

A Conceptual Framework for Large-scale Ecosystem Interoperability^{*,**}

Matt Selway, Markus Stumptner, Wolfgang Mayer,
Andreas Jordan, Georg Grossmann, and Michael Schrefl

Advanced Computing Research Centre
University of South Australia
{andreas.jordan,matt.selway}@mymail.unisa.edu.au,
{mst,wolfgang.mayer,georg.grossmann}@cs.unisa.edu.au,
schrefl@dke.uni-linz.ac.at

Abstract. One of the most significant challenges in information system design is the constant and increasing need to establish interoperability between heterogeneous software systems at increasing scale. The automated translation of data between the data models and languages used by information ecosystems built around official or de facto standards is best addressed using model-driven engineering techniques, but requires handling both data and multiple levels of metadata within a single model. Standard modelling approaches are generally not built for this, compromising modelling outcomes. We establish the SLICER conceptual framework built on multilevel modelling principles and the differentiation of basic semantic relations that dynamically structure the model and can capture existing multilevel notions. Moreover, it provides a natural propagation of constraints over multiple levels of instantiation.

Keywords: Metamodelling, Conceptual models, multilevel modelling

1 Introduction

Lack of interoperability between computer systems remains one of the largest challenges of computer science and costs industry tens of billions of dollars each year [1,2]. Standards for data exchange have in general not solved the problem, as standards are not universal or universally applied even within a given industry, leading to *heterogeneous ecosystem* with large groups of software systems built around different standards that must interact to support the entire system lifecycle. We are currently engaged in the “Oil and Gas Interoperability Pilot” (or simply OGI Pilot) that aims for the automated, model-driven transformation of data during the asset lifecycle between two of the major data standards in the Oil & Gas industry ecosystem. The main standards considered by the project are the ISO15926 suite of standards [3] and the MIMOSA OSA-EAI specification [4].

^{*} The final publication is available at link.springer.com

^{**} DOI: 10.1007/978-3-319-25264-3_21

These standards and their corporate use¹ are representative of the interoperability problems faced in many industries today, and the cost of changing a code base and continuous change in the base data make conventionally programmed interoperability solutions too expensive to build and maintain. To enable sensor-to-boardroom reporting, the effort to establish and maintain interoperability solutions must be drastically reduced. This is achieved by developing model transformations based on high level conceptual models.

The core contribution of this paper is threefold: (1) We compare the suitability of different multi-level modelling approaches for the integration of ecosystems in the Oil & Gas industry, (2) provide the SLICER (Specification with Levels based on Instantiation, Categorisation, Extension and Refinement) relationship framework to overcome limitations of existing approaches, and (3) evaluate the framework on an extended version of the comparison criteria from [5].

2 Ecosystem Interoperability

The suite of standard use cases defined by the Open O&M Foundation covers the progress of an engineering part (or plant) through the Oil & Gas information ecosystem from initial specification through design, production, sales, deployment, and maintenance including round-trip information exchange. The data transformations needed for interoperability require complex mappings between models covering different lifecycle phases, at different levels of granularity, and incorporating data and (possibly multiple levels of) metadata within one model.

Notably, different concepts are considered primitive objects at different stages of the lifecycle. For example, during design the specification for a (type of) pump is considered an object that must be manipulated with its own lifecycle (e.g. creation, revision, obsolescence), while during operations the same object is considered a type with respect to the physical pumps that conform to it and have their own lifecycle (e.g. manufacturing, operation, end-of-life). Furthermore, at the business/organisational level, other concepts represent categories that perform cross-classifications of objects at other levels. This leads to an apparent three levels of (application) data: business level, specification level, and physical entity level. To describe these different levels multi-level modelling (MLM) approaches to model-driven engineering seem a natural fit. Ideally, a flexible conceptual framework should represent the entire system lifecycle, in a way that simplifies mapping between disparate models by the interoperability designer.

The ecosystem transformations use a joint metamodel that serves as the common representation of the information transferred across the ecosystem (cf. Figure 1) and must be able to handle the MLM aspects. As pointed out in [6], such complex domains generally are not dealt with using the classical GAV or LAV (Global/Local As View) querying approach, but require a more general form of mapping describing complex data transformations. Notably, the data

¹ Current industrial participants in the OGI Pilot include: Intergraph, Bentley, AVEVA, Worley-Parsons, for ISO15926; IBM, Rockwell Automation, Assetricity for MIMOSA; various Oil & Gas or power companies as potential end users.

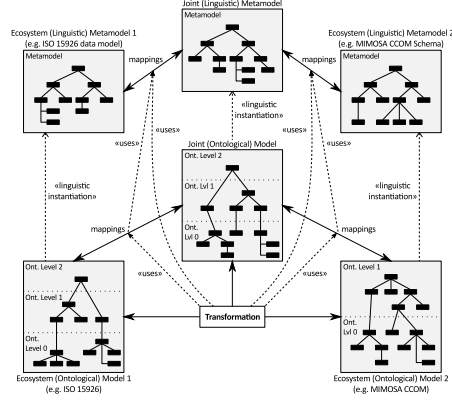


Fig. 1: Ecosystem interoperability through a joint metamodel

integration systems surveyed in [6] generally use languages that do not have MLM or even metamodeling capabilities, and the automated matching capability of the systems listed (e.g., MOMIS, CLIO) is probabilistic. As transformations in the engineering domain must guarantee correctness (e.g., an incorrectly identified part or part type can result in plant failures), heuristic matching cannot replace the need for a succinct and expressive conceptual model for designing the mappings.

A UML-based approach to modelling this situation is shown in Figure 2, in which `ProductCatalogue`, `ProductCategory`, `ProductModel`, and `ProductPhysicalEntity` are explicitly modelled as an abstraction hierarchy using aggregation in an attempt to capture the multi-level nature of the domain². Specialisation is used to distinguish the different categories (i.e. business classifications), models (i.e. designs), and physical entities of pumps. Finally, instantiation of singleton classes is used to model the actual catalogue, categories, models, and physical entities.

From both a conceptual modelling and interoperability perspective, this is unsatisfactory: it is heavily redundant [7,5]; the misuse of the aggregation relationship to represent a membership and/or classification relation results in physical entities that are not intuitively instances of their product models; last it creates difficulty in modelling the lifecycles of both design and physical entities as well as the dynamic introduction of new business categories. This directly affects mapping design as the real semantics of the model are hidden in implementation.

3 Multi-Level Modelling Techniques

A number of Multi-level modelling (MLM) techniques have been developed to address the shortcomings of the UML-based model. While they improve on the UML-based model in various respects, no current approach fulfils all of the criteria necessary for ecosystem interoperability (see Section 4.1).

² While UML 2 supports power types in limited fashion, this example is restricted to more basic UML constructs. Power types are discussed in Section 3.

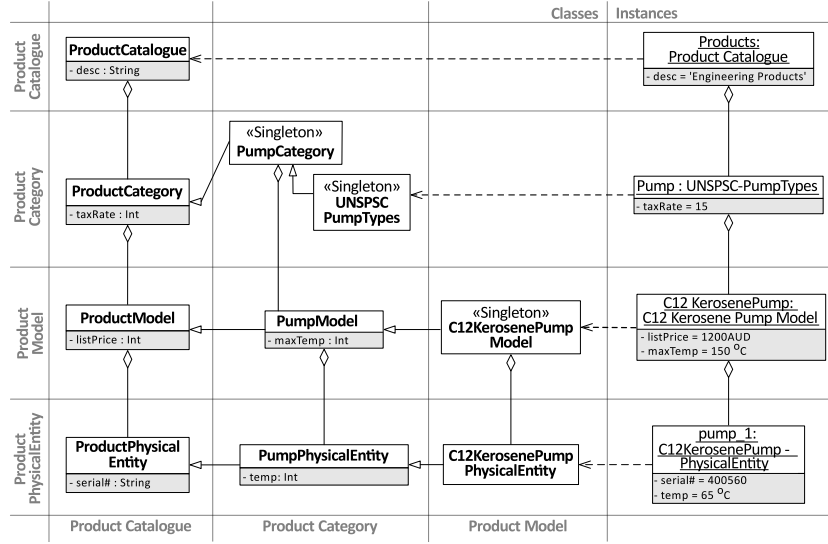


Fig. 2: Product catalogue modelled in plain UML (adapted from [5])

Potency-based MLM Techniques [7] were originally introduced for *Deep Instantiation (DI)* to support the transfer of information across more than one level of instantiation. They separate *ontological* instantiation, which are domain specific instantiation relationships which can be cascaded multiple times, from standard *linguistic* instantiation used in UML meta-modelling. Each model element (e.g. class, object, attribute, or association) has an associated potency value that defines how many times the model element can be ontologically instantiated. That is, a class with potency 2 can have instances of instances, while an attribute with potency 2 can be transferred across two instantiation levels. Modelling elements with potency 0 cannot be further instantiated. For example, the class Pump Model in Figure 3a defines two attributes: $temp^2$ and $maxTemp^1$. Both attributes have the flat data-type integer as range (representing a value in degrees Celsius); however, $maxTemp^1$ is instantiated only once due to its potency of 1, while $temp^2$ is instantiated twice and only receives its complete concrete value at the *O0* level, with the assigned value at *O1* interpreted as a default value for its instances.

In a more complete model, the **ProductCategory** concept would result in an additional level of instantiation. To model the domain appropriately, this approach invariably results in relationships (other than instantiation) crossing level boundaries, which is not allowed under *strict* meta-modelling typically adhered to by potency-based models. For example, 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 [9]. Moreover, the concept **Pump** is in violation of strict meta-modelling as it does not

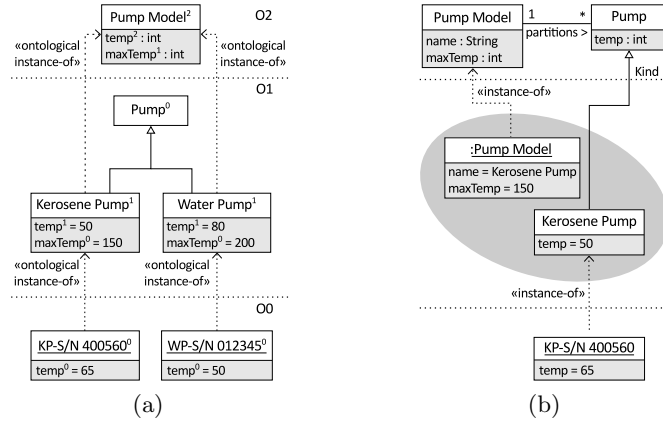


Fig. 3: Extracts of the Deep Instantiation (a) and Power Type (b) based models of the product catalogue example. Superscripts represent *potency*; the ellipse links the type and instance facets of a concept following the notation of [8].

(ontologically) instantiate anything from the level above. An alternative would be to remove **Pump** altogether; however, this would cause a conceptual mismatch with the domain and lead to more complex mappings to models including such a concept. Finally, DI partially conflates specialisation and instantiation semantics through the use of potency, complicating model transformations.

Power types [10] have also been applied in a multi-level context, e.g., [8]. Basically, for a power type t of another type u , instances of t must be subtypes of u . Figure 3b shows an example of the Power Type pattern, where **Pump Model** is the power type for the concept **Pump**; the concept **ProductCategory** would be represented as cascading uses of the power type pattern. As discussed in [5], this leads to complex and redundant modelling, which complicates creation and maintenance of interoperability mappings. However, power typing does not conform to strict meta-modelling with non-instantiation relationships crossing level boundaries, which occurs no matter where you try to relocate the concepts. Giving the *partitions* relation instantiation semantics, as argued in [8], to allow it to cross the level boundary leads to the counter-intuitive notion that the type is a partition of itself. Finally, separating the type facet from the object facet leads to redundancy, when the two facets are really the *same object* [10].

M-Objects and M-Relationships (M standing for multi-level) [11] use a *concretization* relation which stratifies objects and relationships into multiple levels of abstraction within a single hierarchy. 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 include the different **PumpCategories**, **PumpModels**, and **PhysicalPumps**, respectively. While the M-Objects technique produces concise models with a minimum number of relations, the concretization relation between two m-objects (or two m-relationships) must be interpreted in a multi-faceted way (since it

represents specialisation, instantiation, and aggregation), increasing the difficulty of identifying mappings between models.

4 A relationship framework for ecosystem modelling

A highly expressive and flexible approach is required to overcome the challenges involved in modelling large ecosystems and supporting transformations across their lifecycles. A key requirement is the identification of *patterns of meaning* from basic primitive relations that can be identified across modelling frameworks and assist the development of mappings between them. A core observation when building transformations for the real world complexity of the OGI pilot is that a higher level in the model, whether ontological or linguistic, expresses the relationship between an entity and its definition (or description) in two possible ways: abstraction and specification. In *abstraction*, entities are grouped together based on similar properties, giving rise to a concept that describes the group—the entities may be groups of concepts, thus permitting a multi-level hierarchy. With *specification*, an explicit description is laid down, and entities (artefacts) are produced that conform to this specification[12,13]. This is a common scenario in the use of information systems and plays a major role in the framework.

The common factor is that a *level of description* (and therefore the consistency constraints for a formalism that expresses these principles) is not purely driven by instantiation. A different level is also established by enriching the vocabulary used to formulate the descriptions. This corresponds to the concept of *extension* in specialisation hierarchies [14]: if a subclass receives additional properties (attributes, associations etc.) then these attributes can be used to impose constraints on its specification and behaviour. Identifying levels based on the basic semantic relationships between entities enables a flexible framework for describing joint metamodels in interoperability scenarios. Moreover, a domain modeller need not utilise the primitive semantic relations directly; rather, they are identifiable in the conceptual modelling framework of a particular model, supporting the development of mappings between models of different frameworks.

In contrast, most of the MLM approaches summarised in Section 3 focus on the pre-layering of levels as the main designer input, a corollary from the natural rigidity of the level system in linguistic instantiation as intended by UML. However, it is generally assumed by prior specialized work on relationships such as specialization [14], metaclasses [15], materialization, and aggregation [16] that there can be arbitrary many levels of each. Therefore, interoperability solutions must accommodate mappings to models that are not designed according to the strictness criterion, may cover domains at different levels of granularity, and cannot (from the view of the interoperability designer) be adapted.

To support this we present the SLICER (Specification with Levels based on Instantiation, Categorisation, Extension and Refinement) framework based on a flexible notion of levels as a result of applying specific semantic relationships. In the following description of the relations forming the SLICER framework we refer to Figure 4, which shows the application of the framework to the product catalogue

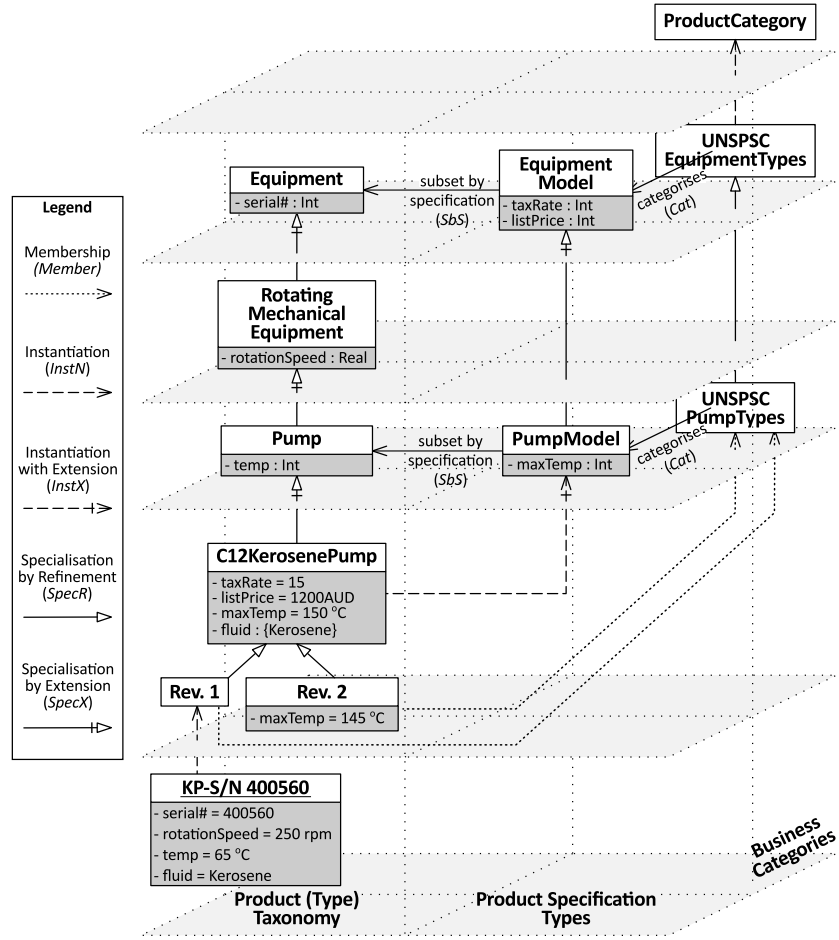


Fig. 4: Product catalogue example modelled using SLICER relationships

example in the context of the OGI Pilot. The diagram exemplifies the finer distinctions made by the SLICER framework including: the increase of detail or specificity (from top to bottom) through the addition of characteristics, behaviour, and/or constraints that are modelled; the explicit identification of characteristics for describing models of equipment (i.e. specifications) themselves, not only the physical entities; the second level of classification through the identification of categories and the objects that they categorise; and the orthogonal concerns of the different stakeholders and stages of the lifecycle (shown by separation into 3 dimensions).

Instantiation and Specialisation: Like other frameworks, SLICER uses instantiation and specialisation as the core relations for defining hierarchies of concepts based on increasing specificity (or decreasing abstraction). In contrast to previous MLM techniques, the levels are not strictly specified, but dynamically

derived based on finer distinctions of relations between more or less specific types. We characterise specialisation relationships along the lines of [14] to distinguish a relationship that *extends* the original class (by adding attributes, associations, or behaviour, i.e., constraints or state change differentiation) or *refines* it (by adding granularity to the description). The latter generally means restricting the range of the attribute or association. In the extreme, it can be seen as restricting an attribute to a singleton domain, i.e., pre-specifying the value upon instantiation.

We identify a specialisation relationship that extends the original class as *Specialisation by Extension* (*SpecX* for short) which adopts standard monotonic specialisation semantics. While it can include (but does not necessitate) *refinement*, it is distinguished based on incorporation of additional attributes. Most importantly, *SpecX* introduces a new model level. For example, in Figure 4, the conceptual entity **Pump** is a specialisation by extension of **RotatingMechanicalEquipment**. This is due to the addition of *temp*.

In contrast, *Specialisation by Refinement* (denoted *SpecR*, only *refines* the original class. Thereby supporting subtypes that restrict the extension of a class without introducing additional model levels. For example, the two revisions of the **C12KerosenePump** design (an important distinction in the example domain) only refine the attributes and constraints of **C12KerosenePump** and are therefore specialisations by refinement.

Similarly, instantiation is characterised as *Instantiation with Extension* (*InstX*), or *Standard Instantiation* (*InstN*). Instantiation always introduces additional model levels and all attributes of the instantiated type have values assigned from their domain. However, *InstX* allows additional attributes, behaviour, etc. to be added that can then be instantiated or inherited further, while *InstN* does not. As such, objects that are the result of *InstN* cannot have instances themselves. The pump model **C12KerosenePump** demonstrates *InstX* as it introduces characteristics specific to that type of pump (e.g. *fluid* with a singleton domain), while **KP-S/N 400560**, representing a physical pump, demonstrates *InstN*.

Categories are concepts that provide external (“secondary”) grouping of entities based on some common property and/or explicit enumeration of its members. In the SLICER framework, we explicitly represent categories through two relationships: *Categorisation* (*Cat*) and *Membership* (*Member*). *Categorisation* relates two concepts, one representing a category and the other a type, where the *members* of the category are *instances* of the type. While *Membership* resembles instantiation (the two are mutually exclusive), it does not place any constraints on the assignment of values to attributes (the category and its members can have completely different sets of attributes). However, this does not preclude the specification of membership criteria, or constraints, for allowing or disallowing the possible members of a category. In addition, categories exist on the same level as the type they categorise. This is intuitive from the notion that their membership criteria (if any) are defined based on the type they categorise. Moreover, sub-categories can be specified through *SpecR* only, supporting refinement of the membership constraint. For example, the concepts **UNSPSCEquipmentTypes** and **UNSPSCPumpTypes** are a pair of categories indicating that their members are equip-

ment models (or specifically pump models for the subcategory) conforming to the UNSPSC standard. The two revisions of `C12KerosenePump` are both *members* of `UNSPSCPumpTypes` (and hence are members of `UNSPSCEquipmentTypes`).

Specifications are the second means of relating an entity to its description, discussed earlier. For this we introduce the *Subset by Specification (SbS)* relation to identify specification types and the parent type of the specification concepts. The specification class (for example `EquipmentModel`) exists at the same level as the type it refers to as it can define constraints with respect to that type. However, subtypes of the specification type can be defined to reference particular properties; so `EquipmentModel` can be specialised to refer to properties of `Pumps`. Together with *InstX*, this relationship can be used to construct the powertype pattern [10].

In the presence of multiple specifications types for a concept, SLICER supports multiple instantiation; that is, an object can be an instance of multiple concepts as long as the concepts are instances of specification types for the same concept. For example, if a second specification type were related to `Pump` then KP-S/N 400560 could instantiate specifications related to both types. This form of multiple partitioning of a type [10] is not supported in standard meta-modelling or multi-level modelling approaches that allow only single instantiation.

Descriptions and Constraints: The final SLICER component is the explicit handling of object descriptions or constraints through the different relations introduced above. Basically, a description refers only to the attributes specific to its object, can be inherited through specialisation and instances of a type (and members of a category) must satisfy its description (or membership criterion). More important, however, is the handling of complex situations requiring the propagation of constraints across multiple instantiation relations. For example, the specification type `PumpModel` may define a constraint involving a relation between *maxTemp* (defined by itself) and *temp* (defined by `Pump`). In this situation the constraint is not applicable to direct instances of `PumpModel` as they do not assign a value to *temp*; instead, the constraint applies to KP-S/N 400560 two instantiations away. We handle this in a natural and intuitive way. Basically, we propagate the part of the constraint that is not applicable at a certain level across instantiation relations until it reaches an object for which it can be evaluated.

Due to lack of space to present the full formal framework, Table 1 summarises the properties of the relations described in this section.

4.1 Evaluation and Comparison

The comparison of MLM approaches in [5] provides the criteria for making domain models more concise, flexible, and simple, but does not cover certain aspects important for defining joint metamodels in an interoperability scenario as in the OGI Pilot³. We therefore extend the comparison by additional criteria (#5-7), as summarized in Table 2:

³ Except for criterion (2), which is necessary for multiple classification hierarchies.

Table 1: Summary of Relation Properties

	<i>SpecX</i>	<i>SpecR</i>	<i>InstX</i>	<i>InstN</i>	<i>Cat</i>	<i>Member</i>	<i>SbS</i>
Refinement	x	x					
New attributes	x		x				
Must cross 1 level	x		x	x			
Can cross levels		x					
Assign values to attrs.			x	x			
Inst.can have inst.	x	x	x		x	x	x
Cat.-Type Relation					x		
Propagates constraints	x	x	x	x		x	x
Base type attr. access					x	x	x
Used f. Powertype			x				x

1. **Compactness** encompasses *modularity* (all aspects related to a domain concept can be treated as a unit⁴) and *absence of redundancy*. In the ecosystem context, modularity as in [5] is too restrictive as disparate models may group concepts differently. Therefore, we use the *locality* criterion instead (see 5).
2. **Query Flexibility** means that queries can be performed to access the model elements at the different levels of abstraction (e.g. querying for all of the product categories, models, physical entities, and their specialisations).
3. **Heterogeneous Level-Hierarchies** allow an approach to introduce “new” levels of abstraction for one (sub-)hierarchy without affecting another (e.g. introducing a *PumpSeries* level of abstraction in between *PumpModel* and *PumpPhysicalEntity* without causing changes to any other hierarchy).
4. **Multiple Relationship-Abstractions** include classification of relationships, querying and specialisation of relationships (and their classes and meta-classes).
5. **Locality** is supported by an approach that can define attributes, relations, and constraints on the model element closest to where they are used (e.g. an attribute most relevant to models or designs themselves should be situated on the concept *ProductModel* rather than some related concept such as *Product*).
6. **Decoupling of Relationship Semantics** if they have clearly delineated semantics from one another rather than combining the semantics of multiple, commonly understood relations. Violating this makes it much harder to perform “sanity checks regarding the integrity of metamodeling hierarchies” [17].
7. **Multiple Categorisation** considers whether or not the modelling approach can support an element being placed in multiple (disjoint) categories. Since an item could simultaneously belong to multiple categories, it is particularly important for business level classification. For example, a pump model being certified by two different standards would place it in the categories for both.

With regard to these criteria, the SLICER approach:

1. allows redundancy-free modelling using a range of relations that include various attribute propagation, inheritance, and assignment semantics;
2. supports query flexibility across relationships and concepts in a domain model;
3. allows heterogeneous level-hierarchies through specific primitive semantic relations and its flexible, dynamic approach to level stratification;

⁴ E.g. Product category, model, and physical entity are grouped together.

Table 2: Summary of comparison (extended and adapted from [5])

Approach/Criterion #	1 [†]	2	3	4	5	6	7
UML	–	+	$\sim/+^1$	$-\sim^1$	+	\sim	+
Deep Instantiation [7]	+	+	–	\sim	\sim	\sim	\sim
MetaDepth [18]	+	+	+	\sim	\sim	\sim	\sim
Dual-Deep Instantiation [9,19]	+	+	+	+	\sim	–	–
Powertypes (Simple) [8]	–	\sim	–	$-\sim^1$	+	\sim	\sim
Powertypes (Ext.)	–	+	+	$-\sim^1$	+	\sim	\sim
Powertype (Onto.) [20]	–	\sim	+	–	+	+	–
Materialization [16]	+	\sim	–	\sim	–	\sim	–
M-Objects [11]	+	+	+	+	–	–	–
SLICER	+	+	+	\sim	+	+	+

Legend: (+) Full Support, (\sim) Partial Support, (–) No Support

[†] redundancy-free only; ¹ using OCL; ² specialization schema; ³ overlay schema

4. supports specialisation and instantiation of domain and range relationships, which constitutes partial support of multiple relationship extractions; however, (meta-)classification of relationships may not be required due to the finer semantic distinctions provided by SLICER ;
5. supports locally specified attributes and relations by design;
6. clearly differentiates a number of important, yet distinct, primitive relations that exist between domain concepts; and,
7. supports multiple categorisations through explicit representation of categories for which objects can belong to multiple concurrently.

Mapping Example Using the semantic distinctions described in the previous section, other models of the ecosystem can be analysed to better identify mappings between them. This requires relating the distinctions made by SLICER to other modelling approaches as without them only partial alignment between models can be found (compare Figures 3 & 4). For brevity, we only discuss the more interesting case of potency-based models due to the greater mismatch (e.g. *Pump* is at a different level and the attributes are located on different concepts).

The distinctions between extension, refinement, categories, and specifications that are made in SLICER are apparent in the potency-based model through:

1. An attribute with potency ≥ 2 suggests *InstX* as the potency indicates that the attribute should be introduced to the concept at the level where its potency = 1 (so that it can be given a value at the next instantiation).
2. A subclass that introduces new attributes over its parent class suggests *SpecX*, while a subclass that only refines the domain of an attribute suggests *SpecR*; this is a direct corollary of the same distinction made in SLICER .
3. An object with potency = 0 and attributes with only potencies = 0 is *InstN*.
4. If all of the instances of an object are specialisations of the same class it suggests an *SbS* relation between the instantiated class and the specialised class. In particular, if the class has attributes with potency > 0 or the instantiated class has attributes with potency = 2 it is most likely the case.

5. Specialised classes that have no attributes with a potency ≥ 1 and that are not identified as related to a specification class (see previous rule) suggests possible use of specialisation for categorisation. The introduction of attributes indicates the boundary between categorisation and categorised object(s).

Applying these rules to the example we can identify *InstN* by rule (3) between the object at *O0* on the left and *KerosenePump*, hence it can be aligned with the bottom level of the joint metamodel. At the *O1* level, *KerosenePump* instantiates *PumpModel* from the level above which includes an attribute with potency 2; thereby identifying *InstX* by rule (1). Moreover, by rule (2) a *SpecX* relation is identified as *KerosenePump* adds attributes over the class it specialises (*Pump*). Finally, all of the instances of *PumpModel* are subclasses of *Pump*, matching rule (4) and indicating the relation *SbS*(*PumpModel*, *Pump*). As a result, this pattern can be matched in the joint metamodel. Note that the result would be the same if the attribute *temp*² were defined on *Pump* as *temp*¹. Finally, if the concept *Pump* were not modelled at all (i.e. it was completely incorporated into *PumpModel*) then the specialisation to some concept could still be inferred due to the combination of specialisation and instantiation embodied by potency.

Conceptual Complexity and Practical Use At first glance, the incorporation of the above described relationship patterns would appear to increase the complexity of the modeler’s task, by increasing the number of basic relationships considered in a model. However, this merely reflects the difficulty of the task. The core data models of the two standards used for the OGI Pilot experiments involve hundreds of classes with thousands of attributes that can be extended further by individual providers.

First, the new relationships in SLICER were developed by identifying special cases that are known to cause problems in practice, leading to wrapping modelling concerns into generic domain associations, naming conventions, or even both, resulting in highly complex and often inconsistent models [21]. The goal of this approach is to enable development of coherent models in domain areas where persistent and large scale effort has *failed* to produce workable models since the modelling approaches used did not permit expressing the salient features of the domain. This state of affairs is confirmed by the rise of interest in MLM methods even outside the interoperability application domain. Though these methods have been largely developed in the software engineering community and are typically based on UML, they generally revolve around conceptual domain models rather than detailed design models.

A key factor was the identification of the extension and specification relationships as distinguishing factors that needed to be explicitly represented. Separating out these two types of relationships, they can be considered as *meta-properties* that annotate or enrich the families of relationships existing in conventional models. This can be seen as an extension of the classical U, I, and R meta-properties used in OntoClean [22]. In the case of the X (extension) meta-property, this can be partly automatically identified by examining property specifications. In the S (specification) case, it would be used to identify what would before have been a generic domain association, or at best an instance of the generic

D (Dependency) meta-property. This indicates an avenue towards automated support for consistent use of these relationships.

Also, given that the need for some sort of MLM capability has been widely recognised, the standard for a complexity comparison would not be conventional modelling methods, but the other MLM methods. Compared to standard potency-based methods (e.g., MetaDepth), a SLICER model employs mostly the standard relationships found in conceptual models, with limited extensions that carry domain semantics that any modeller would be familiar with. In comparison, assigning potency value requires thinking in terms of multiple metalevel instantiations, and potentially restricts instantiations at lower levels. This raises the question whether a potency-based designer can predict the implications of his modelling choice for all later additions to the model. Compared to Deep Instantiation and Dual Deep Instantiation, SLICER supports designer-oriented terminology and a clear separation of semantic concerns. Powertypes are based on implicit constraints that cannot be expressed in standard conceptual models, and require observing implicit dependencies between subtype partitions and powertype membership. The M-Objects and Materialisation relationship both conflate multiple different semantic relations into a single relationship (called concretization in the former and materialisation in the latter). This actually makes the task harder for an analyst since the model does not help him to keep track of the semantic distinctions that gave rise to the different treatment in the domain in the first place.

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 and transformation.

In this paper we have described the SLICER framework for large scale ecosystem handling. It is based on the notion that a model-driven framework for creating mappings between models of an ecosystem must identify the underlying semantics of a model based on the relationship between an entity and its description, most clearly embodied by artefacts subject to their specifications. This led to the introduction of a set of primitive relationship types not so far separately identified in the literature. For one, they reflecting the fundamental conceptual modelling distinction between extension and refinement [14], thus allowing to succinctly express distinct semantics encountered in multilevel and multi-classification modelling scenarios. On the other hand, they represent the first time elevation of the *specification* relationship to full “citizenship” status among the relationships in

the conceptual model. Together these relationships capture underlying design assumptions encountered in various potency and powertype approaches and enable their separate examination and study. This allows to express the complex mapping of large scale interoperability tasks in a consistent and coherent manner. The identification of extension and specification/description as ontologically relevant meta-properties offers as a natural next step the connection to our work on artifact and specification ontologies [12]. Future work includes the extension of the underlying formal framework and the investigation of the above mentioned connections to formal ontological analysis.

References

1. N. Young and S. Jones. SmartMarket Report: Interoperability in Construction Industry. Technical report, McGraw Hill, 2007.
2. Fiatch. Advancing Interoperability for the Capital Projects Industry: A Vision Paper. Technical report, Fiatch, February 2012.
3. ISO. *ISO 15926 – Part 2: Data Model*. 2003.
4. MIMOSA. *Open Systems Architecture for Enterprise Application Integration*. 2014.
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. S. Bergamaschi, D. Beneventano, F. Guerra, and M. Orsini. Data integration. In *Handbook of Conceptual Modeling*, pages 441–476. Springer, 2011.
7. C. Atkinson and T. Kühne. The Essence of Multilevel Metamodeling. In *Proc. of UML 2001*, LNCS 2185, pages 19–33. Springer, 2001.
8. C. Gonzalez-Perez and B. Henderson-Sellers. A powertype-based metamodeling framework. *Software & Systems Modeling*, 5(1):72–90, 2006.
9. B. Neumayr, M. A. Jeusfeld, M. Schrefl, and C. Schütz. Dual deep instantiation and its ConceptBase implementation. In *Proc. CAISE '14*, pages 503–517. 2014.
10. J. J. Odell. Power types. *JOOP*, 7:8–12, 1994.
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. A. Jordan, M. Selway, W. Mayer, G. Grossmann, and M. Stumptner. An ontological core for conformance checking in the engineering life-cycle. In *Proc. Formal Ontology in Information Systems (FOIS 2014)*. IOS Press, 2014.
13. S. Borgo, M. Franssen, P. Garbacz, Y. Kitamura, R. Mizoguchi, and P.E. Vermaas. Technical artifact: An integrated perspective. In *FOMI 2011*. IOS Press, 2011.
14. M. Schrefl and M. Stumptner. Behavior consistent specialization of object life cycles. *ACM TOSEM*, 11(1):92–148, 2002.
15. W. Klas and M. Schrefl. *Metaclasses and Their Application*. LNCS 943. 1995.
16. R. C. Goldstein and V. C. Storey. Materialization. *IEEE Trans. on Knowl. and Data Eng.*, 6(5):835–842, Oct. 1994.
17. T. Kühne. Contrasting classification with generalisation. In *Proceedings APCCM '09*, pages 71–78, Australia, 2009.
18. J. de Lara, E. Guerra, R. Cobos, and J. Moreno-Llorena. Extending deep meta-modelling for practical model-driven engineering. *Computer*, 57(1):36–58, 2014.
19. B. Neumayr and M. Schrefl. Abstract vs concrete clabjects in dual deep instantiation. In *Proc. MULTI'14 Workshop*, pages 3–12, 2014.

20. O. Eriksson, B. Henderson-Sellers, and P. J. Ágerfalk. Ontological and linguistic metamodeling revisited: A language use approach. *Information and Software Technology*, 55(12):2099–2124, 2013.
21. Barry Smith. Against idiosyncrasy in ontology development. In *Proc. Formal Ontology in Information Systems (FOIS 2006)*, pages 15–26, 2006.
22. Christopher A. Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. *Data Knowl. Eng.*, 39(1):51–74, 2001.