m-relationships (multi-level relationships) were introduced in [11] along with the *concretization* relation which stratifies objects and relationships into multiple levels of abstraction. The m-objects

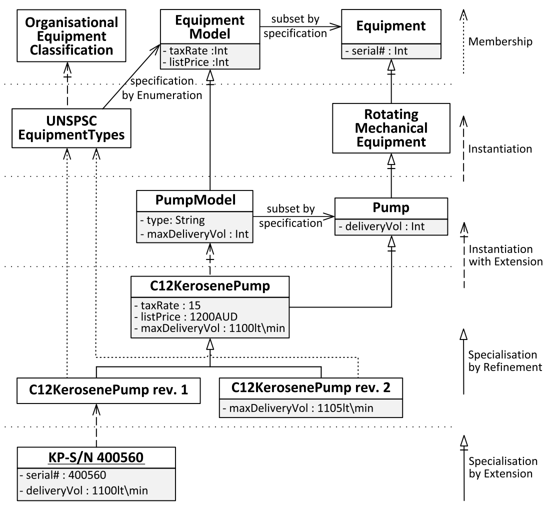

Fig. 3: Product catalogue example modelled in our framework

technique allows for the encapsulation of the different levels that relate to a specific domain concept in the m-object representing that concept. Furthermore, it combines the different abstraction hierarchies for specialisation, instantiation, and aggregation into a single concretization hierarchy. As such, the example situation would be modelled with a top-level concept ProductCatalogue containing the definition of its levels of abstraction: category, model, and physical entity. The lower levels would then include objects for the different PumpCategories, PumpModels, and PhysicalPumps, respectively.

While the m-objects technique produces concise models with a minimum number of relations, they hide complex semantics. This can lead to difficulties in interpreting the models as the concretization relation between two m-objects (or two m-relationships) must be interpreted in a multi-faceted way.

3 Modelling Extensions to Support Multilevel Modelling

Recently, the notion that every part of an object model is an object (embraced in certain OO programming and conceptual modelling [2] approaches) has regained popularity [8,6]. We follow this approach and treat all model elements as objects that can include both type and instance facets.

The MLM approaches summarised in the previous section place the emphasis on a clear and consistent layering of the levels in the model. However, when applied to ontological instantiation in information modelling for systems interoperability, the level construct actually becomes an artefact of the modelling outcome. Domain engineers do not think in levels; they work in terms of semantic relationships, and there can be arbitrary many levels for each of them. Capturing these in terms of potency is useful if the originating models are available for reorganisation according to the strictness criterion and if solid estimates exist on the expected number of levels that future developers may want to add in a particular domain. In the interoperability space this is generally not the case. We will now examine the specific relationships used in our framework.

3.1 Instantiation vs. specialisation relationship

In MLM, the existence of two basic relationship types for increasing specificity is commonly assumed: (1) instantiation which can be broadly defined by the conformance of instances to types in which specificity is increased by assigning distinct values to attributes (if they exist); and (2) specialisation (or generalisation), which makes a concept more specific by including finer distinction (based on the Liskov Substitution Principle (LSP) [12]). Intuitively our modelling extensions are compatible with the intuition of the LSP, however formal proof to verify such a claim is left for future work.

Viewed in terms of increasing specificity it becomes obvious that what we have generally referred to as specificity, and what MLM approaches measure by potency, actually represents *two different conceptual relations*: One is the reference of a specification to the specified item. This captures the powertype aspect of the relationship and is also at the core of the *materialization* relationship in the conceptual modelling literature. The other is the definition of the vocabulary used in the specification. It should be clear that this distinction is of fundamental importance for interoperability mappings, since they rely on being able to treat the specification of a system separately from its runtime state.

We characterise specialisation relationships along the lines of [13] to distinguish a relationship that *extends* a class (by adding attributes, associations, or behaviour) from one that *refines* a class (by adding granularity to the description). We call a specialisation relationship that extends the parent class a *Specialisation by Extension* (*SbE*) and adopt standard monotonic specialisation semantics. As such it can include but does not necessitate *refinement*. Most importantly, this form of specialisation introduces a new model level (as opposed to standard meta-modelling and MLM techniques where modelling levels are fixed a priori).

In contrast, a specialisation relationship that only refines the parent class is called *Specialisation by Refinement* (*SpecR*), which allows the introduction of subtypes that restrict the domain of the specialised class (e.g., by restricting the domains of properties and associations, or adding domain constraints on properties) but without introducing additional model levels. This allows for an arbitrary number of subtypes that simply refine the level of granularity.

We characterise instantiation as either *Instantiation with Extension* (*InstX*) or *Standard Instantiation* (*InstN*). Both forms of instantiation introduce additional model levels; however, standard instantiation means that all attributes of the type being instantiated must be assigned a value from their domain, while *InstX* allows for additional attributes, behaviour, etc. to be added to the concept that can then be instantiated or inherited further to lower model levels.

The *Subset by Specification* (*SbS*) relationship represents the existence of a class of specification construct that identifies particular subtypes of another type. The specification (for example EquipmentModel) exists at the same level as the type it refers to, because the specification can only refer to properties of that type (and not to properties of individual subtypes). It is, however, possible to define subtypes of this type of specification to reference particular properties (so EquipmentModel can be specialised to PumpModel which can refer to properties of Pumps). Together with $InstX$, this relationship can be used to construct the powertype pattern [9]. In Figure 3, PumpModel could be modelled as the powertype of EquipmentModel since it specialises EquipmentModel and the instance of PumpModel, i.e. C12KerosenePump, is an indirect subtype (by extension) of Equipment.

Different to UML associations, most conceptual models permit general associations that represent domain specific relationships. We identify two particular such associations. The first we call $Member$. In contrast to the instantiation relation which it otherwise resembles, *member* does not have any constraints on the assignment of values to attributes as it is purely a basic set membership relation. However, this does not preclude the specification of membership criteria, or constraints, for allowing or disallowing the possible members of a set. $Member$ does require the existence of a "primary" instantiation relation. The second such association is *Specification by Enumeration SbE*, which represents a relationship between concepts A and B that describes how the extensions of the sets of entities that they represent are related. Specifically, it means that the *members* of A are *instances* of B. These relations permit us to emulate multiple inheritance by establishing statements about categories, without defining the attributes or associations of the included object or the category itself.

Below we present the formal properties of our model. The first argument denotes the instance or specialised concept, whereas the second argument represents the type or general concept. Relations $Member$ and SbE are jointly used to specify a classification scheme that subdivides the instances of a concept. Model elements are organised in levels numbered such that the type-level number is one less than the instance-level. Function *level* returns the level number of each element. Each element is described by a set of typed attributes, some of which may have assigned values, and a constraint expression stating necessary properties of the object's instances. Function *attr* maps each object to a set of attribute names, function *type* associates a data type to each object-attribute pair, and partial function *val* returns the value associated with a given object-attribute pair (if one has been assigned). We implicitly assume that primitive data types and their possible values are implicitly modelled as concepts and instances, respectively. Function *desc* maps each concept to a constraint expression capturing the properties that all instances of the object must satisfy, and *names* returns the attribute labels used in said description.

Domains:

O	Set of objects	L	Set of attribute labels
$\mathbb{N}$	Natural numbers	$\mathcal{S}$	Constraint language over attributes in L

Functions:

$level:$	$O \mapsto \mathbb{N}$	The level at which an object is defined (zero is top level)
$attr:$	$O \mapsto 2^L$	The set of attribute labels for an object
$type:$	$O \times L \mapsto O$	The type of an attribute of an object
$val:$	$O \times L \mapsto O$	The value of an attribute of an object
$desc:$	$O \mapsto \mathcal{S}$	Constraint expression instances of an object must satisfy
$names:$	$\mathcal{S} \mapsto 2^L$	The attribute labels used in a constraint expression

Relations:

InstN	$\subseteq O \times O$	$\mathit{InstN}(x,c)$: x is an instance of c
InstX	$\subseteq O \times O$	$\mathit{InstX}(x,c)$: x is an instance-with-extension of c
SpecR	$\subseteq O \times O$	$\mathit{SpecR}(c,c')$: c is a specialisation-by-refinement of c'
SpecX	$\subseteq O \times O$	$\mathit{SpecX}(c,c')$: c is a specialisation-by-extension of c'
Member	$\subseteq O \times O$	$\mathit{Member}(x,c)$: x is in a *Member* relation with c
SbE	$\subseteq O \times O$	$\mathit{SbE}(c,c')$: c is a *Specification-by-Enumeration* of c'
SbS	$\subseteq O \times O$	$\mathit{SbS}(c,c')$: c is a *Subset-by-Specification* of c'

We use (ρ^*) ρ^+ to denote the (reflexive-)transitive closure of relation ρ.

Definitions

The *Spec* relation generalises the two forms of specialisation
$\mathit{Spec}(c,c') \leftrightarrow (\mathit{SpecR}(c,c') \vee \mathit{SpecX}(c,c'))$

An object is a leaf iff it has no specialisations $\mathit{Leaf}(c) \leftrightarrow \nexists x : \mathit{Spec}(x,c)$

The *Inst* relation generalises the two forms of instantiation
$\mathit{Inst}(x,c) \leftrightarrow \mathit{InstN}(x,c) \vee \mathit{InstX}(x,c)$

Object x is a general instance of c iff it is instance of or specialises an instance of (a specialisation of) c
$\mathit{GenInst}(x,c) \leftrightarrow \exists x' \exists c' : \mathit{Spec}^*(x,x') \wedge \mathit{Inst}(x',c') \wedge \mathit{Spec}^*(c',c)$

Axioms for specialisation and instantiation

Inst, *Spec*, *Member* are jointly acyclic $(\mathit{Inst} \cup \mathit{Spec} \cup \mathit{Member})^*(x,c) \rightarrow x \neq c$

InstN and *InstX*, *SpecR* and *SpecX* are mutually exclusive
$\mathit{InstN}(x,c) \rightarrow \nexists c' : \mathit{InstX}(x,c')$ $\mathit{InstX}(x,c) \rightarrow \nexists c' : \mathit{InstN}(x,c')$
$\mathit{SpecR}(x,c) \rightarrow \nexists c' : \mathit{SpecX}(x,c')$ $\mathit{SpecX}(x,c) \rightarrow \nexists c' : \mathit{SpecR}(x,c')$

Inst, *Spec* restricted to a unique parent object
$\rho(x,c) \wedge \rho(x,c') \rightarrow c = c'$ for $\rho \in \{\mathit{Inst}, \mathit{Spec}\}$

Only leaf objects can be instantiated $\mathit{Inst}(x,c) \rightarrow \mathit{Leaf}(c)$

Level consistent with *Inst*, *Spec*
$\mathit{Inst}(x,c) \rightarrow \mathit{level}(x) = \mathit{level}(c) + 1$
$\mathit{SpecR}(x,c) \rightarrow \mathit{level}(x) = \mathit{level}(c)$ $\mathit{SpecX}(x,c) \rightarrow \mathit{level}(c) \leq \mathit{level}(x)$

Axioms for schema consistency

Attribute type must be consistent with level order
$a \in \mathit{attr}(x) \wedge t = \mathit{type}(x,a) \rightarrow \mathit{level}(t) < \mathit{level}(x)$

SpecR does not change attribute set $\mathit{SpecR}(x,c) \rightarrow \mathit{attr}(c) = \mathit{attr}(x)$

SpecX extends attribute set $\mathit{SpecX}(x,c) \rightarrow \mathit{attr}(c) \subset \mathit{attr}(x)$

Spec may specialise attribute types
$\mathit{Spec}(x,c) \wedge a \in \mathit{attr}(x) \cap \mathit{attr}(c) \wedge t = \mathit{type}(x,a) \wedge t' = \mathit{type}(c,a) \rightarrow \mathit{Spec}^*(t,t')$

InstN does not change attribute set $\mathit{InstN}(x,c) \rightarrow \mathit{attr}(c) = \mathit{attr}(x)$

InstX extends attribute set $\mathit{InstX}(x,c) \rightarrow \mathit{attr}(c) \subset \mathit{attr}(x)$

Inst does not change attribute type
$\mathit{Inst}(x,c) \wedge a \in \mathit{attr}(x) \cap \mathit{attr}(c) \rightarrow \mathit{type}(x,a) = \mathit{type}(c,a)$

Inst instantiates all attributes of c
$\mathit{Inst}(x,c) \wedge a \in \mathit{attr}(c) \wedge t = \mathit{type}(x,a) \rightarrow \exists v : v = \mathit{val}(x,a) \wedge \mathit{GenInst}(v,t)$

Axioms for Member and Specification-by-Enumeration

Member relation must be consistent with level order
$\mathit{Member}(x,c) \rightarrow \mathit{level}(c) < \mathit{level}(x)$

Specification-by-Enumeration must be consistent with level order
$\mathit{SbE}(c,t) \rightarrow \mathit{level}(t) \leq \mathit{level}(c)$

Specification-by-Enumeration classifies general instance of the type
$\mathit{SbE}(c,t) \wedge \mathit{Member}(x,c) \rightarrow \mathit{GenInst}(x,t)$

Axioms for Subset-by-Specification

Subset-by-Specification must be consistent with level order
$SbS(c, c') \rightarrow level(c) = level(c')$
Instances of each subset specification must be a specialisation of the partitioned type
$SbS(c, c') \wedge Inst(x, c) \rightarrow Spec^{+}(x, c')$
The specification may refer only to the attributes of the partitioned type
$SbS(c, c') \wedge \phi = desc(c) \rightarrow names(\phi) \subseteq attr(c')$

Axioms for Descriptions
Constraints can use only attributes defined in its associated object
$\phi = desc(c) \rightarrow names(\phi) \subseteq attr(c)$
Constraints must respect the specialisation hierarchy
$Spec(c, c') \wedge \phi = desc(c) \wedge \phi' = desc(c') \rightarrow (\phi(x) \rightarrow \phi'(x))$
Instances of a object must satisfy its constraint
$GenInst(x, c) \wedge \phi = desc(c) \rightarrow \phi(x)$
Members in a *Member* relation must satisfy its classifier's constraint
$Member(x, c) \wedge \phi = desc(c) \rightarrow \phi(x)$

4 Comparison of MLM Techniques

A comparison of MLM approaches was performed in [5] based on a number of criteria from the perspective of reducing accidental complexity; that is, mismatches between what is being modelled and the modelling formalism being used. These criteria are: (1) Compactness, (2) Query Flexibility, (3) Heterogeneous Level-Hierarchies, and (4) Multiple Relationship-Abstractions. These criteria are important for modelling the domain in a concise, flexible, and simpler way. However, they do not cover certain aspects of particular importance to our domain and application in the OGI Pilot. We consider two additional criteria:

(a) ***Locality of Attributes & Relationships*** refers to what model elements attributes and relationships are defined on. Attributes/relationships are defined *locally* if they are defined on the model elements closest to where they are used. For example, an attribute relevant to product designs should be situated on the concept ProductModel rather than a related concept such as Product. This is in contrast to the modularity aspect, which attempts to minimise the different locations at which attributes and associations are located; however, it is particularly important in terms of interoperability as the different domain models exhibit different modelling approaches and scope, and attributes and associations may not have been grouped together consistently across the information system ecosystems. Having a flexible framework that can handle such a situation elegantly is important.
(b) ***Clarity of Relations' Semantics*** is concerned with whether the relations of the approach have clearly delineated semantics from other relations, or if they combine the semantics of multiple, commonly understood, relations together. For example, while a relationship that combines both the semantics of specialisation and instantiation may simplify the graphical representation of the model, the confounding of multiple relations in one could cause difficulties for constructing model transformations. Moreover, a number of issues can arise if the distinctions between relations are de-emphasised. Weakening the intrinsic differences between established relations comes at a significant cost such as "*sanity checks* regarding the integrity of metamodelling hierarchies that otherwise would not exist" [14].

In applying the criteria to our approach we conclude that it:

– is modular as it treats the class and object facets of a concept together allowing the specification of information regarding a concept in one place;
– allows redundancy-free modelling by using a range of relations that include various attribute propagation, inheritance, and assignment semantics;
– supports query flexibility through the different relationships and concepts in a domain model, e.g. different pump models can be accessed by retrieving the *instances* of PumpModel, the models in a particular category can be retrieved by accessing the *members* of the desired PumpCategory, and the physical pumps can be accessed through the instances of Pump;
– allows heterogeneous level-hierarchies through its flexible and dynamic nature of level stratification;
– supports specialisation and instantiation of domain and range of relationships;

While it is a balancing act to not produce too many relations, many of the relations in our approach are special cases of well known relations, alleviating possible issues with understandability and adding information of finer granularity. Moreover, existing domain models could be analysed to identify such distinctions, improving the identification of model transformations for interoperability.

There are three potency-based approaches in the comparison, the latter two of which are newly added to the comparison: Deep Instantiation [7], MetaDepth [15], and Dual-Deep Instantiation (DDI) [8]. MetaDepth allows the specification of different *views* (similar to the modelling spaces proposal of [2]), which allow for heterogeneous level-hierarchies. DDI extends DI with explicit levels based on “sort” hierarchies and attributes and associations with *2* potencies (source and target) rather than the single potency of DI and MetaDepth. DDI supports query flexibility (i.e. descendents of an object at a particular (sort) level can be queried in ConceptBase), heterogeneous level-hierarchies and multiple relationship-abstractions.

All three potency-based techniques can support locality of attributes and relations; however, it comes at the cost of not taking advantage of potency, i.e. restricting the possible potencies to zero or one. Similarly, under this restriction, standard DI and MetaDepth have clear cut relations, while using higher potency starts to mix instantiation with specialisation semantics. Finally, DDI more strongly combines the instantiation and specialisation relations and, hence, does not support the last criteria at all.

Assessing the power type approach (both simple and extended) with respect to the new criteria reveals that it supports locally specified attributes and relations. Whether the approach supports clarity of relation semantics is unclear as it depends on whether or not the partitions relation is provided with instantiation semantics [10]. However, the power type approaches only partly support compactness and only the extended power type approach fully supports query-flexibility while the simple approach only provides partial support. In terms of relationship abstraction, both approaches require OCL to provide this support.

The application of the additional criteria to m-objects show that the clarity of relation semantics is low, due to the combination of the relations aggregation, specialisation, and instantiation. In addition, as the intention of the M-Objects technique is to encapsulate all of the attributes and relationships of a concept on a single m-object, it does not support locally specified attributes and relationships.

5 Conclusion

Effective exchange of information about processes and industrial plants, their design, construction, operation, and maintenance requires sophisticated information modelling and exchange mechanisms that enable the transfer of semantically meaningful information between a vast pool of heterogeneous information systems. This need increases with the growing tendency for direct interaction of information systems from the sensor level to corporate boardroom level. One way to address this challenge is to provide more powerful means of information handling, including the definition of proper conceptual models for industry standards and their use in semantic information management. In this paper we have described our modelling framework for large scale ecosystem handling and the extended relationship types that help to succinctly express data models across a heterogeneous information system ecosystem.

References

1. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, March 1976.
2. C. Atkinson and T. Kühne. Processes and products in a multi-level metamodeling architecture. *IJSEKE*, 11:761–783, 2001.
3. ISO. *ISO 15926 – Part 2: Data Model.* 2003.
4. *Open Systems Architecture for Enterprise Application Integration.* MIMOSA, 2012.
5. B. Neumayr, M. Schrefl, and B. Thalheim. Modeling techniques for multi-level abstraction. In *The Evolution of Conceptual Modeling.* Springer LNCS 6520, 2011.
6. B. Henderson-Sellers, T. Clark, and C. Gonzalez-Perez. On the search for a level-agnostic modelling language. In *Proc. CAISE'13*, pages 240–255, 2013.
7. C. Atkinson and T. Kühne. The Essence of Multilevel Metamodeling. In *Proc. of UML 2001*, LNCS 2185, pages 19–33. Springer, 2001.
8. B. Neumayr, M. A. Jeusfeld, M. Schrefl, and C. Schütz. Dual deep instantiation and its ConceptBase implementation. In *Proc. CAISE '14*. Springer, 2014.
9. J. J. Odell. Power types. *JOOP*, 7:8–12, 1994.
10. C. Gonzalez-Perez and B. Henderson-Sellers. A powertype-based metamodelling framework. *Software & Systems Modeling*, 5(1):72–90, 2006.
11. B. Neumayr, K. Grün, and M. Schrefl. Multi-level domain modeling with m-objects and m-relationships. In *Proceedings APCCM '09*, pages 107–116, 2009.
12. Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994.
13. M. Schrefl and M. Stumptner. Behavior consistent specialization of object life cycles. *ACM TOSEM*, 11(1):92–148, 2002.
14. T. Kühne. Contrasting classification with generalisation. In *Proceedings APCCM '09*, pages 71–78, Australia, 2009.
15. J. de Lara, E. Guerra, R. Cobos, and J. Moreno-Llorena. Extending deep metamodelling for practical model-driven engineering. *Computer*, 57(1):36–58, 2014.