

Multilevel Mapping of Ecosystem Descriptions

Short Paper ^{*,**}

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

University of South Australia, Australia,
{matt.selway, andreas.jordan}@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. Beyond individual applications, today’s enterprise applications require automated information exchange across the system lifecycle of information ecosystems—large scale families of software built around official or de facto standards. The automated translation of data between the data models and languages used in these ecosystems is best addressed using model-driven engineering techniques, but requires the handling of both data and multiple levels of metadata within a single model. Standard modelling approaches are generally inconsistent with these requirements, leading to compromised modelling outcomes. In this paper we discuss the use of the SLICER framework built on multilevel modelling principles for transformation purposes. The framework provides natural propagation of constraints over multiple design and instantiation levels that cover different engineering lifecycle phases. We discuss the concept of metamodelling spaces and give an example of a concrete transformation application.

Keywords: Metamodelling, Lifecycle models, interoperability

1 Ecosystem Interoperability

The 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]. Standards usually are not universal nor universally applied even within a given industry. The engineering industry refers to the set of software systems that interact to provide services across the entire system lifecycle of their domain as a *heterogeneous ecosystem*. We are 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: ISO15926 [2]

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

^{**} DOI: 10.1007/978-3-319-26148-5_15

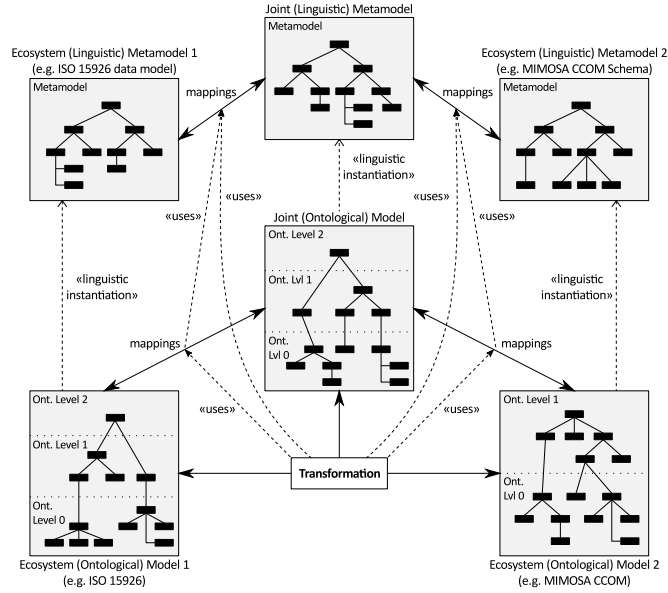


Fig. 1: Ecosystem interoperability through a joint metamodel

generally used for EPC (Engineering, Procurement, and Construction), and the MIMOSA OSA-EAI for Operations & Maintenance.¹

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 multiple levels of (application) data: business, specification, and physical entity level. Enabling ecosystem transformation requires a flexible conceptual framework for representing these levels of information. 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 does not replace the need for a succinct and expressive conceptual model that facilitates design of the mappings by a human designer. We use a framework for ecosystem transformations based on a joint metamodel that serves as the baseline for the transferred information [3].

When trying to model ecosystem mappings in UML, aggregation is normally used in an attempt to capture the multi-level nature of the domain, and specialisation is used to distinguish the different categories (i.e. business classifications), models (i.e. designs), and physical entities of pumps. Finally, instantiation of

¹ Current industrial participants in the OGI Pilot include: Intergraph, Bentley, AVEVA, Worley-Parsons, IBM, Rockwell Automation, Assetricity.

singleton classes is used to model the actual catalogue, categories, models, and physical entities. This creates redundancy [4], physical entities that are not intuitively instances of their product models, and difficulty in modelling the lifecycles of both design and physical entities as well as the dynamic introduction of new business categories. This makes the definition of mappings more difficult as the real semantics of the model are hidden in implementation.

Multilevel Modelling (MLM) approaches [4,5,6] have been developed to address such situations, but in general use the concept of *potency*: a numeric value that requires pre-layering of levels as the main designer input, thus setting down a fixed number of levels from the start. In contrast, prior work on relationships such as specialization [7], and metaclasses [8] assumed that there can be arbitrary many levels of each. Therefore, interoperability solutions must accommodate mappings to models that may cover domains at different levels of granularity, are not designed according to the strictness criterion (i.e. only instantiation relationships may cross level boundaries), and cannot be changed to fit the needs of the interoperability designer. Transformation solutions such as [3] will therefore profit from a more flexible level definition framework.

In the context of enterprise integration frameworks, our work is situated in the space of Model-Driven Interoperability as described in [9]. Much of this work focused on Universal Enterprise Modeling Languages (UEML) [10]. Our approach assumes that in a setting of heterogeneous ecosystems, it is via effective transformation languages and meta-models where flexibility can be achieved.

2 The SLICER relationship framework

When examining different description levels in the engineering lifecycle, a higher level generally 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. These entities may themselves be groups of concepts, thus permitting a multi-level hierarchy. With *specification*, an explicit description is given, and entities (artefacts) are produced that conform to this specification.

A *level of description* can be established either by instantiation, or by enriching the vocabulary used to formulate the descriptions. This corresponds to the concept of *extension* in specialisation hierarchies [7]: 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. To support this we have developed SLICER (Specification with Levels based on Instantiation, Categorisation, Extension and Refinement) framework based on a flexible notion of levels as a result of applying the semantic relationships below. A detailed presentation of the conceptual framework and comparison to other multi-level modelling approaches was given in [11]. Figure 2 shows the application of the framework to a product catalogue example.

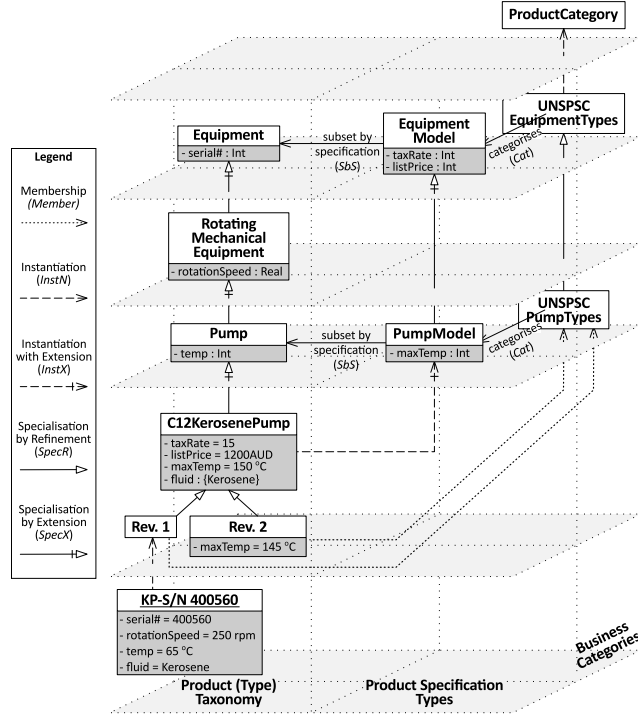


Fig. 2: Product catalogue example modelled using our framework

Instantiation and Specialisation In contrast to previous MLM techniques, levels are not strictly specified, but dynamically derived based on finer distinctions of relations between more or less specific types. Specialisation relationships *extend* the original class (by adding attributes, associations, or behaviour, i.e., constraints or state change differentiation, *SpecX*) or *refines* it (by adding granularity to the description, *SpecR*). Of the two, only *SpecX* introduces a new model level.

Similarly, instantiation is characterised as *Instantiation with Extension (InstX)*, allowing additional attributes) or *Standard Instantiation (InstN)*. Instantiation always introduces additional model levels. A key difference between the two is that an object created through *InstN* cannot be instantiated further.

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)*, which we do not discuss further). *Categorisation* relates two concepts, one representing a category and the other a type, where the *members* of the category are *instances* of the type. Categories exist on the same level as the type they categorise.

Specifications are expressed via the *Subset by Specification (SbS)* relation, which identifies specification types and the parent type of the specification

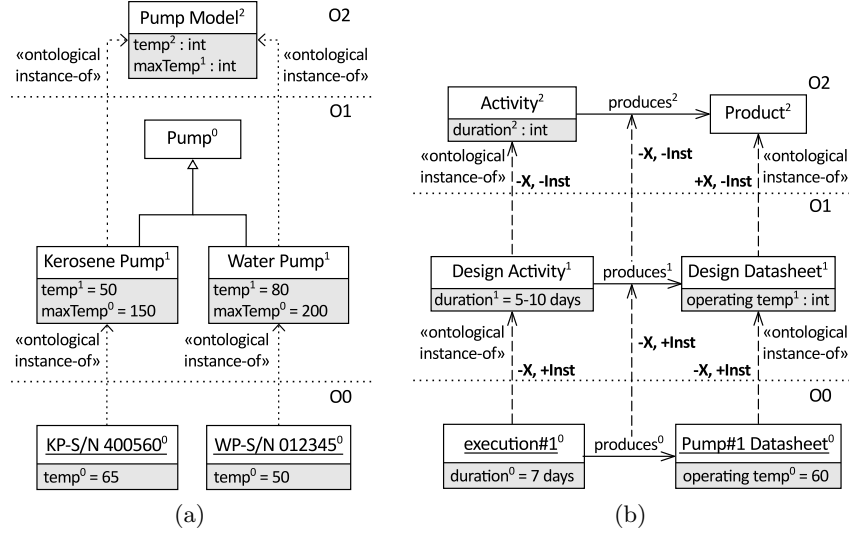


Fig. 3: (a) Potency-based model of the product catalogue example. (b) Meta-relationship properties being used to identify whether a relationship should be specialisation or instantiation.

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. **Descriptions** A description, e.g. a set of constraints, can refer only to the attributes specific to its object (or, if the description is for a category or specification, the attributes of the type associated with the category or specification) and are inherited through specialisation, while instances of a type (and members of a category) must satisfy its description (or membership criterion).

3 Lifecycle management

A Level Alignment Example Using the semantic distinctions described in the previous section, other models in an ecosystem can be analysed to better identify mappings between them. We use potency-based models for the example (cf. Figures 2 & 3a), identifying the following alignment indicators:

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.

Meta-relationship properties 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, in practice it can be seen that the new relationships are really created by identifying special cases that would already have been reflected in domain models, but have usually been left implicit and hidden within generic domain associations or naming conventions (or even both, as in ISO 15926), resulting in highly complex and even inconsistent models [12]. In particular, these special cases result from the use of extension and explicit specifications. These can be considered as *meta-properties* (X+/-, S+/-) of the existing relationships, in the sense of the U, I, and R meta-properties used in OntoClean [13]. In the case of the X (extension) meta-property these can be partly automatically identified by examining property specifications. In the S (specification) case, it would be used to identify and distinguish what would before have been a generic domain association, or at best an instance of the generic D (Dependency) meta-property.

Figure 3b displays the application of the meta-properties to identify specialisation vs. instantiation. As can be seen, the bottom-most “instantiation” relationships have all been marked as *not* including extension, but incorporate instantiation as all attributes have been assigned values. Therefore, instantiation (*InstN*) is the correct relationship. In the top row, however, we can see that no instantiation occurs—the assignment of 5-10 days in *Design Activity* can be considered refinement as the “value” is a range and the potency remains > 0 —and there is a mix of extension and not extension. Therefore, the correct relationships would be *SpecR* between *Design Activity* and *Activity*, and *SpecX* between *Design Datasheet* and *Product*. In practice, using these properties to automatically identify the

correct relationships helps to reveal similarities between different (meta-)models when engineering transformations between them.

Metamodelling Spaces were an attempt to resolve the issue of strict meta-modelling by separating modelling domains such that they have their own set of metamodel levels within their own space, irrespective of the levels of other metamodelling spaces [14]. A hierarchy of metamodelling spaces can then be formed by one way dependencies between spaces such that the source meta-modelling space is on a metalevel with respect to the target space, thereby allowing the source space to reference elements on any level of the target space without violating the strictness criterion. This ensures *global strictness* of the model by eliminating cycles between meta-levels. A key aspect of using metamodelling spaces is to make appropriate use of instantiation and inheritance. This helps maintain *local strictness* within metamodelling spaces.

While this approach provided good separation of concerns, both within and between modelling spaces, it failed to account for situations where two metamodelling spaces are mutually dependent on one another, as the dependency between spaces can be only one way. For example, a process that produces a diagram that represents the process that was performed to create it (where the process metamodelling space and the product/diagram metamodelling space are separate). Moreover, it does not resolve the issue if there is a local cycle between meta-levels of a metamodelling space. For example, a class diagram that refers-to (or represents) the classes used to define the class diagram. This type of cycle between a more concrete object that represents a more abstract object from its own hierarchy occurs frequently in larger models that support ecosystem and lifecycle management. Part of the limitations of the original approach is that it attempts to stay with a four-layer architecture, such as that used by MOF/UML. When not constrained to four layers, concepts and objects can more easily be situated in the level most appropriate to their definitions.

The important point of the metamodelling spaces solution is that there is an encompassing meta-level involved that provides the definitions of relationships that pass between metamodelling spaces (and would otherwise violate the strictness criterion). The SLICER (linguistic) meta-model provides such an encompassing meta-level, which defines appropriate relationships for the crossing of metamodelling spaces and levels.

Mapping identification and execution We have implemented a mapping tool, the *UniSA Transform Engine* (or, UTE), enabling data integration with transformation operators based on model-driven principles and a modular, extensible architecture. Mappings are defined in terms of fully declarative *mapping templates*—small reusable fragments—which considerably facilitate the integration process. The transformation engine uses mapping templates to find occurrences of data patterns in specific source models and composes the target model(s). OGI Pilot demonstrations of transformations from ISO 15926 to MIMOSA CCOM or vice versa have been given at industry events, e.g. ISA Automation Week.

The example in Figure 4 shows on the right a subset of a *joint model* based on our multi-level modelling framework and on the left a subset of the MIMOSA

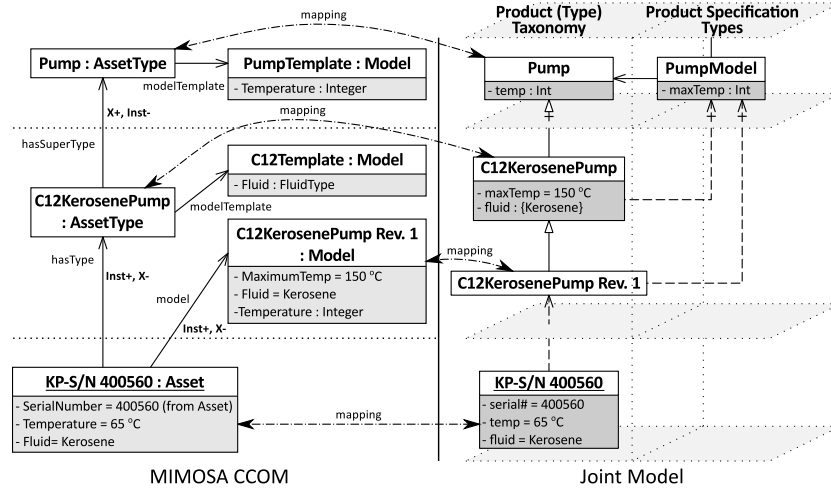


Fig. 4: Transformation example between Joint Metamodel and MIMOSA CCOM

CCOM specification². The MIMOSA specification is based on a UML class model (the linguistic metamodel) and a reference data library (the ontological model). CCOM provides general concepts such as *Asset*, *AssetType*, and *Model* in the UML model, while the reference data includes specific types, such as *Kerosene Pump*. Organisational instance data, such as the specific pump *KP-S/N 400560*, extends the ontological model.

There are two aspects to this: the identification of mappings between the linguistic metamodels, and those between the ontological metamodels. In the former, we can identify correlations between the *hasType* association and as an *Instantiation* in SLICER, and between *hasSuperType* and *Specialisation*. Moreover, the *model* relationship between *Asset* and *Model* can be seen as an *Instantiation* type relationship, with an implied *Specialisation* between the *Model* and the *AssetType* for the *Asset*. This is because the *Model* should incorporate all of the attributes of the *modelTemplates* associated with the hierarchy of *AssetTypes*.

Using this information, we can induce levels in the CCOM model, which would otherwise be flat, and align them with the Joint model to aid in identifying mappings between them. Starting with the lowest level, which is in this case the kerosene pump *KP-S/N 400560*, we can query the joint model and, using standard matching technique, identify the mapping between the two kerosene pumps. This serves as the basis for extending the mappings. By using the information in the models to determine which meta-properties hold for the CCOM relationships and by performing additional querying and reasoning on the joint model (e.g. querying the attributes of the types), further mappings can be identified.

Having identified these mappings in a concrete example model, we can define more abstract mapping templates for UTE. For example, one template may

² For simplicity, attributes are shown within the objects rather than as separate Attribute objects

transform an *Asset* its *Model* (through the *model* association) and its *AssetType* (through the *hasType* association) into an object at the lowest level of the joint model and two types, one specialising the other, with the more specialised type being instantiated by the object. In addition, such a transformation could infer the existence of a specification somewhere in the type hierarchy such that the more specialised type is in an *InstX* relationship with it.

4 Conclusion

The SLICER framework [11] provides a multi-level base model that allows making explicit the types of relationships that will exist between different stages in the engineering lifecycle, e.g., industrial artefacts relative to their specifications. Identifying the underlying semantics required for the relationships in such a model permits creating mappings across ecosystem boundaries, the establishment of a set of guidelines for consistent use of these relationships, so that they can be used as an aid for identifying consistent levels across mappings.

References

1. N. Young and S. Jones. SmartMarket Report: Interoperability in Construction Industry. Technical report, McGraw Hill, 2007.
2. ISO. *ISO 15926 – Part 2: Data Model*. 2003.
3. S. Berger, G. Grossmann, M. Stumptner, and M. Schrefl. Metamodel-Based Information Integration at Industrial Scale. In *Proc. MODELS 2010*, volume LNCS 6395, pages 153–167. Springer, 2010.
4. C. Atkinson and T. Kühne. The Essence of Multilevel Metamodeling. In *Proc. of UML 2001*, LNCS 2185, pages 19–33. Springer, 2001.
5. C. Gonzalez-Perez and B. Henderson-Sellers. A powertype-based metamodeling framework. *Software & Systems Modeling*, 5(1):72–90, 2006.
6. J. de Lara, E. Guerra, R. Cobos, and J. Moreno-Llorena. Extending deep metamodeling for practical model-driven engineering. *Computer*, 57(1):36–58, 2014.
7. M. Schrefl and M. Stumptner. Behavior consistent specialization of object life cycles. *ACM TOSEM*, 11(1):92–148, 2002.
8. W. Klas and M. Schrefl. *Metaclasses and Their Application*. LNCS 943. 1995.
9. D. Chen, G. Doumeingts, and F. B. Vernadat. Architectures for enterprise integration and interoperability. *Computers in Industry*, 59(7):647–659, 2008.
10. Y. Ducq, D. Chen, and B. Vallespir. Interoperability in enterprise modelling: requirements and roadmap. *Adv. Eng. Informatics*, 18(4):193–203, 2004.
11. M. Selway, M. Stumptner, W. Mayer, A. Jordan, G. Grossmann, and M. Schrefl. A conceptual framework for large-scale ecosystem interoperability. In *Proc. ER’15*, Stockholm, 2015. To appear.
12. Barry Smith. Against idiosyncrasy in ontology development. In *Proc. Formal Ontology in Information Systems (FOIS 2006)*, pages 15–26, 2006.
13. Christopher A. Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. *Data Knowl. Eng.*, 39(1):51–74, 2001.
14. C. Atkinson and T. Kühne. Processes and products in a multi-level metamodeling architecture. *IJSEKE*, 11:761–783, 2001.