
University of South Australia

The Library
www.library.unisa.edu.au

Educating Professionals, Creating and Applying Knowledge, Engaging our Communities

This is the accepted manuscript (post print) version of a published work that appeared in final form in *Software and systems modeling*, vol. online, pp. 1-26, Copyright 2016 Springer.

This version has been peer reviewed, but may lack the final formatting of the published version.

This article is copyright and may be used for non-commercial purposes. Refer to the published work for the definitive version.

Noname manuscript No.
(will be inserted by the editor)

An Integrated Multi-Level Modeling Approach for Industrial Scale Data Interoperability

Muzaffar Igamberdiev · Georg Grossmann · Matt Selway · Markus Stumptner

Received: date / Accepted: date

Abstract Multi-level modeling is currently regaining attention in the database and software engineering community with different emerging proposals and implementations. One driver behind this trend is the need to reduce model complexity, a crucial aspect in a time of analytics in *Big Data* that deal with complex heterogeneous data structures.

So far no standard exists for multi-level modeling. Therefore, different formalization approaches have been proposed to address multi-level modeling and verification in different frameworks and tools. In this article, we present an approach that integrates the formalization, implementation, querying and verification of multi-level models. The approach has been evaluated in an open-source F-Logic implementation and applied in a large scale data interoperability project in the oil and gas industry. The outcomes show that the framework is adaptable to industry standards, reduces the complexity of specifications and supports the verification of standards from a software engineering point of view.

Keywords Multi-level modeling, interoperability, multi-level model reasoning, F-Logic, multi-level model querying, multi-level model verification

The final publication is available at Springer via `http://dx.doi.org/10.1007/s10270-016-0520-6`
The final publication includes some corrections not made in this accepted manuscript.

M. Igamberdiev, G. Grossmann, M. Selway and M. Stumptner
Advanced Computing Research Centre
School of IT and Mathematical Sciences
University of South Australia, Mawson Lakes, SA 5095, Australia
E-mail: firstname.lastname@unisa.edu.au

1 Introduction

Multi-level modelling (also called *deep meta-modeling*), a paradigm to model at an arbitrary number of meta-levels instead of traditional two (model and meta-model), was originally proposed in the seminal work of Atkinson and Kühne [3,14]. It is currently regaining attention with different emerging proposals and implementations. An international workshop, MULTI[1], focusing on this research area has been organized in conjunction with the MODELS conference [8,9] for the last two years. One driver behind the application of multi-level modeling is to simplify complex models. In the context of this paper, simplification means clarification of the hierarchical structure of a model through separating domain-specific from language-specific concepts by organizing the model as the same and intuitive as in the underlying application domain. This advantage is crucial for the work presented here, in particular, to simplify data interoperability at an industry scale and for big data applications. Recent work enriches multi-level modeling with new features [5, 52], proposes a formalization for multi-level modeling [54] or demonstrates its practical application [23,37]. The most often mentioned arguments for multi-level modeling are *increased expressiveness*, by introducing multiple classifications [12], and *reduced complexity* [14,53]. These arguments seem to be a contradiction, because one might expect increasing expressiveness may lead to increased complexity, while multiple classifications allow breaking down a complex specification into smaller and simpler levels. Three advantages have been identified based on a concrete use case from the oil and gas industry [37]: (1) simplification of the standards' specifications by classifying elements into ontological and linguistic elements, (2) simpler management and evolution of standards by structuring them into multiple ontological *instance-of* levels, and most importantly, (3) verification of correctness properties on the specification of standards. The hierarchical complex data structures of domain standards justified the oil and gas industry as an accurate application area for multi-level modeling. The three advantages facilitate the interoperability between software systems.

The development life cycle of an interoperability solution consists of multi-level conceptualization, formalization, matching (mapping) and transformation, and verification phases. Recent approaches [5,52,54] use different frameworks or tools to address these development phases. This paper addresses all these phases in an integrated multi-level modeling framework. It focuses on addressing multi-level modeling semantics, implementing and verifying them in Flora-2, an open-source implementation of F-Logic with numerous extensions [40].

Contribution: The contribution presented in this article is threefold: First, an Open integrated framework for Multi-Level Modeling (OMLM) is developed. Compared to existing approaches it adds a new dimension called the *realization dimension*. Existing multi-level modeling approaches distinguish

[1] MULTI - International Workshop on Multi-Level Modelling, http://goo.gl/EaVok1

between a *linguistic-* and an *ontological dimension* and ignore the requirement that a multi-level modeling framework needs to be implemented using an existing language. Since no standard for multi-level modeling and no existing language support multi-level modeling features directly, a realization dimension is needed to explicitly capture the mapping of the multi-level meta-model to an existing programming language. Such a dimension provides multiple advantages such as decoupling the multi-level framework from the implementation language, comparing of implementations in different programming languages and automating code generation.

The second contribution covers the implementation of the presented framework in Flora-2 and its evaluation of a large industry project in the oil and gas domain, the *Oil & Gas Interoperability Pilot (OGI Pilot)*. Flora-2 offers knowledge representation features and reasoning capabilities. The implementation allows formalization, querying and verification multi-level models using a single language, Flora-2.

The third contribution is a verification framework called *MULti-LEvel Reasoner (MULLER)*. MULLER aims to verify correctness properties of multi-level models using verification rules. It is also implemented in Flora-2. An integrated modeling approach is enabled by using the same language for specifying the semantics and verification of multi-level models.

This paper extends previous work [35] substantially in three ways: (1) the semantics of the OMLM meta-model are described in a new Section 3, (2) the framework has been evaluated in the context of the industry use case *OGI Pilot* in a new Section 6, and (3) the related work section was rewritten and extended with recently published work.

The paper is organized as follows. Section 2 discusses the use case from the oil and gas industry by illustrating the benefits of the multi-level modeling approach. Section 3 explains the semantics of the OMLM framework. The implementation of the framework is described in Section 4. Section 5 addresses the reasoning component (MULLER) of the integrated approach, and Section 6 provides an evaluation of the framework followed by a discussion of related work in Section 7. The paper concludes and highlights future research in Section 8.

2 Use case

This section provides details of our oil and gas industry example and the benefits of multi-level modeling in this domain. Initially, the OGI Pilot use case will be introduced, followed by a brief overview of multi-level modeling and its application in this specific context.

2.1 Standards based interoperability

Large-scale interoperability is one of the main challenges in all industries that rely on large engineering plants for their operation. Inadequate interoperability

is estimated to cost about $15.8 billion per year, representing between 2.84 percent of operation and maintenance cost [27,28], according to the US Capital Facilities Industry and the construction and engineering domain.

At least, two types of the large-scale data interoperability can be considered: (1) the interoperability between individual components of stakeholders' information ecosystems, (2) the interoperability between the standards, which includes all inter-operating ecosystems. The former one can be faster, however when we consider number of inter-operating stakeholders together with their data models, architecture, software and different data representations, then the data interoperability gradually becomes challenging. The latter type of interoperability organizes all information ecosystems of inter-operating stakeholders into the standards to facilitate data interoperability.

In the context of the oil and gas industry, the standard based interoperability is performed between two industrial standards, ISO 15926 [36] and MIMOSA's Open Systems Architecture for Enterprise Application Integration (OSA-EAI) [50].

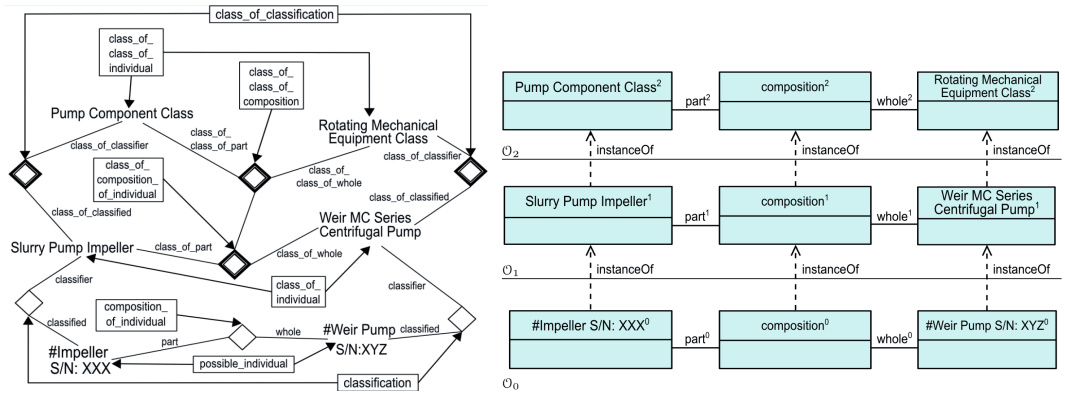
The ISO 15926 standard is used to design very large physical assets, such as an oil refinery or rig. It relies on a 4-dimensional (perdurantist) information model specified in STEP/EXPRESS, RDF, OWL and first order logic. It consists of multiple parts. In this paper we focus on Part 2, the *Data Model* and Part 4, the *Reference Data Library (RDL)*. The data model contains base concepts and relationships in ISO 15926 and RDL is an extensive library of domain specific entities. Both parts represent the core specification of the standard. The OSA-EAI is another standard in the oil and gas industry, responsible for operation and maintenance of engineering units, such as oil processing plants. It provides different yet overlapping viewpoints of the same domain information. It relies on UML, XML and XML Schema to express data formats.

As an example for the standard based interoperability, we present an industrial use case to support data interoperability between the ISO 15926 and OSA-EAI standards.

2.2 Oil & Gas Interoperability (OGI) Pilot

Much effort has been invested into the development of data standards to overcome the interoperability issue in the oil and gas industry. One effort is the joint academic-industry project OGI Pilot hosted by MIMOSA[2] and supported by the ISO TC 184 OGI Technical Specification project. The goal of the OGI Pilot is increased automation in the digital hand-over of design information of very large physical assets to owner operators for the operation and maintenance of assets. This requires identifying commonalities and differences as well as open gaps in the specifications of two major standards in this domain: ISO 15926 and OSA-EAI. Within the OGI Pilot we have identified some challenges [49] of which we will focus on the simplification and verification of ISO 15926 by

[2] MIMOSA: http://www.mimosa.org



(a) Instance diagram (b) Multi-level representation

Fig. 1: Two representations of the same example: *relating an impeller to a pump*.

means of the OMLM and MULLER frameworks, respectively.

Example: An excerpt from an engineering diagram specifies that "*The impeller with serial number XXX is part of the Weir Pump with serial number XYZ*". Fig. 1a displays a model of this example using *instance diagram* notation appearing throughout the ISO 15926 documentation [36].

In the ISO 15926 terminology, each box represents a *class* from Part 2 of the specification and identified by its label. A diamond represents a *relationship* where a diamond with a thick line represents a *class of relationship*. A (class of) relationship has *roles* which are displayed by labeled arcs connected to the diamond, e.g., "part" and "whole" are the two roles of the relationship "composition_of_individual". A symbol with prefix # is a *possible individual with a temporal part*, e.g., "#Impeller S/N: XXX" is a possible temporal part with identifier "Impeller S/N: XXX". The remaining elements are *classes* identified by their labels, e.g., "Pump Component Class". The arrows from the boxes to other elements indicate instantiation.

2.3 Application of multi-level modeling on the OGI pilot

One of the goals of the application of multi-level modeling is to separate a flat RDL model into linguistic and ontological dimensions. The two-level RDL model is a flat model in a sense that it contains a mixture of linguistic and ontological elements. The separation of (linguistic and ontological) concerns is achieved by applying the OMLM framework. For example, Jordan et al. [37] applied rules to the ISO 15926 specification for the transformation of the flat model into a multi-level model. One of those rules assigns a model level according to the prefix of a class label, e.g., "class_of_composition" is an instance of

"class_of_class_of_composition" and removes the prefix "class_of_" to simplify the notation. Fig. 1b shows the result of applying those rules to the example introduced above.

Some of the advantages of a multi-level model representation over the flat model are: (1) distinguishing between the linguistic and ontological *instance-of* relationships and concepts, (2) separation of concerns through multiple levels making it easier for users to focus on particular aspects of the model, (3) reduced complexity, and (4) clarified terminology which improves understandability. The reduced complexity is achieved not in number of modeling elements, but in separating the linguistic and ontological elements and relationships that lead to efficient task distribution between the roles of model developer and domain engineer. Additionally, complexity is reduced by organizing the hierarchical model structure as the same and intuitive as in the underlying domain.

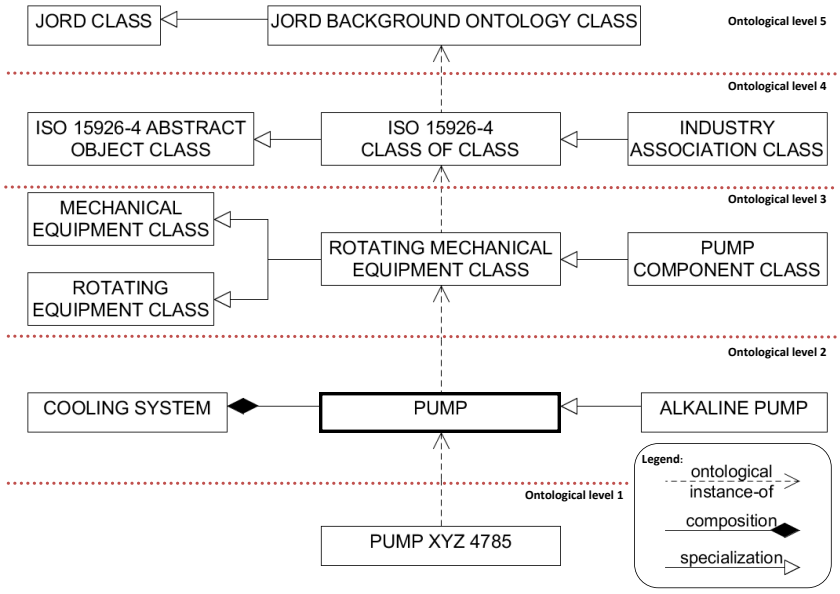


Fig. 2: A pump and its related concepts are divided into ontological levels, after applying multi-level modeling to RDL (ISO 15926-4)

As another example, the hierarchical structure of an industrial pump is illustrated in Fig. 2. The pump is one of the building components in RDL. Pumps are used in many complex physical systems like a cooling system or refinery. In addition to Fig. 1b, Fig. 2 demonstrates a different perspective on the application of multi-level modeling to the concept of a pump focusing on its related hierarchical structure. All ontological "instance-of" relationships

and classes (ontological levels 2-5) in Fig. 2 would be located in M2 and M1 levels of a flat model as the linguistic elements. Multi-level modeling places these domain related concepts and relationships in the ontological dimension by separating them from the linguistic elements. The elements that represent the object-oriented modeling principles are considered as linguistic and the rest represent domain related concepts as ontological elements (see Fig. 2). Five ontological levels are introduced (by means of the ontological instance-of relationships) to clarify the model. The semantics of multi-level modeling and model inconsistencies are checked by verification rules.

A formal framework is required to verify the correctness properties of multi-level models and to perform queries, for example, to support matching with other standards and perform model transformations. The next section describes the semantics of the proposed OMLM framework.

3 Open multi-mevel modeling (OMLM) framework

As the name says, the OMLM framework presented in this paper is "open" (in the sense that it allows a particular domain model to be plugged-in and its implementation in a language of choice) and based on multi-level modeling principles. As an example implementation language we use Flora-2, which is discussed in Section 4. The OMLM framework distinguishes itself from existing approaches by the fact that it has a plugging mechanism for the domain models with their own linguistic elements and considers an extra dimension, a realization dimension. The realization dimension captures the concepts of an implementation language at the model level and is not considered in existing approaches [13,52,54]. We first describe the framework and its application workflow and then explain the semantics and user roles.

3.1 The OMLM meta-model

The meta-model of the OMLM framework with its plugged-in ISO 15926 classes is shown in Fig. 3. First we explain the meta-model, then discuss the plug-in mechanism of the framework. The meta-model encompasses classes covering three dimensions: (1) the linguistic dimension provides concepts related with grammar and syntax for defining ontological dimension model elements. (2) Domain related concepts and relationships are involved in the ontological dimension. (3) The concepts in these two dimensions are realized or implemented by one or multiple concepts in the realization dimension. Fig. 3 shows a minimal realization dimension which consists of only a single concept, *object*. Therefore, everything in the OMLM framework is represented as an object.

The linguistic dimension of OMLM consists of the concept subclass hierarchy shown in Fig. 3. The concept at the top of the hierarchy is `Instantiable Element` from which other concepts such as `Clabject`, `Domain Connection`

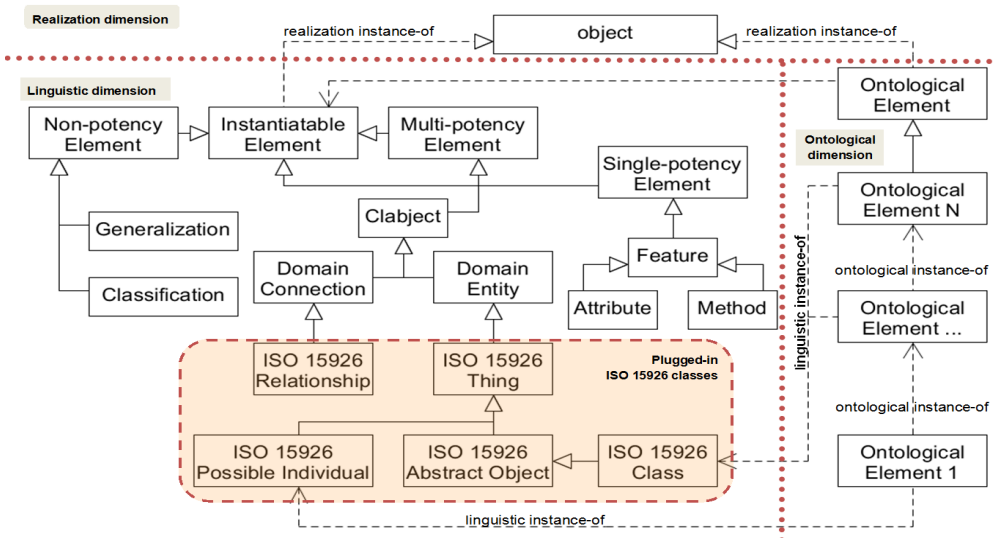


Fig. 3: The meta-model of the OMLM framework with the plugged-in ISO 15926 model (application settings)

and `Domain Entity` inherit. Potency based elements have been inspired by related work [10,54] and represent the core modeling elements and relationships. All subclasses of `Instantiatable Element` are considered as linguistic elements.

All concepts of the ontological dimension are the linguistic instance-of the elements in the linguistic dimension. The linguistic elements are the subclasses of `Instantiatable Element`. The ontological elements are the subclasses of `Ontological Element` and linguistic instance-of `Clabject`. The linguistic instance-of relationship between `Ontological Element N` and `ISO 15926 Class` is a specialization of the linguistic instance-of relationship between `Ontological Element` and `Instantiatable Element`.

We briefly discuss the motivation and origin of the concepts in OMLM. The framework centers on the notion of `Instantiable Element`, which is the root element and superclass of all other concepts, such as `Clabject` and `Feature` [10]. This notion means that a linguistic element can be linguistically instantiated into the ontological dimension. *Potency* represents the possible number of instantiations of a concept through ontological levels [54]. We distinguish between single-potency, multi-potency and non-potency elements. *A single potency* element can only be instantiated once. *A multi-potency* element's instance can be instantiated again. In contrast, *A non-potency element* is used at any ontological level and is not ontologically instantiated. Inspired by [10] and [54], we have combined `Instantiable Element` with the single, multi and non-potency concepts and used them as the root triad of our framework. All other concepts are categorized based on this triad. Note that within

OMLM, unbounded potency (i.e. models can be extended over an unlimited number of classification levels) [11] is supported in the way that a rule automatically assigns/updates potency to all elements at the ontological levels. The potency assignment rule runs after creating or updating model elements.

Classification and *generalization* (a.k.a. specialization) are the key relationships of the OMLM framework and used across all ontological levels without ontological instantiation. Classification represents an instance-of relationship between an instance and its type, i.e. an instance is an instance-of or is classified by its type. OMLM distinguishes three types of classification: the *linguistic*, *ontological* and *realization*. Generalization relationship is a relationship between a generic and a specific model element. `Clabject` represents a dual facet, class and object, of a concept, as introduced by Atkinson and Kühne [12]. Domain specific concepts and relationships are represented by the subclasses of `Clabject`, the `Domain Entity` and `Domain Connection` elements respectively. `Feature` element is similar to UML Features with `Method` and `Attribute` subclasses.

Applying the OMLM framework. The OMLM framework is used in two ways: without or with the plugged-in linguistic dimension. The majority of domain models can be directly modeled by the elements of OMLM (see Fig. 3 except the plugged-in ISO 15926 classes outlined with dashed lines). These domain models do not need to plug their linguistic elements into OMLM, since they are already addressed by the OMLM meta-model. However, some domain models or *application settings* have their own linguistic elements in combination with domain elements used for a particular application. The plug-in mechanism supports a connection created between the linguistic elements of a domain model and the OMLM framework in the process of applying multi-level modeling. A domain model is plugged into the framework by means of subclassing the elements of the linguistic dimension (see the plugged-in ISO 15926 classes in the box outlined with dashed lines in Fig. 3). The linguistic elements of a domain model that are aligned with object-oriented modeling principles are selected to be used as the classes in a linguistic plug-in to the OMLM framework. The rest of the model elements are considered as ontological elements. The RDL use case represents such a domain model. This mechanism is an advantage of the OMLM framework, which plugs a domain model with specific linguistic elements into the framework, unlike in MetaDepth [44] and Melanee [4] approaches.

Fig. 3 shows how the "ISO 15926 specific" linguistic model elements are plugged into the OMLM meta-model (a box outlined with dash lines in Fig. 3). `Thing` is the root element of the ISO 15926 hierarchy. It subdivides into physical and abstract things that are represented by `Possible Individual` and `Abstract Object`. The ISO 15926 standard defines `Possible Individual` as "a thing that exists in space and time". In contrast, `Abstract Object` is "a thing that does not exist in space-time" [36]. A key subclass of `Abstract Object` is `Class` . All ontological elements are instances of `Class`, except the ones at ontological level 1, which are instances of `Possible Individual`.

Ontological elements are domain related concepts, specifically in case of ISO 15926, they are engineering (oil and gas, construction, capital facility industry) concepts.

3.2 The OMLM workflow

The OMLM can be applied in two ways to create a multi-level model in a particular domain: one can create them by defining concepts and relationships or transform them from already existing two-level models. The framework supports both options. In case of creating a new model, a domain engineer builds his/her domain model from the concepts and relationships in the OMLM meta-model. Since all existing models are represented in two-level models, the transformation of two-level models into multi-level ones would be enormously helpful for successful adoption of multi-level modeling in academia and industry. In this paper we focus on the process (workflow) for transforming two-level models into multi-level models. The workflow elements and steps are illustrated in Fig. 4. The concrete application of the workflow is discussed in the implementation section, particularly in Section 4.2.

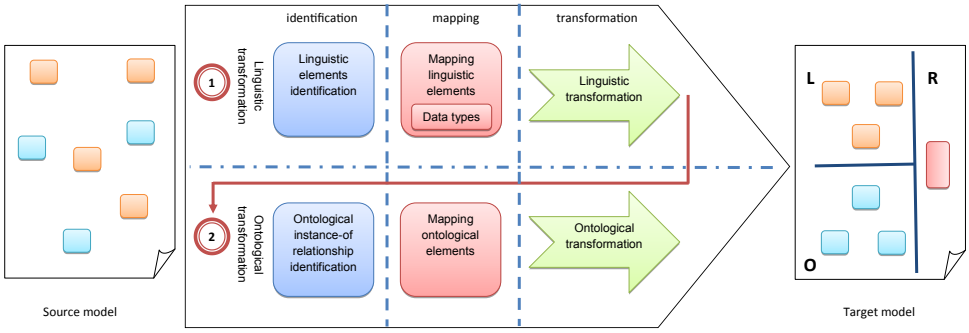


Fig. 4: The OMLM workflow describes the steps for applying the framework to a specific domain model

The conceptual view of the two-level to multi-level transformation framework reflects the OMLM framework and distinguishes between the linguistic and ontological transformations as illustrated in Fig. 4. While the source model represents a two-level model, the target model represents a multi-level model based on the linguistic (L), ontological (O) and realization (R) dimensions. In addition, the linguistic and ontological transformations consist of identification, mapping and transformation steps.

Firstly, within the linguistic transformation, some decisions are made to identify linguistic elements in the identification stage. Usually the linguistic elements are obvious, however sometimes because of diverse hierarchical domain

structures it is not easy to identify them. The domain engineer can support this decision stage. Secondly, based on the identification, the linguistic elements and primitive data types are mapped to the target linguistic dimension. The mappings of the linguistic elements are specified manually, since they rely on input from domain experts. Finally, the actual linguistic transformation is performed in a transformation language (the top arrow in Fig. 4).

The ontological transformation proceeds in identical stages. Initially, a relationship that divides the ontological dimension into levels should be determined. Traditionally, it is the classification relationship, however sometimes it can be some other relationship. In ISO 15926, prefixing a class name with `classOf` indicates an implicit change in levels [37]. For example, the `classOfXYZ` class is assumed to be at one level above the class `XYZ`, even though they are not connected with any instantiation relationship.

Similarly, the mappings are performed in the ontological context. Finally, the ontological transformation splits the original models into ontological levels based on the decided relationship. The generated multi-level model consists of three dimensions: linguistic, ontological and realization (see Fig. 4).

3.3 The OMLM semantics

The semantics of a model are important for its modeling, implementation and verification. Table 1 contains description of the core elements and structure of the framework. Table 2 describes the plugged-in (into the linguistic dimension) ISO 15926 classes. We have decided to base the linguistic plug-in on `Thing`, `Class` and `Relationship` concepts, since they contain domain related properties. For example, they have `defaultRdsId` and `IdPCA` properties for Reference Data Service (RDS) and POSC Caesar Association (PCA)[3] related identification. These domain related properties could not be generalized and introduced in `Clabject` concept.

3.4 User roles in the OMLM framework.

Different user roles are involved in the use case when using the OMLM framework. The produced multi-level model is expected to be used from five user-perspectives: operations and domain engineer, asset designer, model and language developer. Fig. 5 illustrates these users in the context of the framework.

1. The operations engineer is responsible for daily operations of an engineering asset, such as an oil platform or refinery. He/she uses the concrete physical elements (e.g. `PUMP XYZ 4785`), as the instances of `Possible Individual`, at the bottommost ontological level.

[3] PCA promotes the development of open specifications to be used as standards for enabling the interoperability of data, software and related matters. https://www.posccaesar.org/

Table 1: Multi-level modeling semantics

No.	Concept	Description
1	Dimension	Multi-level modeling is organized in three dimensions: realization, linguistic and ontological. They represent implementation, syntax & grammar and domain knowledge perspectives, respectively. Each dimension is represented by its concepts (see Fig. 3).
2	Realization dimension	realizes the multi-level modeling described in linguistic and ontological dimension. The semantics of realization depend on the language used to implement the framework.
3	Linguistic dimension	contains the syntax and grammar of the multi-level model. The linguistic model elements are instantiated in the ontology dimension.
4	Ontological dimension	represents the structural hierarchies of ontological elements and their relationships in the underlying domain. The semantics of an ontological concept is specified by its ontological types. The ontological dimension can have an arbitrary number of ontological levels.
5	Realization instance-of relationship	is a cross-dimension instance-of relationship between an ontological or linguistic element and an element in the realization dimension. For example, all ontological and linguistic elements are *objects* (see Fig. 3).
6	Linguistic instance-of relationship	is a cross-dimension instance-of relationship from the ontological to linguistic element. Each model element in the ontological dimension has a linguistic instance-of relationship to a model element in the linguistic dimension.
Realization dimension concepts		
7	Object	It implements all linguistic and ontological elements. The property dimension characterizes each element.
Linguistic dimension concepts		
8	Instantiatable element	represents the root element of the subclass hierarchy in the linguistic dimension and is abstract.
9	Single-potency element	The instances of a single-potency element can only be instantiated once.
10	Multi-potency element	The instances of a multi-potency element are instantiated more than once. The number of multiple instantiations is defined by model element's potency. One of the core elements of multi-level modeling, the `Clabject` element is a subclass of multi-potency element.
11	Non-potency element	is used at any ontological level and is not ontologically instantiated. This element is abstract and has the classification and generalization subclasses, which represent relationship concepts in the ontological dimension.
12	Clabject	An abstract concept, has a dual facet as a class/type and an object/instance. Its instances at the topmost and bottommost ontological levels represent only class and only object facet respectively. At the intermediate levels, all `Clabject` instances have both facets. The subclasses of the `Clabject` element, `Domain Entity` and `Domain Connection` represent the entities and relationships of the underlying application domain [10].
13	Feature	an abstract concept that is either a `Method` or an `Attribute`. Since it is a single-potency element, it is instantiated only once. The `Method` concept was inspired by MetaDepth [44]. A `Method` is instantiated as the same as method overriding. A `Method` is defined at the top ontological level and is instantiated/called/overridden at one of the bottom levels. MetaDepth has adapted EOL (a feature to define a method on a global scope and attach methods to meta-class) to implement a method overriding mechanism across meta-levels. When a method is called/ instantiated on a model element, its type is looked up using dynamic dispatch. In Flora-2, method instantiation can be supported by a rule for type lookup.
Ontological dimension concepts		
14	Ontological element	represents the elements used in the underlying domain. A linguistic instance-of an element (e.g. `Clabject`) at the linguistic dimension.
15	Ontological instance-of relationship	an instance-of relationship between two ontological elements. Represents a type-instance relationship between domain elements.
16	Ontological level	(Not shown explicitly in Fig. 3) represents an abstraction level within the hierarchy of domain concepts in the ontological dimension. Groups a set of ontological elements which are at the same level of abstraction.
17	The topmost-level element	located at the topmost level of the ontological hierarchy and has the class facet only.
18	The bottommost-level element	located at the bottommost level of the ontological hierarchy and has the instance facet only. Is not instantiated further. The bottommost ontological level is also called as *instance level* within the RDL model.

Table 2: Multi-level modeling semantics of the ISO 15926 standard

Concept	Description
ISO 15926 Thing	`Thing` is the root element of the ISO 16926 data model. All model elements in ISO 15926 are represented as the instances of `Thing`. It is subclassed into `Possible Individual` and `Abstract Object`. The former represents a concrete object and is only instantiated at the bottommost ontological level. The bottommost level elements are instances of `Possible Individual`. `Abstract Object` represents an abstract `Thing` and has a subclass `Class` that instantiates in the ontological dimension.
ISO 15926 Class	represents the elements of the domain. Can be instantiated at all ontological levels except the bottommost level.
ISO 15926 Relationship	A specific class of `Domain connection` that defines a relationship between domain elements.

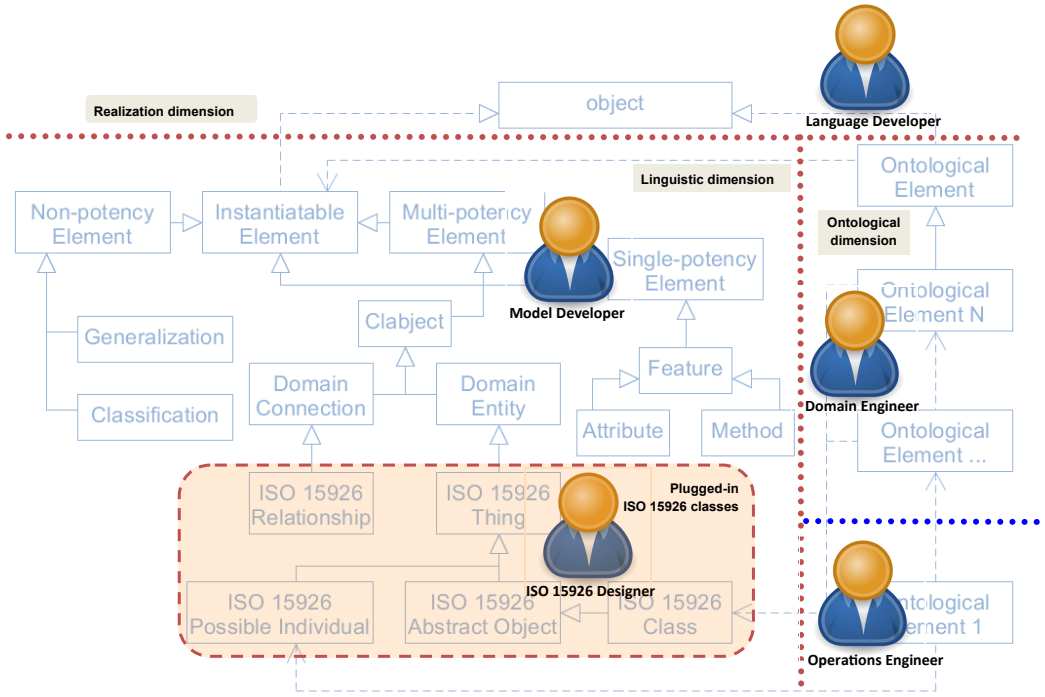


Fig. 5: User roles in the OMLM framework.

2. The domain engineer works at all ontological levels except the bottom-most one. He/she designs the structural hierarchy of a domain model by using the RDL elements. An example of such a domain model is observed in Fig. 2.

3. The ISO 15926 designer extends and updates the ISO 15926 standard. This role can also perform tasks from the previous two roles. Depending on the application, these three roles can be performed by a domain engineer.

4. The model developer designs and maintains the linguistic dimension of the OMLM framework. He/she cooperates with the ISO 15926 designer and the language developer to transfer the semantics and clarify the requirements of the framework.

5. The language developer implements the model elements of the ontological and linguistic dimension in a specific implementation platform/language, for example in Flora-2 in our case.

Example 1 In order to illustrate the roles of involved users in the multi-level modeling context better, consider an example of designing and operating a concrete and complex piece of equipment. *A debutanizer* separates butane from natural gas during the refining process. The process involves usage of the pump model (Fig. 2). The operations engineer configures and operates the instances of `Possible Individual` such as condenser, pump (`PUMP XYZ 4785` in Fig. 2), separator and reboiler. The operational parameters and design of the debutanizer are organized by the domain engineer, who also matches the composing parts of the debutanizer (ontological levels 2-5 in Fig. 2). The ISO 15926 designer makes sure that the debutanizer is designed from the ISO 15926 model elements (the ISO 15926 specific elements in Fig. 5). The model developer provides the multi-level modeling elements to model the debutanizer. The debutanizer is modeled in a specific programming language by the language developer (realization dimension in Fig. 5). Finally, the model is implemented by a software developer (for the sake of brevity, this role is not shown in Fig. 5).

4 Implementation of the OMLM framework in Flora-2

In this section, we first introduce Flora-2 briefly. We then describe an implementation of the OMLM framework in Flora-2. Afterwards, the workflow steps to obtain a multi-level model from the original RDL model are discussed.

Flora-2 is an F-Logic dialect and we use F-Logic for several reasons. First, because of its knowledge representation and reasoning features, and the fact that its frame syntax is the core of the RIF-BLD dialect of the RIF (Rule Interchange Format) Semantic Web standard. Most importantly in terms of representation language choices, common Semantic Web languages such as OWL or RDF are based on the Open World Assumption (OWA) due to their original goal of permitting the free combination of arbitrary, independently created knowledge fragments via the World Wide Web. Conversely, the Close World Assumption (CWA) at the class level, as embedded in F-Logic and similar object-rule or object-constraint languages, is significantly more conducive

to represent and reason about technical systems in (in particular, large-scale) design contexts, as demonstrated by real-world applications of such languages to industrial configuration [26,58], business process design [48], and even natural language requirements engineering [56,42] using the OMG SBVR standard. (It is worth re-emphasizing that the CWA applies only at the class level and still permits the automated construction of arbitrarily large structures of F-Logic instances.)

Flora-2 stands for *F-Logic Translator* [40] and has a simple and expressive syntax with well-defined declarative and object-oriented frame-based semantics. It is a unified language of F-logic, HiLog, Transaction Logic and defeasible reasoning. It benefits from a natural way of meta-programming in the style of HiLog and from logical updates in the style of Transaction Logic [40]. Flora-2 combines object-oriented frame based modeling with declarative style. It has a higher-order syntax and a first-order semantics. It supports knowledge representation features such as typing, object identity, complex objects, methods, classes/ subclass hierarchy and modularization, and reasoning features such as inheritance, meta-reasoning, rules, queries and scoped inference and classification [1]. It supports complex data/object structures that exactly match with our use. These characteristics make it practical to apply to multi-level modeling and verification.

Example 2 To introduce Flora-2 syntax, we represent the object elements of the realization dimension as shown in Listing 1. It defines the mapping of the realization dimension into Flora-2.

```
1 /** The Realisation dimension **/
2 Object[dimension*=>Dimension].
3 ontological:Dimension.
4 linguistic:Dimension.
```

Listing 1: The representation of the realization dimension in Flora-2

The Flora-2 object, named `Object`, has an inheritable (`*=>`) property called `dimension` with a value `Dimension` (line 2). The `ontological` and `linguistic` elements have the instance-of relationship (denoted by `:`) with `Dimension` (lines 3-4), i.e. they are instances of `Dimension`. The `dimension` property is used to identify all instances of `Object`.

The primitive data types of the OMLM framework reuse the datatypes of Flora-2. They include different types such as `_long`, `_integer`, `_double`, `_decimal`, `_string`, `_symbol`, `_object`, `_time`, `_date`, `_dateTime` and `_duration`. Within the OGI Pilot use case, EXPRESS data types (e.g. `ExpressBinary`, `ExpressInteger` and `ExpressString`) are used in the original ISO 15926 model. We transfer from them into Flora-2 primitive data types.

4.1 The OMLM implementation

Many modeling elements like objects, properties, inheritance, primitive data types, predicates and rules are the built-in features of Flora-2. To support the multi-level modeling semantics, we have implemented some features, such as the linguistic, ontological and realization dimension, potency, potency assignment mechanisms and relationships. We introduce two key relationships in this subsection, particularly the linguistic and ontological classifications (instance-of) relationship.

Linguistic instance-of relationship: a linguistic relationship between a type and its instance. We introduced a new operator, the '`<:`' to represent the relationship. It was introduced as an operator that bonds its two arguments. The operator is used between two arguments in a statement (e.g. `PUMP<:CLABJECT.`).

Ontological instance-of relationship: an ontological relationship between a type and its instance. It is denoted by '`<::`'. Similarly, the operator is introduced by '`:-_op()`' statement.

4.2 Application of the OMLM workflow

The workflow described in Section 3.2 is applied to the use case introduced in Section 2. The result of application of the workflow is shown in Fig. 6. The RDL model elements are referred by their types in the data model. To transform the two-level ISO 15926 model into a multi-level model, we split the transformation into two separate steps: syntactical and the two level to multi-level model transformations (see Fig. 6). The latter one is performed by means of a model transformation framework also implemented in Flora-2. The generated multi-level model is verified in Section 5.

4.3 Syntactical transformation

The syntactical transformation mapped the ISO 15926 model represented in OWL/RDF format into an intermediate representation in Flora-2 (see example in Fig. 7). The mapping uses a visitor pattern to visit the hierarchies of the model elements via the parsing and rendering features of the OWL Java API [34]. The classes and individuals of ISO 15926 are mapped into Flora-2 objects with the standard Flora-2 instance-of (denoted by '`:`') relationship. The one-to-one mapping can be observed between property and value elements in Fig. 7.

While we use the transformation on the RDL model,the source model can be any model that conforms to the ISO 15926 specification. The target model is generated as Flora-2 facts in the form `obj[prop->value]`. For the sake of identification, traceability and change propagation, we used the `defaultRdsId` and `hasIdPCA` properties of classes. They uniquely identify each class used in the model. In the next stage, we consider to transform a two-level into a multi-level model.

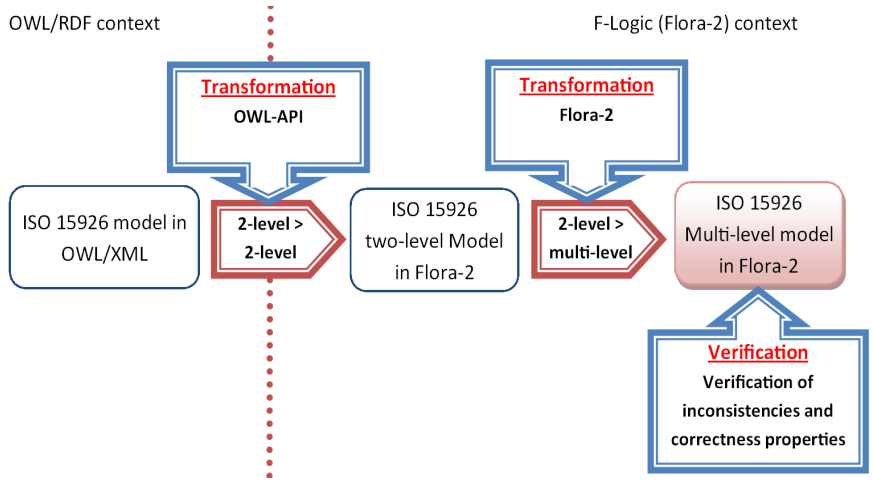


Fig. 6: Implementation scenario of the multi-level modeling

```
<part2:PossibleIndividual>
        <rdl:hasIdPCA>RDS331081291</rdl:hasIdPCA>
        <rdfs:label>ryland</rdfs:label>
        <rdl:hasDesignation>ryland</rdl:hasDesignation>
        <rdl:hasStatus>Recorded</rdl:hasStatus>
        <rdl:hasCreator>ryland</rdl:hasCreator>
        <rdl:hasCreationDate>2006.12.11</rdl:hasCreationDate>
</part2:PossibleIndividual>
```



```
RDS331081291:PossibleIndividual.
RDS331081291[label->"ryland"].
RDS331081291[hasDesignation->"ryland"].
RDS331081291[hasCreator->"ryland"].
RDS331081291[hasCreationDate->"2006.12.11"].
RDS331081291[hasStatus->"Recorded"].
RDS331081291[hasIdPCA->'RDS331081291'].
```

Fig. 7: Syntactic transformation example

4.4 The two-level to multi-level model transformation

The second transformation stage transforms the two-level model into a multi-level model. Both the source and target models are represented in Flora-2. The target model is generated by means of a transformation framework. It realizes mappings from the source (two-level) to target (multi-level) model and is illustrated conceptually in Fig. 8.

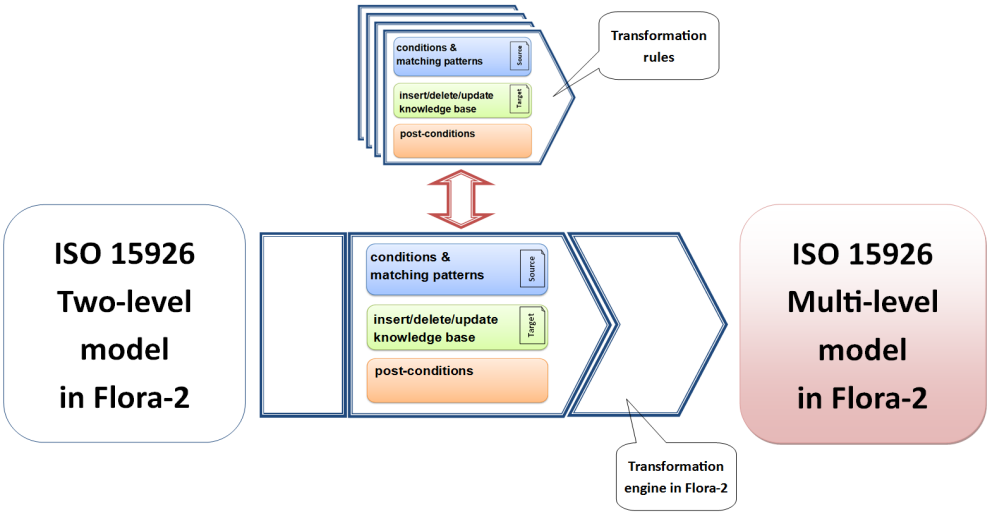


Fig. 8: The two-level to multi-level model transformation framework

The transformation framework is implemented in Flora-2 rules. These rules consist of a head and a body part (e.g. `rule_head(p1,p2) :- body`). The head consists of the name and parameters of a transformation rule. The body comprises of the source, target and post-condition. The conditions and patterns that should be matched in the source model are addressed in the source model section of the transformation rule (as shown in Fig. 8). The facts that are added to the target model, as a result of a match in the source model section, take place in the target model section, while the post-condition section contains the conditions that should be satisfied after the rule has been applied.

The transformation framework has been designed in a modular way so that different transformation rules can be applied (Fig. 8). Each transformation rule specifies the transformation from a two level specification to a multi-level specification in a declarative way. Two example transformations are provided below.

Example 3 The transformation in Listing 2 converts a classification relationship into an ontological instance-of relationship in the ontological dimension. As shown, the `TransfClassif` transformation rule has four parameters (line 1). It fetches any instances of the classification relationship such that the `hasClassified` and `hasClassifier` properties match the respective parameters in the source model. For each matched instance, the transformation rule adds `Classified` and `Classifier` objects and associates them with the ontological instance-of relationship ('`<::`') in the target model. The source and target models are stored in separate Flora-2 modules (reusable code libraries) indicated by the respective variables (`?SourceM` and `?TargetM`).

```
1 %TransfClassif(?Classified, ?Classifier, ?SourceM, ?TargetM) :-
2     ?_CI[hasClassified->?Classified,
```

```
3     hasClassifier->?Classifier]@?SourceM,
4     insert{ (?Classified<::?Classifier)@?TargetM }.
```

Listing 2: A transformation of a classification relationship

Example 4 Similarly, the transformation of a specialization relationship is illustrated in Listing 3. This rule introduces a specialization relationship (denoted by the `::` operator) between a subclass represented by variable `?Sub` and a superclass represented by `?Sup` (line 4).

```
1 TransformSpecialization(?Sub, ?Sup, ?SourceModule, ?TargetModule) :-
2   ?_SI:Specialization[hasSubclass->?Sub,
3   hasSuperclass->?Sup]@?SourceModule,
4   insert{(?Sub::?Sup)@?TargetModule}.
```

Listing 3: A transformation of a specialization relationship

As a summary, a two-level model has been transformed to a multi-level one through two subsequent transformations. The first one transforms into a flat two-level representation (1:1 mapping) in Flora-2, and the second one into a multi-level model. The multi-level model will be used for querying and verification purposes in the next section.

5 Querying and verification

The querying and verification activities can be performed at all levels of ontological abstraction, and linguistic dimension. The advantage of having all models in a single knowledge base, like in the case of Flora-2 is that they are accessed, queried or used by the verification and transformation rules at any ontological level. Verification of correctness properties is performed by the *MULti-LEvel Reasoner (MULLER)* framework. Querying and verification are performed without changing the context to a specific ontological level.

Setting the context to a particular ontological level is a good idea if one needs to access model elements in that particular ontological level. If queries and verification rules involve accessing elements and relationships from upper or lower meta-levels, for example in verifying the level-skipping classification or cross-level generalization relationships (see Section 5.2), then the restricted scope will be a limitation. The OMLM and MULLER frameworks support both restricted and entire scope options.

Different querying and verification tasks are performed:

- Querying the linguistic and ontological instantiation, and generalization (specialization) hierarchy of the model elements;
- Querying the composing parts of the engineering elements, e.g. a pump is a component of a combustion engine;

– Detecting inconsistencies by verifying the correctness properties. For instance, the ontological level-skipping and cyclic relationships can be detected by respective verification rules.

In this section, the reasoning features of Flora-2, such as querying and verification have been applied to the target multi-level model. We firstly demonstrate how to query objects and then how to verify correctness of a model.

5.1 Querying in the OMLM framework

The ontological level elements and their properties are extracted by means of querying. An ontological level can be extracted into a separate Flora-2 module to restrict the scope of querying and verification. The whole multi-level model is stored in the knowledge base and is available for querying or extraction. The scope of queries and verification rules covers all ontological levels and the linguistic dimension.

A knowledge base is built from the facts (e.g. `pump[part->impeller].`) and is easily queried. There are different ways to query a knowledge base in Flora-2 as illustrated in Listing 4.

```
1  ?- ?X<:PossibleIndividual.
2  ?- RDS999869206<:?Y.
3  ?- ?X[hasDesignation->"PUMP"].
4  ?- RDS999869206 <:: ?Y.
5  ?- ?X <:: RDS208394.
6  ?- ?X <:: ?Y[hasDesignation->"ROTATING MECHANICAL EQUIPMENT CLASS"].
7  ?- ?X[hasClassOfWhole->'RDS277244']@rdl.
8  ?- ?X[hasClassOfWhole->RDS277244[label->?L]]@rdl.
9  ?- ?errors = setof {?_error |
10                 subsequent_potency(?_X,?_Y),
11                 ?_error=[?_X,?_Y]},
12                 ?errors.length@_basetype=?number_of_violations.
```

Listing 4: Queries in Flora-2

Lines 1-2 in Listing 4 demonstrate the linguistic instance-of relationship (`<:`) from two perspectives. Line 1 queries an element that is the linguistic instance-of `PossibleIndividual`. Line 2 queries from the opposite perspective to find out what the linguistic instance of the model element `RDS999869206` is. In the same way, lines 4-5 query for the model element that connects with ontological instance-of (`<::`) relationship. Line 3 queries a model element (`?X`) which has the `hasDesignation` property that is equal to `PUMP`. Similarly, lines 7-8 will retrieve a model element which has the `hasClassOfWhole` property with a value of `RDS277244`. In addition, line 8 will display a label of the value, which will result with `COOLING SYSTEM`. The last query in lines 9-12, will display an error set and number of violations of `subsequent_potency` verification rule. This rule makes sure that potency numbering of model elements is in subsequent order. We will continue with describing verification rules in the next subsection.

5.2 Model verification by the MULLER framework

The MULLER framework provides a list of correctness properties for multi-level modeling, and corresponding rules to verify these properties. Before continuing on model verification, we wish to clarify its meaning, since there are ambiguous interpretations in the literature. Other works use different terms including consistency and integrity checking, validation, well-formedness and satisfiability [47,24,33,30,41]. In our paper, we support González and Cabot's [30] interpretation and define model verification as the usage of formal methods to achieve model correctness. The scope of models being verified is limited to structural (static) models. The verification rules are aimed to find the inconsistencies in the ISO 15926 specification according to the semantics of the object-oriented modeling and OMLM framework. For example, the verification rules include consistency, well-formedness, integrity and multiplicity constraint checks. They are all expressed as Flora-2 rules. Domain specific (e.g. ISO 15926) correctness properties, such as a pump should contain an impeller, can be assessed by rules.

Taking into account an arbitrary number of classification levels and the size of RDL model (618,403 model elements), model correctness becomes critically important. Here we demonstrate some rules to verify the accuracy of models.

The properties we intent to verify are the cross-level, potency and uniqueness principles, well-formedness (e.g. conformance and completeness) and other properties as listed in Table 3. To verify cross-level relationship we check if they skip or cross the ontological levels.

The conformance and completeness properties are based on the meta-model conformance principle. However, the list is not complete and other perspectives of multi-level semantics [24,32] can be verified. The cross-level verification rules detect an error when these rules succeed, whereas the conformance and completeness, and potency and uniqueness verification rules find an error in case of failure of these rules. The linguist dimension constraints and domain specific verification rules consider both result options (see Table 3). The verification rules can include the integrity constraints of the underlying domain. For example, in the ISO 15926 case, the additional range restriction constraints are represented as first-order logic axioms, which can be directly transferred to Flora-2 rules.
To demonstrate the verification of the properties in Flora-2, the listings for verification rules are provided for some properties in Table 3. The first three rules address cross-level, followed by potency based and violation metrics properties.

Listing 5 represents a verification rule to find classification (instance-of) relationships that cross ontological levels. The level-skipping ontological classification (Property 1 in Table 3), which contradicts multi-level modeling semantics. In the context of the example (Fig. 2), `PUMP` cannot be an instance of `INDUSTRY ASSOCIATION CLASS` or `JORD BACKGROUND ONTOLOGY CLASS`. The rule finds an object (`?X`) that has an ontological class (`?Y`) at more than one (lines 5-6) ontological level above. The negation operator is denoted by

Table 3: Verification properties in multi-level modeling

No	Verification property	Description
		Cross-level verification
1	Level-skipping ontological classification	Ontological classification can only exist between objects at adjacent ontological levels and not if the instance-of relationship crosses more than one level. A model element can only be an instance of an element from the adjacent meta-level above.
2	Cross-level ontological generalization	Generalization relationship connects model elements at the same ontological level.
3	Cross-level ontological composition	A model element can be composed from model elements at the same ontological level.
		Conformance and completeness verification
4	Linguistic conformance	A model element linguistically conforms to its linguistic type. That means it is an instance of its linguistic type.
5	Ontological conformance	A model element conforms to its ontological type, i.e. it is an ontological instance of its type.
6	Ontological model element completeness	A domain specific model consists of a set of elements. An instance of that model should have the instances for all elements in the set. Completeness holds if all elements in the set are instantiated.
7	Ontological level completeness	Similar to model element completeness, except the model element set covers all model elements at an ontological level. Completeness/coverage ratio depends on the number of instantiated model elements at an ontological level.
		Potency and uniqueness verification
8	Unique linguistic instantiation	Each model element in the ontological dimension can only instantiate a single linguistic model element
9	Subsequent potencies	The model potencies in adjacent ontological levels should have subsequent order. This rule is applied only for the transformed models.
10	Generalization involves elements at the same ontological level	Model elements at both ends of a generalization relationship must be at the same ontological level regardless of their potencies.
11	Composition ends are elements at the same ontological level	A composition relationship is allowed between a part and whole elements at the same ontological level.
		Metrics and other verification properties
12	Number of verification errors(violations)	Count the number of occurrences of a particular verification violation
13	Cyclic generalization (subclass) relationship	A violation that represents a sequence of sub-classes that end up with the original superclass.
14	Linguistic dimension constraints	The linguistic constraints of a particular ontological model
15	Domain specific verification properties	Verification rules (constraints) related to model elements in the ontological dimension.

"\+" symbol in Flora-2. Since the bottommost ontological level starts with potency zero, the potency of an element at the next ontological level (line 4) is greater than the current one (line 3 and 5). By executing this verification rule, one can check whether a particular or any model element is connected to an incorrect relationship. It is used to detect inconsistencies in multi-level models, especially the ones that are transformed from two-level models.

```
1 levelSkippingOntolClassification(?X, ?Y):-
2         ?X <:: ?Y,
```

```
3         ?X[potency->?_XP],
4         ?Y[potency->?_YP],
5         (?_YP > ?_XP),
6         \+(?_YP = ?_XP + 1).
```

Listing 5: The verification rule for the level-skipping ontological classification property in Flora-2

Generalization relationship connects model elements at the same ontological level (rule 2 in Table 3). For example, PUMP cannot have generalization and composition relationships with ROTATING MECHANICAL EQUIPMENT (Fig. 2), since both relationships cross ontological levels. The cross-level ontological generalization rule in Listing 6 checks whether this object-oriented principle is violated . The first part (lines 1-3) checks whether ?X and ?Y are connected by generalization(::) and ontological classification (<::) at the same time. The next part checks potencies (lines 6-9). Lines 5-9 deal with the case, where a multi-level model is transformed from a two-level model. Lines 1-3 deal with the case, where a new model created based on a plain multi-level model implies coherent potency, since the potency is decreased in every instantiation of an element.

```
1 clog(?X, ?Y):-
2         ?X :: ?Y,
3         ?X <:: ?Y.
4
5 clog(?X,?Y):-
6         ?X :: ?Y,
7         ?X[potency->?_XP],
8         ?Y[potency->?_YP],
9         \+(?_YP = ?_XP).
```

Listing 6: The verification rule of the cross-level ontological generalization

The cross-level ontological composition rule is similar with Listing 6. It checks whether ?X has a composition relationship (?X[?hasPart->?Y] and ?X[?hasWhole->?Y]) with ?Y across ontological levels. One of the important reasoning features of Flora-2 is that while the violations for specific model elements are detected by providing their names as arguments, the violations of any model elements in the whole knowledge base (model) are identified by using ?X and ?Y variables, without providing their values.

Model elements at both ends of a generalization and composition relationship must be at the same ontological level, regardless of their potencies. In the context of the example, ISO 15926 CLASS OF CLASS and INDUSTRY ASSOCIATION CLASS must be at the same ontological level to be involved in a generalization relationship.

```
1 ?X <:: ?Y :-
2         _isbasefact(?X<::?_Z),
3         ?_Z<::?Y,
4         ?Y \= _object.
5 same_level_generalization(?X,?Y) :-
```

```
6         (?X[potency->?_PX]) :: (?Y[potency->?_PY]),
7         ?X <:: (?_XC[potency->?_PXC]),
8         ?Y <:: (?_YC[potency->?_PYC]),
9         ?_XC :: ?_YC,
10        (?_PXC-?_PX) == (?_PYC-?_PY).
```

Listing 7: The verification rule for generalization (property 10 in Table 3)

Listing 7 checks whether the elements taking part in a generalization relationship (line 6) are at the same ontological level (lines 7-10). In order to detect whether the ends of a generalization relationship are at the same ontological level, the verification rule looks for any generalization relationship between the classifiers of both elements in their ontological classification hierarchy (lines 7-9). If it is found, the rule determines the distances (number of ontological instantiations) to the classifier classes (line 10). If the distances are equal, the involved model elements are at the same level (line 10). If no generalization relationship is found between the classifier classes, then the rule cannot decide whether they are at the same ontological level. For the sake of this verification rule (Listing 7), transitive ontological classification is implemented (lines 1-4). It means when one queries the ontological instances or classes of an element, it returns all elements in the ontological classification hierarchy of that element. The transitive ontological classification can be assigned a different operator in case of need; however we use the default operator in Listing 7 for the sake of simplicity. The unary predicate `_isbasefact` is used to query Flora-2 facts (line 2). The transitive rule fetches all intermediate instances of the top classifier (lines 2-3), except the root Flora-2 `_object` (line 4). The 'not equal to' sign is denoted by '\=' operator in Flora-2 (line 4).

```
1    subsequent potency(?X,  ?Y):-
2         ?X <:: ?Y,
3         ?X[potency->?_PX],
4         ?Y[potency->?_PY],
5         ?_Z is ?_PX + 1,
6         ?_PY \= ?_Z.
```

Listing 8: The verification rule for subsequent potencies of the adjacent elements (property 9 in Table 3)

The subsequent potency verification rule (Listing 8) checks for sequencing between potencies of the elements at the adjacent meta-levels. First it get the potencies (lines 4-5) of the adjacent elements that are related with ontological classification (line 3). Afterwards the potency of the instance is incremented by one (line 6) and the result checked for equality with the potency of its ontological type (line 7).

```
1  ?- ?errors = setof {?_error |
2         subsequent_potency(?_X,?_Y),
3         ?_error=[?_X,?_Y]},
4         ?errors.length@_basetype=?N_violations.
```

Listing 9: Number of verification errors/violations (property 12 in Table 3)

Lastly, Listing 9 demonstrates a query to count number of errors that violate the subsequent potency verification rule.

6 Evaluation

The cost-effective computation and accurate results of transformation and verification rules make them feasible in an industrial context. The evaluation is addressed from five perspectives: performance, scalability, algorithmic improvements, ontological level identification and the inconsistencies within the RDL model in the respective subsections.

The evaluation is performed in the context of the use case. The rule execution and performance analysis were conducted in a regular desktop machine, since our approach can be performed on the local machines of the domain engineers. The machine for evaluation had an Intel Core i5 3.40GHz processor with 8GB of memory running Flora-2 version 0.99.5 on Windows 7 Enterprise. The performance measurements were performed several times (3-4 times) and the resulting average time and memory size was reported.

6.1 Performance

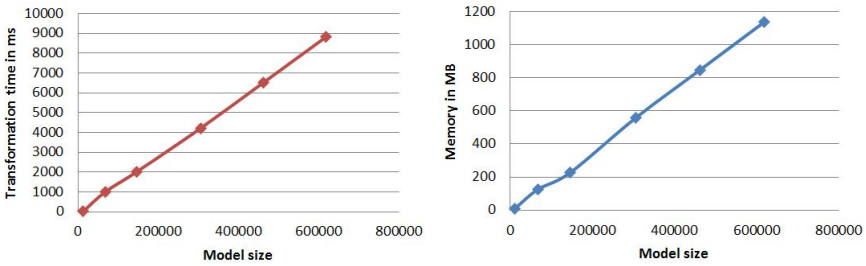
The goal of the performance subsection is to show the feasibility of the implementation in Flora-2 since querying and verification rule implementation in Flora-2 is known to be challenging [17]. The reasoning engine can face memory limitations if the inference features of the engine are not used carefully. The inference engine can be delayed by different causes such as delayed subgoals, recursive predicates, negations and tabling [40]. In addition, we have identified the implementation and performance challenges and addressed them with the efficient algorithmic improvements in Section 6.3.

The performance of our approach has been evaluated based on time and memory consumption of the rules to demonstrate and improve the implementation. We empirically evaluated our approach on the real-world ISO 15926 model with the general performance demonstrated in Table 4. The compilation times do not include the verification time.

The first stage of the execution, the syntactic model transformation, took more time (approximately 2 minutes) to load the model than the actual transformation (8.82 seconds) with reasonable memory consumption (1136 MB). The transformation into the multi-level model took 59 seconds for the classification (44 seconds) and specialization (15 seconds) transformations and only 53 ms to load the RDL facts in memory. Notably, it took 58 minutes to compile the whole static RDL (consisting of 2,885,196 F-Logic facts). The extended compilation or loading time is an implementation limitation of the (Flora-2) compiler, not the framework and is linear in the number of facts (roughly 1.2ms in average to compile a single fact). However the whole RDL model must only be compiled once; specific transformation instances would

Table 4: Empirical evaluation of the use case

Performance criteria	Results
The syntactic model transformation	
Number of processed objects	618,403
Number of processed properties	2,266,793
Time spent to load the model	116 seconds
OWL/RDF to Flora-2 transformation time	8.82 seconds
Memory cost of transformation	1,136MB
The two-level to multi-level model transformation	
Model size	2,885,196 facts
Whole Reference Library(RDL) compile time	58 minutes
Loading time of Flora-2 facts	53 ms
Memory cost of loading	2.5GB
TransformClassification execution time	44 seconds
TransformSpecialization execution time	15 seconds



(a) Transformation time versus model size (b) Memory cost versus model size

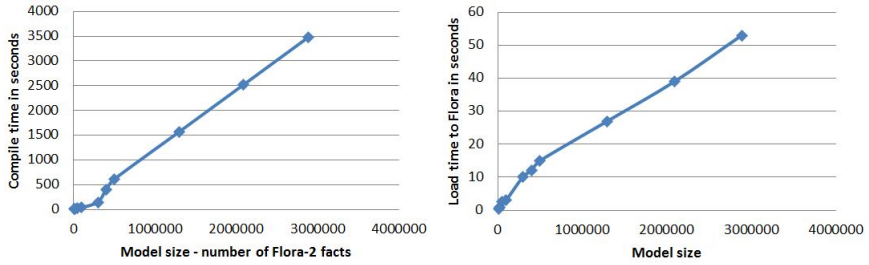
Fig. 9: Transformation time and memory cost increase linearly with model size

be model fragments of smaller size that can rely on the already transformed and compiled RDL for the verification of correctness properties. The reasoners distinguish between load time and querying (reasoning) time if applications frequently operate on preloaded ontologies or models, where load time can be neglected [16].

6.2 Scalability

The scalability of our approach has been evaluated in transformation time and memory consumption. In order to observe the behavior of the measures we have gradually increased the model size as illustrated in Fig. 9. Transformation time and memory usage shows a stable linear behavior as model size increases.

The syntactic transformation performance showed a linear tendency for both transformation time and memory usage. The RDL model that consists of 618,403 objects was transformed into Flora-2 facts in 8.8 seconds.



(a) Compile time in Flora-2 versus model size(b) Loading time in Flora-2 versus model size

Fig. 10: Compile and loading times of the multi-level models

A similar linear trend is observed in the two-level to multi-level transformation as illustrated in Fig. 10. The trend applies to both compilation and execution time. To optimize the memory usage and compile time, the original model has been split into smaller chunks, so that it takes less total time to compile several chunks of the file. The model was split automatically by a data extraction tool, the AWK. The loading time is much faster and shows linear behavior as well.

The memory cost during loading of the multi-level model into Flora-2 shows linear behavior as well, consuming up to 2.5 MB of RAM. The actual transformation to the multi-level model was faster, requiring 44 seconds for the `TransformClassification` and 15 seconds for `TransformSpecialization` transformation rules, consuming 2.5GB of RAM.

6.3 Algorithmic improvements

The verification and transformation rules took reasonable amount of time (upto 44 seconds for TransformClassification rule) when we increased model size. In order to improve the performance of rules further, we decided to detect condition statements that cost more time to process and then to refactor their algorithms. During analysis, we detected time consuming statements by applying the following four algorithmic techniques:

- *Negation avoidance* - Negations that are used in rules and queries consume more time than non-negation statements. The negation operation must consider all possible solutions (state space) in order to verify the negation condition. By avoiding explicit negation operations, the performance has been notably increased.
- *Tabling avoidance* - Generally, tabling (caching) improves the performance of rules and queries. However, when dealing with large models, tabling can take a significant amount of memory. Furthermore, we had limited memory available for experiments, specifically particularly 8 GB of RAM. Since rules and queries table all possible inferences and intermediate results, the

maximum amount of memory can be filled up quickly, which leads to delays in rule executions.
- *Recursion avoidance* - Similar to tabling, recursion generally improves performance, particularly the speed of queries and rules. However, recursive predicates can quickly increase memory usage when they are used with large models. Memory overflow causes reduction in query and rule execution speed. In addition, existence of circular relationships can be checked beforehand.
- *Usage of `_isbasefact()` predicate* - The `_isbasefact()` predicate in Flora-2 considers only the base fact without its inferences. It saves processing time of a rule or a query that would be spent to check inferences. For example, it avoids checking all instances in a classification relationship.

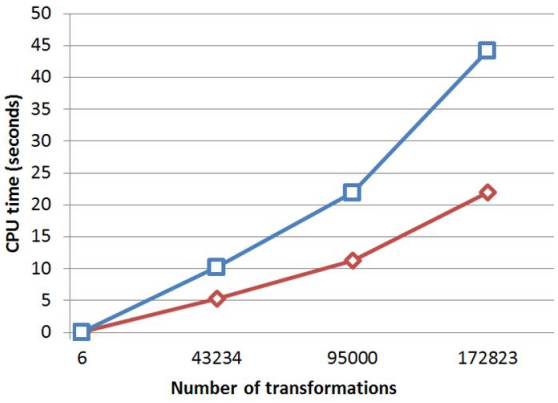


Fig. 11: Transformation rule execution time before (the top line) and after (the bottom line) the algorithmic improvement

The performance of rule and query execution has been significantly improved after applying these four techniques. A particular improvement example of a classification transformation that split the RDL model into ontological levels is illustrated in Fig. 11. It demonstrates rule execution time before and after applying the four optimization techniques. Overall, execution time was roughly halved.

6.4 Level identification

In this subsection we evaluate the two-level to multi-level model transformation. We demonstrate the occurrence of example model elements on particular ontological levels as a result of the transformation in Table 5.

Each entry on the right hand side of the table lists the number of instances of the class labeling the row at particular level. It becomes clear that the lev-

Table 5: Ontological levels and number of elements at each level

	Ontological levels		
Elements	1	2	3
PossibleIndividual	44		
ClassOfIndividual		41,937	
ClassOfClass			14,348
Relationship	118		
ClassOfRelationship		79,226	
ClassOfClassOfRelationship			394

els partition the set of all instances of the classes of the original (two-level) model. For example, there are only 44 instances of `PossibleIndividual` at ontological level 1, since RDL focuses at level 2 or higher. Users of the RDL then instantiate and use these reference data. Most of the elements are located at ontological level 2, which is the most used level of the RDL. `ClassOfClass` instances are located at the top level. Some of them classify other `ClassOfClass` instances by being in circular relationship to themselves (not a result of our modeling decisions but so established by the ISO 15926 standard). The relationships in the RDL are mostly located at level 2, since that level contains the majority of other elements. The relationships establish links within level 2 and between elements of levels 2 and 3. In particular, the `ClassOfClassOfRelationship` links `ClassOfClass` elements together. A small number of `ClassOfClassOfRelationship` versus a large number of `ClassOfClass` indicates that the rest of the `ClassOfClass` elements connect with `ClassOfIndividual` elements.

6.5 Inconsistencies in the RDL (ISO 15926) model

The verification rules are also used to find the inconsistencies in the RDL model. The inconsistencies are the cases which contradicts with the semantic of the OMLM. The following inconsistencies have been found:

- naming inconsistencies in the names of model elements. The `CLASS` and `CLASS OF CLASS` suffixes are not consistent, for example one can come across with the model elements with the CLASS and CLASS OF CLASS suffixes at the same ontological levels (see Fig. 2). Within the OMLM these CLASS suffixes are redundant;
- cyclic subclass relationships;
- cross-level ontological classification relationships.

```
1 /** Results for the cyclic subclass relationship verification **/
2 Found cyclic subclass relationship (?X::?X) => RDS574954291
3 Found cyclic subclass relationship (?X::?X) => RDS22601516315
4 Found cyclic subclass relationship (?X::?X) => RDS16764854
5 Found cyclic subclass relationship (?X::?X) => RDS61133327145
6 Found cyclic subclass relationship (?X::?X) => RDS61133327159
```

```
7 # cyclic subclasses found => 5
```

Listing 10: The results of the cyclic subclass relationship verification rule

The verification results of the cyclic subclass relationship rule (Rule 13, Table 3) are shown in Listing 10. The verification rule has found five RDL elements with cyclic subclass relationships.

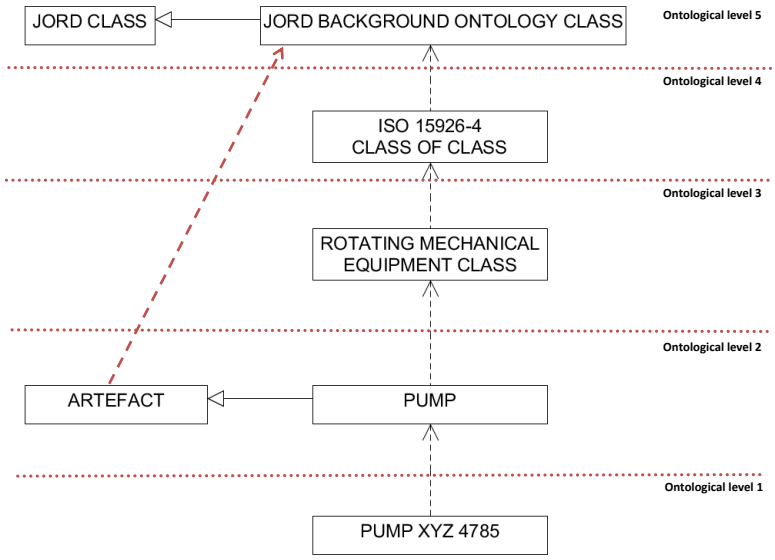


Fig. 12: A level-skipping ontological classification in the RDL

As another example of inconsistency, Fig. 12 demonstrates a violation, detected by the level-skipping ontological classification verification rule (Rule 1, Table 3). In the original RDL model, `ARTEFACT` has an instance-of relationship with `JORD BACKGROUND ONTOLOGY CLASS`. This classification relationship is illegal by our rules, since it is crossing two ontological levels (ontological levels 3 and 4 in Fig. 12). It indicates that `ARTEFACT` element is categorized by `JORD BACKGROUND ONTOLOGY CLASS` ontology. These types of relationships can be renamed as categorization relationships to separate them from classification relationships. These inconsistencies can be shared with the RDL designers, so that more precise and accurate versions will be produced.

7 Related work

Related work has been analyzed from the different research perspectives of this paper. These perspectives are:

1. multi-level modeling for data interoperability;
2. usage of multi-level modeling in different standards and domains;
3. comparison between linguistic dimension(meta-model)s of multi-level approaches;
4. support for transformation in multi-level environment;
5. multi-level modeling frameworks;
6. model verification approaches.

The perspectives are addressed in the respective subsections.

7.1 Multi-level modeling for data interoperability

Within the context of this paper, the scope of the concept of interoperability is considered at three levels of granularity: (1) interoperability requirements and solutions, (2) interoperability in MDE and (3) multi-level modeling for data interoperability.

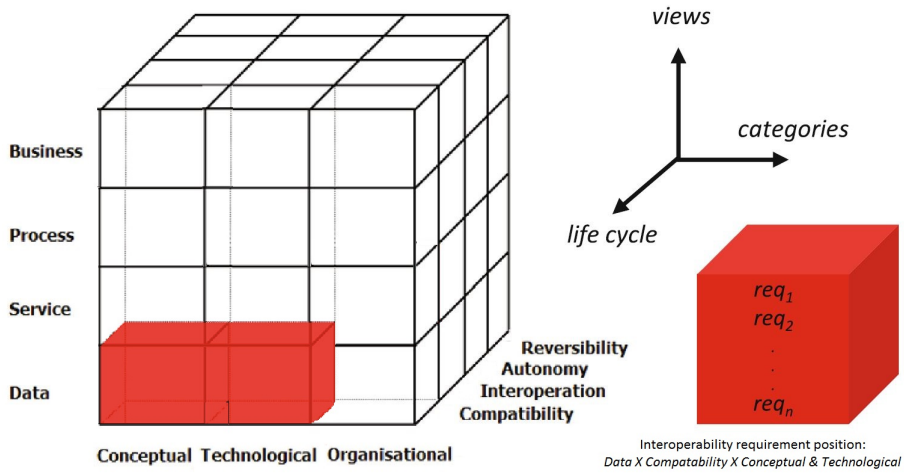


Fig. 13: The positioning of our approach within the framework for enterprise interoperability requirements - adapted from [22]

The first level was analyzed by a framework for enterprise interoperability requirements [22]. The framework analyzed the interoperability in three dimensions: the views, categories and life cycles as illustrated in Fig. 13. The

views represent the impact domains (levels) of a problem of interoperability. These are the enterprise domains, namely business, process, service and data. The categories represent the abstraction levels of the interoperability, which are conceptual, technological and organizational. The last life cycle dimension represents the maturity levels of interoperability: compatibility, interoperation, autonomy and reversibility. Interoperability requirements are formed based on these dimensions. Our approach is positioned at the "data × compatibility × conceptual and technological" coordinate as illustrated in Fig. 13. Our view is focused on data interoperability. We cover both the conceptual and technological categories by means of the OMLM and MULLER frameworks, and the implementation in Flora-2. At the moment, our approach is positioned at the compatibility stage of the interoperability life cycle (axis), where the compatibility between different representations of the ISO 15926 standard is considered.

The second level of granularity is the interoperability within MDE. Recent review work [29] has systematically analyzed interoperability approaches in the MDE community. The authors consider the previously identified interoperability features for model-driven development processes, such as the management of heterogeneity and interoperability verification. In addition, they extracted features such as meta-extensions, model-weaving and pivot artifact. These features are used to design MDE oriented interoperability framework, where the interoperability levels are defined based on the abstraction levels, namely specification and execution. These interoperability levels correspond to the categories dimension of the framework for enterprise interoperability requirements [22]. The pivot artifact is important, since its usage corresponds to 72,7% of the total studies analyzed [29]. It is an intermediate artifact (a meta-model or ontology) to perform the interchange of information. The authors suggested to use the meta-model pivots to manage the structural heterogeneities and consider usage of ontologies for discovering semantic equivalences. As suggested, our approach uses (multi-level) meta-model pivots to manage structural hierarchies of the OGI Pilot.

Interoperability by means of multi-level modeling is the most granular level of the interoperability within the context of this research. The Open Data as a Service (ODaaS) approach uses multi-level modeling to construct open data applications [51]. The architecture of the approach consists of a set of domain models and meta-models, a library of 'injectors' to import data from heterogeneous sources, and a REST-based API to provide access to the models for users or domain specific applications. The domain descriptions are the classification of concepts and their successive refinements by means of the multi-level modeling. The data modeling is organized based on the generic and domain (meta-)models. The challenges in definitions of data domains and categories are resolved by multi-level modeling. The approach uses the MetaDepth tool [44] to support multi-level modeling. Similarly, our approach also applies multi-level modeling to data interoperability in the OGI Pilot. It reorganizes the domain knowledge by means of the OMLM framework to simplify the domain knowledge.

7.2 Usage of multi-level modeling in different standards and domains

In order to offer multi-level modeling as a solution in a specific case, one needs to know when and how to apply it in practice. In de Lara et al. [46], the authors determined five different multi-level patterns to analyze their occurrences in the meta-models from different domains to decide whether the application of multi-level modeling is beneficial. These patterns are the type-object, dynamic feature addition, dynamic auxiliary domain concepts, relation configurator and element classification. The majority of the pattern occurrences are found in the software architecture and enterprise/process modeling domains. As an example for the latter domain, multi-level modeling is suggested as the best solution for language extension in [6]. It addresses an ongoing extension of languages and services to support the continuously evolving customized views of enterprise systems. The mentioned software engineering domains are the traditional example and application domains. Multi-level modeling has the potential to be used not only in traditional application areas, such as software architecture and enterprise modeling, where many problems are intrinsically multi-level [46], but also in non-traditional domains, such as in the oil and gas industry. Our approach exemplifies the application of multi-level modeling to the latter non-traditional domain. Similar to the software engineering domains, the underlying ISO 15926 (meta-)model contains the type-object patterns. The rest of the patterns are not found in ISO 15926, however, tool support can be provided by Flora-2 by introducing particular rules.

7.3 Comparison between linguistic dimensions (meta-models) of multi-level approaches

The linguistic dimensions used by the various multi-level modeling approaches differ in a variety of ways. The linguistic dimension of the OMLM (Fig. 3) was inspired by the dimensions of the popular multi-level modeling tools, Melanee [4] and MetaDepth [44]. In particular, single (attribute-like) and multi-potency (clabject-like) elements correspond to the respective elements in the deep meta-modeling approach [44, 54]. A Clabject can be instantiated a single time at the specified metalevel and a number of times that is defined by potency in case of the single and multi-potency elements respectively. Accordingly, we have introduced non-potency elements, such as generalization and classification, which can be instantiated at any ontological level. The rest of the concepts of the linguistic dimension was inspired from the Melanee approach [39].

The Pan Level Model (PLM) [39] contains an additional 'SetRelationship' with the equality, inversion and complement subclasses. These set relationships are supported by Flora-2, however they were not considered in the linguistic dimension of the OMLM. An alternative methodological approach to the Clabject is the powertype that uses standard classes both conventionally and as meta-classes [31].

Another interesting work approaches multi-level modeling from the language engineering perspective [21]. The authors claim that the current multi-level concepts such as the Clabjects and Powertypes are not complete as a basis for meta-modeling. In addition, the authors suggest to address several features: meta-circularity (self description), uniformity and extensibility. Based on these features they propose the Kernel language, which is based on representing everything as objects, the same as our Flora-2 implementation.

Another approach unifies the linguistic and ontological dimensions [24] for the sake of conformance checking. The linguistic dimension is positioned as a top ontological level that allows checking all model elements as the ontological conformance. Our approach differs in the way that it introduces the realization dimension instead of shifting the linguistic dimension on top of the ontological one. The realization dimension can be seen as an analogy of the tables in a database that store the linguistic and ontological information. The separate realization dimension has the following benefits:

- the implementation perspective is represented in the OMLM framework, while other approaches do not consider it in their multi-level meta-model;
- ability to specify the Clabject instantiation, e.g. as Java or Flora-2 objects;
- direct (realization) instantiation of the object;
- the realization dimension conceptually refers to an implementation language. It provides a possibility to compare with other linguistic dimensions (meta-models) by changing the linguistic dimension of OMLM;
- decouples the implementation language and different language implementations can be realized.

7.4 Support for transformation in multi-level environments

Transformations are supported in a multi-level modeling environment by extending two-level model transformations, such as ATL (ATLAS Transformation Language) and ETL (Epsilon Transformation Language). The syntax and semantics of the languages are extended to support the multi-level perspective. Use of ATL for multi-level concepts is realized by means of so-called ADL adapter that supports modeling and access to the linguistic and ontological model elements separately [7]. The ETL is extended with similar syntax and rule matching semantics to access the linguistic and ontological types, attributes and indirect ontological types [45]. Moreover, the ETL extension approach considers different transformation schemes, such as the multi-level target, refining and linguistic transformations. Similarly, our approach implements the OMLM framework in Flora-2 to support multi-level modeling. We introduced operators for the linguistic and ontological classification (see Section 4). Each model element can access its linguistic and ontological types and instances by means of the operators. Flora-2 rules have been used as transformation rules in the two-level to multi-level model transformation. In this paper we have focused on the transformations between two-level and multi-level domains, because of nature of the underlying RDL model. However, the Flora-2

implementation also allows to perform transformations between multi-level models.

The transformation rules in Flora-2 are not yet full-fledged transformations, like ATL or ETL. We are working towards supporting different customized features of transformations. The transformation rules of our approach are injective and can match with several parameters in the source model. Another important feature is the verification and querying on the transformation rules as explained in Listing 11.

7.5 Multi-level modeling frameworks

We have analyzed the respective multilevel modeling frameworks based on four criteria (see Table 6): (1) Querying support, (2) Verification support, (3) Integrated framework(a single framework for modeling, transformation, querying and verification), (4) Tool support and (5) Syntax type (graphical and textual).

A recent review paper [30] on formal verification of static software models in MDE distinguishes two phases in model verification: formalism and reasoning. In this paper we consider both, where formalism corresponds to OMLM and reasoning is supported by querying and verification in the MULLER framework.

The architectural approach for data integration shown in [20] supports querying at the conceptual level and defines the integrated framework at the architectural level. The Level-Agnostic Modeling Language (LML) [10] provides a foundation for querying and verification by a unified and simplified reasoning and model checking. The F-Logic based Object Modeling Language (F-OML) supports almost all evaluation criteria with a limited multi-level modeling support, with only classification and instantiation. The Dual Deep Instantiation (DDI) approach [52] supports multi-level modeling principles as well. Its ConceptBase implementation addresses meta-modeling and verification of constraints. The similarity with our approach is that they support implementation, querying and verification in a single framework. The main difference of MULLER is that we use a single language framework, where ConceptBase uses different languages in their framework. A formalization of deep meta-modeling is thoroughly addressed by Rossini et al. [54], however the querying is based on OCL constraints (without support for higher order reasoning) and its verification is performed with EVL, an implementation of OCL. OCL and EVL are different languages than that used for the formalization. While both Melanee and Diagram Predicate Framework (DPF) tools use graphical (visual) syntax, the rest of the tools use textual syntax.

OCL represents first order logic. In Flora-2, we can use higher-order reasoning together with first order reasoning. The high-order reasoning is demonstrated from two perspectives: multi-level modeling semantics and reasoning on the verification rules.

(1) OCL is not aware of the distinction between linguistic and ontological classification and the ontological levels, i.e. OCL and EVL cannot access a specific model element at the certain ontological level. In contrast MULLER (Flora-2) can access any ontological/linguistic model element at any ontological/linguistic level directly by its name or by its relationship. For example, at the `PUMP` class level (see Fig. 2), one can easily access the properties of the `INDUSTRY ASSOCIATION CLASS` by a statement:

```
INDUSTRY ASSOCIATION CLASS[hasDefinition->?X].
```

where `?X` returns the value of `hasDefinition` property. The statement can be used in queries and verification rules. OCL needs to be extended to support multi-level modeling semantics. One recent example is DeepOCL [38].

(2) The second perspective represents the querying and verification on the verification rules. Flora-2 allows to query rules themselves in the rule base. That provides an ability to verify the verification or transformation rules, as illustrated in Listing 11.

```
1  ?- clause{%TransfClassif(?X,?Y,?SM,?TM), (?_[?_->?X,?_->?Y]@?SM, ?Z)}.
2
3  ?X = ?_h7597
4  ?Y = ?_h7602
5  ?SM = ?_h7607
6  ?TM = ?_h7612
7  ?Z = (${flora_put_attr(?_h7625,fldynrulevarcheck,
8  ['?Classified' = ?_h7597, '?Classifier' = ?_h7602])@_prolog(flrwhen)},
9  ${insert{?_h7597<::?_h7602@_h7612}})
10
11 1 solution(s) in 0.0000 seconds
```

Listing 11: Querying the TransfClassif transformation rule

The rule base of Flora-2 is queried by means of `clause(head,body)` statements as illustrated in Listing 11. Line 1 represents a query and the rest (lines 3-9) shows the result of the query. The head and body use variables and patterns to match the verification rule (line 1). The query queries for the `TransfClassif` transformation rule to find out whether the classified and classifier properties (lines 3-4 and 8) are transformed to relevant variables (line 9). The variable names `?_h7597` and `?_h7602` are used for both properties (lines 8) and variables (line 9), which indicate the properties in the original model will be transformed to variables in the ontological classification relationship (line 9). By means of such queries, the framework verifies the validity of a transformation rule before its execution.

7.6 Model verification approaches

The verification of (multi-level) models has been analyzed from four perspectives (see Table 7): properties, multi-level support, verification, implementation/tool support.

Table 6: Comparative evaluation of multi-level approaches

Approaches / Criteria	**Querying support**	**Verification support**	**Integrated framework**	**Tool support**	**Syntax type**
Information integration [20]	conceptual representation	X	at architectural level	X	X
Melanee and LML [10]	OCL queries	OCL	Melanee	LML & ATL	graphical (EMF)
F-OML & PathLP [15]	✓	✓	F-Logic	PathLP	textual
A formalization based on DPF [54]	OCL queries	Epsilon Validation Language (EVL)	MetaDepth	EOL ETL EGL	textual
DDI [52]	✓	axioms & constraints in ConceptBase	✓	ConceptBase	textual
A Diagrammatic Approach [43]	OCL queries	OCL	DPF	DPF	graphical (EMF)
OMLM	first & higher-order reasoning	✓	Flora-2	Flora-2	textual

Several approaches have addressed verification of correctness properties, in particular satisfiability (of certain axioms) [32, 52, 19] and conformance [55, 24]. Guerra and de Lara [32] list conformance based verification properties as strong and weak satisfiability, and liveliness of a class. The approach by [24] supports ontological and linguistic conformance. Similarly, MULLER can also support both types of conformance and meta-model completeness. Alternatively, different types of inconsistencies have been verified between UML class, state-chart and sequence diagrams [25]. Our goal is not to verify the consistency between different representations of the same model, but to verify the syntactic (by meta-models) and semantic (by verification rules) correctness of a model.

Different verification techniques are used for verification of correctness properties. The approaches use model finders, constraint programming systems, Datalog, Answer Set Programming (ASP), and some of them are implemented in constraint languages such as OCL and Weighted Constraint Rule Language (WCRL). Similarly, we have used Flora-2 for both formal specification and verification. Multi-level modeling is supported in the recent approaches [32, 55, 24, 52] as illustrated in Table 7.

Table 7: Model verification approaches

Approach	**Correctness properties**	**MLM support**	**Verification technique**	**Tool support**
Integrity constraints [32]	satisfiability	✓	relational logic, model finder (SAT solver)	USE Validator
Incremental consistency check [24]	linguistic, ontological conformance and evolution	✓	constraint language	Model/Analyzer
NIVEL [2]	check that formalisation matches the notion of valid model	✓	weight constraint rule language (WCRL)	WCRL
UML/OCL constraint programming [19, 18]	weak and strong satisfiability, lack of constraint redundancies & subsumptions, liveliness of a class	X	Constraint solver, Constraint Satisfaction Problem(CSP)	ECLiPSe constraint programming system
UML/OCL Boolean satisfiability [57]	consistency of system states & redundancy of OCL constraints	X	constraint language	SAT solver
ConceptBase [52]	Satisfiability (of the axioms)	✓	Datalog-neg	Datalog
CARE [55]	conformance	✓	meta-modeling	Answer Set Programming(ASP)
MULLER	conformance, inconsistencies (see Table 3)	✓	F-Logic & meta-modeling	Flora-2

8 Conclusion

In this paper, we have presented an integrated multi-level modeling approach that integrates semantics specification and verification in the respective OMLM and MULLER frameworks. The integration was strengthened by using a single language (Flora-2) for multi-level conceptualization, matching and transformation, and querying and verification phases of data interoperability. The approach was evaluated in a real-life industry (OGI Pilot) project in the oil and gas domain.

The OMLM framework is applicable to standard data model specifications and demonstrated the benefit of applying multi-level modeling principles to ordinary two-level data models. Compared to existing multi-level modeling approaches, it contains an additional realization dimension which captures the implementation perspective of the framework. This dimension achieved a de-coupling from an implementation language and increased re-usability. Furthermore, we developed a workflow that describes how the framework can be easily applied. We have implemented the framework in the F-logic dialect -

Flora-2 and evaluated the framework on the ISO 15926 standard. The evaluation showed clearly the feasibility of the approach. The complexity of the ISO 15926 standard has been reduced by improving the clarity in:

- linguistic and ontological model elements;
- ontological levels;
- class naming;
- relationships between ontological model elements.

The reasoning capabilities of Flora-2 in combination with the presented rules to verify correctness properties of multi-level models helped to identify inconsistencies in the ISO 15926 standard from a software engineering point of view and turned out to be a practical verification tool for software development of the standard. The approach contributes to tool support of multi-level modeling community and shows its practical applicability.

This research will serve as a base for future studies to investigate querying and verification of the rule base, which can lead to verification of transformation and verification rules themselves. On the industrial side, we plan to apply the approach to the operations and maintenance standard, OSA-EAI.

Acknowledgements This research was supported by the South Australian Premier's Research and Industry Fund.

References

1. Juergen Angele, Michael Kifer, and Georg Lausen. Ontologies in F-Logic. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 45–70. Springer Berlin Heidelberg, 2009.
2. Timo Asikainen and Tomi Männistö. Nivel: a metamodelling language with a formal semantics. *Software & Systems Modeling*, 8(4):521–549, 2009.
3. Colin Atkinson. Meta-modelling for distributed object environments. In *Enterprise Distributed Object Computing Workshop [1997]. EDOC'97. Proceedings. First International*, pages 90–101. IEEE, 1997.
4. Colin Atkinson and Ralph Gerbig. Melanie: multi-level modeling and ontology engineering environment. In *Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards*, page 7. ACM, 2012.
5. Colin Atkinson and Ralph Gerbig. Level-Agnostic Designation of Model Elements. In *Proc. of ECMFA 2014*, volume LNCS 8569, pages 18–34. Springer, 2014.
6. Colin Atkinson, Ralph Gerbig, and Mathias Fritzsche. A multi-level approach to modeling language extension in the enterprise systems domain. *Information Systems*, 2015.
7. Colin Atkinson, Ralph Gerbig, and Christian Vjekoslav Tunjic. Enhancing classic transformation languages to support multi-level modeling. *Software & Systems Modeling*, 14(2):645–666, 2013.
8. Colin Atkinson, Georg Grossmann, Thomas Kühne, and Juan de Lara, editors. *Proceedings of the Workshop on Multi-Level Modelling co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2014)*, volume 1286 of *CEUR Workshop Proceedings*, 2014.
9. Colin Atkinson, Georg Grossmann, Thomas Kühne, and Juan de Lara, editors. *Proceedings of the Workshop on Multi-Level Modelling co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2015)*, 2015.

10. Colin Atkinson, Bastian Kennel, and Björn Goß. The level-agnostic modeling language. In Brian Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering*, pages 266–275. Springer, 2011.
11. Colin Atkinson, Bastian Kennel, and Björn Goß. Supporting constructive and exploratory modes of modeling in multi-level ontologies. In *Procs. 7th Int. Workshop on Semantic Web Enabled Software Engineering, Bonn (October 24, 2011)*, 2011.
12. Colin Atkinson and Thomas Kühne. Rearchitecting the UML infrastructure. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 12(4):290–321, 2002.
13. Colin Atkinson and Thomas Kühne. Model-driven development: a metamodeling foundation. *Software, IEEE*, 20(5):36–41, 2003.
14. Colin Atkinson and Thomas Kühne. Reducing accidental complexity in domain models. *Software and System Modeling*, 7(3):345–359, 2008.
15. Mira Balaban and Michael Kifer. Logic-based model-level software development with F-OML. In *Model Driven Engineering Languages and Systems*, pages 517–532. Springer, 2011.
16. Jürgen Bock, Peter Haase, Qiu Ji, and Raphael Volz. Benchmarking OWL reasoners. In *Proc. of the ARea2008 Workshop, Tenerife, Spain (June 2008)*, 2008.
17. Felix Burgstaller, Dieter Steiner, Michael Schrefl, Eduard Gringinger, Scott Wilson, and Sam van der Stricht. AIRM-based, fine-grained semantic filtering of notices to airmen. In *Integrated Communication, Navigation, and Surveillance Conference (ICNS), 2015*, pages D3–1. IEEE, 2015.
18. Jordi Cabot, Robert Clarisó, and Daniel Riera. UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models Using Constraint Programming. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 547–548, New York, NY, USA, 2007. ACM.
19. Jordi Cabot, Robert Clarisó, and Daniel Riera. On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*, 93:1–23, 2014.
20. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Cooperative Information Systems, 1998.*, pages 280–289. IEEE, 1998.
21. Tony Clark, Cesar Gonzalez-Perez, and Brian Henderson-Sellers. A foundation for multi-level modelling. In *MULTI 2014–Multi-Level Modelling Workshop Proceedings*, page 43, 2014.
22. Nicolas Daclin and Sihem Mallek-Daclin. Towards a sustainable implementation of interoperability solutions: Bridging the gap between interoperability requirements and solutions. In *Enterprise Interoperability*, volume 213 of *Lecture Notes in Business Information Processing*, pages 73–82. Springer Berlin Heidelberg, 2015.
23. Juan de Lara, Esther Guerra, Ruth Cobos, and Jaime Moreno Llorena. Extending deep meta-modelling for practical model-driven engineering. *The Computer Journal*, 57(1):36–58, 2014.
24. Andreas Demuth, Markus Riedl-Ehrenleitner, and Alexander Egyed. Towards Flexible, Incremental, and Paradigm-agnostic Consistency Checking in Multi-level Modeling Environments. In *MULTI 2014–Multi-Level Modelling Workshop Proceedings*, page 73, 2014.
25. A. Egyed. Automatically detecting and tracking inconsistencies in software design models. *Software Engineering, IEEE Transactions on*, 37(2):188–204, March 2011.
26. Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and Markus Zanker. Configuration knowledge representation for semantic web applications. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, 17(1):31–50, January 2003. Special issue on Configuration.
27. Fiatech. Advancing Interoperability for the Capital Projects Industry: A Vision Paper. Technical report, Fiatech, February 2012.
28. M. P. Gallaher, A. C. O'Connor, Jr. Dettbarn, J. L., and L. T Gilday. Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry. Technical report, NIST, 2004.
29. Giovanni Giachetti, Francisco Valverde, and Beatriz Marín. Interoperability for model-driven development: Current state and future challenges. In *Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on*, pages 1–10. IEEE, 2012.

30. Carlos A González and Jordi Cabot. Formal verification of static software models in MDE: A systematic review. *Information and Software Technology*, 56:821–838, 2014.
31. Cesar Gonzalez-Perez and Brian Henderson-Sellers. A powertype-based metamodelling framework. *Software & Systems Modeling*, 5(1):72–90, 2006.
32. Esther Guerra and Juan de Lara. Towards Automating the Analysis of Integrity Constraints in Multi-level Models. In *MULTI 2014 - Multi-Level Modelling Workshop Proceedings*, page 63, 2014.
33. Ramzi A Haraty, Mirna F Naous, and Azzam Mourad. Assuring consistency in mixed models. *Journal of Computational Science*, 5(4):653–663, 2014.
34. Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.
35. Muzaffar Igamberdiev, Georg Grossmann, and Markus Stumptner. An implementation of multi-level modelling in F-logic. In *Proc. of the Workshop on Multi-Level Modelling (MULTI14) co-located with MoDELS 2014*, volume 1286 of *CEUR*, pages 33–42, 2014.
36. ISO. ISO 15926: Industrial automation systems and integration — Integration of lifecycle data for process plants including oil and gas production facilities. Technical report, ISO, 2004.
37. Andreas Jordan, Georg Grossmann, Wolfgang Mayer, Matt Selway, and Markus Stumptner. On the application of software modelling principles on ISO 15926. In *Proc. of the Modelling of the Physical World (MOTPW) Workshop at MODELS 2012*. ACM, 2012.
38. Dominik Kantner. Specification and Implementation of a Deep OCL Dialect. Master's thesis, Department of Business Informatics and Mathematics Chair of Software Engineering, 2014.
39. Bastian Kennel. *A unified framework for multi-level modeling*. PhD thesis, University of Mannheim, 2012.
40. Michael Kifer, Guizhen Yang, Hui Wan, Chang Zhao, Polina Kuznetsova, and Senlin Liang. Flora-2: User's Manual. *Flora*, 2:4, 2013.
41. Soon-Kyeong Kim and David Carrington. A formal mapping between uml models and object-z specifications. In *ZB 2000: Formal Specification and Development in Z and B*, pages 2–21. Springer, 2000.
42. Mathias Kleiner, Patrick Albert, and Jean Bézivin. Parsing sbvr-based controlled languages. In *Proceedings 12th International Conference on Model Driven Engineering Languages and Systems MODELS 2009*, pages 122–136, Denver, CO, October 2009.
43. Ole Klokkhammer. A diagrammatic approach to deep metamodelling. Master's thesis, Department of Informatics University of Bergen, 2014.
44. Juan Lara and Esther Guerra. Deep Meta-Modelling with MetaDepth. In *TOOLS 2010*, volume LNCS 6141, pages 1–20. Springer, 2010.
45. Juan Lara, Esther Guerra, and Jesus Sanchez Cuadrado. Model-driven engineering with domain-specific meta-modelling languages. *Springer SoSyM*, 2013.
46. Juan De Lara, Esther Guerra, and Jesús Sánchez Cuadrado. When and how to use multilevel modelling. *ACM TOSEM*, 24(2):12, 2014.
47. Francisco J Lucas, Fernando Molina, and Ambrosio Toval. A systematic review of uml model consistency management. *Information and Software Technology*, 51(12):1631–1645, 2009.
48. Wolfgang Mayer, Peter Killisperger, Markus Stumptner, and Georg Grossmann. A declarative framework for work process configuration. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, 25(2):145–165, 2011.
49. Wolfgang Mayer, Markus Stumptner, Georg Grossmann, and Andreas Jordan. Semantic Interoperability in the Oil and Gas Industry: A ChallengingTestbed for Semantic Technologies. In *AAAI 2013 Fall Symposium on Semantics for Big Data*, 2013.
50. MIMOSA. Open Systems Architecture for Enterprise Application Integration (OSA-EAI) 3.2.3. Technical report, MIMOSA, 2012.
51. A. Mora Segura, J. Sanchez Cuadrado, and J. De Lara. ODaaS: Towards the model-driven engineering of open data applications as data services. In *Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International*, pages 335–339, Sept 2014.
52. Bernd Neumayr, Manfred A. Jeusfeld, Michael Schrefl, and Christoph Schätz. Dual Deep Instantiation and Its ConceptBase Implementation. In *Proc. of CAiSE 2014*, LNCS 8484, pages 503–517. Springer, 2014.

53. Bernd Neumayr, Michael Schrefl, and Bernhard Thalheim. Modeling techniques for multi-level abstraction. In *The Evolution of Conceptual Modeling*, pages 68–92. Springer, 2011.
54. Alessandro Rossini, Juan de Lara, Esther Guerra, Adrian Rutle, and Uwe Wolter. A formalisation of deep metamodelling. *Formal Aspects of Computing*, 26(6):1115–1152, 2014.
55. Johannes Schönböck, Angelika Kusel, Jürgen Etzlstorfer, Elisabeth Kapsammer, Wieland Schwinger, Manuel Wimmer, and Martin Wischenbart. CARE – A Constraint-Based Approach for Re-Establishing Conformance Relationships. In *APCCM 2014*, CRPIT Vol.154, pages 19–28. ACS, 2014.
56. Matt Selway, Wolfgang Mayer, and Markus Stumptner. Semantic interpretation of requirements through cognitive grammar and configuration. In *Proc. Pacific Rim Conference on Artificial Intelligence (PRICAI) 2014*, volume LNCS 8862, pages 496–510. Springer, 2014.
57. Mathias Soeken, Robert Wille, Mirco Kuhlmann, Martin Gogolla, and Rolf Drechsler. Verifying uml/ocl models using boolean satisfiability. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1341–1344, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.
58. Markus Stumptner, Gerhard Friedrich, and Alois Haselböck. Generative constraint-based configuration of large technical systems. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, 12(4):307–320, December 1998.

Author Biographies



Muzaffar Igamberdiev is a PhD candidate in the School of IT and Mathematical Sciences at the University of South Australia. His research interests are mainly related to Model Driven Engineering(MDE), particularly model transformations, metamodelling, model verification and multi-level modeling.



Georg Grossmann is a lecturer in the School of IT and Mathematical Sciences at the University of South Australia. He is working on the integration of business processes and complex data structures for systems interoperability and has applied theory successfully in industry projects. He received a PhD from UniSA in 2008 which received the Ian Davey Research Thesis price for the most outstanding PhD thesis. Current research interests include integration of service processes, ontology-driven integration and distributed event-based systems. He is currently Co-Chief Investigator in the Data to Decisions CRC (D2D CRC) and an SA Government funded project on "software interoperability for the oil and gas sector".



Matt Selway is a research fellow in the School of IT and Mathematical Sciences at the University of South Australia. His research interests are Software Engineering and Artificial Intelligence, particularly Natural Language Processing.



Markus Stumptner is a professor in the School of IT and Mathematical Sciences at the University of South Australia. His research area sits at the junction of different computing disciplines across software engineering (meta-modeling), data and process management, and Artificial Intelligence. He is director of the Advanced Computing Research Centre (ACRC) and program leader for Data Storage and Management in the Data to Decisions CRC.