# Applying Multi-Level Modeling to Data Integration in Product Line Engineering

Damir Nešić
*ITM/MMK/MDA*
*Royal Institute of Technology*
Brinellvägen 83, Stockholm, Sweden
damirn@kth.se

Mattias Nyberg
*ITM/MMK/MDA*
*Royal Institute of Technology*
Brinellvägen 83, Stockholm, Sweden
matny@kth.se

***Abstract*—Developing systems according to the *Product Line Engineering* (PLE) paradigm is a process in which different types of engineering artifacts are created with the aim of reusing them in different configurations of the same system. Ensuring that different system configurations satisfy various functional and non-functional properties is ensured by analyzing different artifacts but because they are maintained by different tools, sometimes even manually, achieving and especially automating such analyses is a challenging task. Overcoming this issue can be achieved through *data integration* of existing data which implies creating an *information model* that specifies how will the existing data fragments be related, captures relevant domain constraints, and most importantly captures the fact that some data objects are classes in one tool and instances in another. This paper reports on the experiences from applying the *Multi-Level conceptual Theory* (MLT), to the problem of information modeling for data integration in the PLE context. Being a *Multi-Level Modeling*, *powertype-based* framework, MLT allows separation of the class and instance facet of modeled entities while keeping them explicit. Some of the MLT modeling constructs are particularly useful for capturing the refinement levels of the modeled artifacts and for succinctly capturing constraints like *disjointess* or *completeness* among them. The paper also reports certain aspects of the studied case that could not be expressed using MLT. The studied case comes from a real data-integration project from the heavy vehicle manufacturer, Scania CV AB.**

***Index Terms*—multi-level modeling, information modeling, product line engineering, linked data**

## I. Introduction

Development and evolution of safety-critical *Software-Intensive Systems* (SIS) is a process involving various engineering disciplines that produce engineering artifacts across the complete SIS lifecycle, e.g. software, hardware, various models, and documents. Performing different analyses across the lifecyle is essential for sucesfull SIS development and deployment, e.g. is versioning or tracing data consistent, or more importantly, does SIS conform to prescribed safety or reliability standards. Automating such analyses is often a daunting task [21] because individual artifacts are isolated in individual tools that use own principles and technologies for artifact management.

Development of *highly configurable SIS*, also known as *Product Line Engineering* (PLE) [19], brings additional difficulties. The goal of PLE is simultaneous development of a multitude of different product configurations. In other words, an additional challenge is to access and manage the data that allows contextualizing the analysis operations with respect to possible product configurations. Similarly to the artifacts data, it is common that the data describing the applicability of an artifact to a particular product configuration is scattered across the lifecycle in different forms [4], [16].

Enabling access to the data necessary for designing different analyses can be achieved through *data integration* [9] techniques that can be used to extract and integrate existing artifacts data from individual tools without modifying the existing tool landscape or enterprise processes. Next to the traditional approaches for data integration [9], [12] suported by technologies like *relational databases* [13] and *ER* or *UML* information models [13], the idea of *Linked Data* (LD) [5] has in recent years been applied to the problem of semantic data integration on Internet, but also in different engineering domains, primarily through *OSLC standards* [25].

As in any data integration scenario, the integrated data must conform to an *information model* that in the case of LD can be created using *RDFS* or *OWL* languages [24]. There are two main aspects of these languages that make their use in an enterprise context difficult. Firstly, both languages are based on the *Open-World Assumption* (OWA) [6] which may lead to inconclusive results of data analysis. Secondly, in data integration scenarios, some entities are simultaneously both classes and objects [18] and this is not expressible in OWL while RDFS can express this fact but without any constraints that prevent creation of inconsistent or contradictory information models. As an alternative to RDFS and OWL, but also to traditional *MOF-compliant* modeling frameworks that face the same issue with entities being both classes and instance, work in [17] has investigated the applicability of the *Multi-Level Modeling* (MLM) paradigm [2], [15] for information modeling in the PLE context. More specifically, the *Multi-Level conceptual Theory* (MLT) [8] was used to interpret PLE-specific concepts inside MLT framework in order to support modeling of different artifacts and their configuration data with an emphasis on semantically rich and correct models.

The present paper reports on the experiences from applying the framework presented in [17] on the case of information-

This work was funded by the ITEA 14014 ASSUME project with the support from Scania CV AB.

model creation for LD-based data integration in the real industrial PLE context from the heavy vehicle manufacturer, Scania CV AB. This report contributes to the field of MLM in two ways: firstly, the considered case comes from a real, large-scale data-integration project for safety-critical SIS development, thus contributing to the limited knowledge about MLM paradigm applicability in the industrial setting; secondly, the information model is intended for data integration in a PLE context based on LD principles which a is novel application for MLM approaches.

The rest of the paper is organized as follows. Section II presents relevant PLE and LD concepts followed by a brief description of the MLT framework. Section III presents the details of the created information model and Section III-B presents its use in the data integration process. Section IV discuses the benefits and shortcomings of the applied modeling framework and is followed by Section V that surveys related work. Finally, Section VI concludes the paper.

## II. Background

This section introduces the PLE and LD concepts followed by a brief explanation of the MLT framework and PLE concepts described in [17].

### A. Product Line Engineering

Figure 1 exemplifies the overall idea of PLE. The main goal of PLE is to engineer artifacts that realize or describe a product in a way such that these artifacts can be reused in several different product configurations. Capturing artifact reuse is achieved by representing different product configurations in terms of configuration options, also known as *features* [19]. For example, an individual truck configuration could be described as having features: engine, brakes, cab etc. The features and their mutual dependencies are captured in a *variability model* [19], in the case of features known as the *feature diagram* [3], that captures all possible product configurations in terms of features. This phase of PLE is known as *domain engineering*. Left part of Figure 1 shows a fragment from a feature diagram. An example of a dependency could be, if the feature strong engine is selected than the feature small brakes cannot be selected.

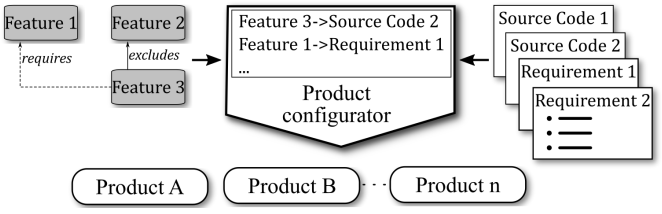


Fig. 1. Basic idea behind the Product Line Engineering development paradigm

Once the variability model is established, features can be mapped to one or more artifacts, i.e. to specific versions of engineering artifacts. For example, if a product configuration has a strong engine, then a particular engine control sofware must be used. The mappings between the features and artifacts are known as *presence conditions* [20] and they are exemplified in the middle part of Figure 1. The presence conditions are arbitrarily complex propositional formulas over the set of features in the variability model. Individual products can be derived in the so-called *application phase* of PLE by using the *product configurator*, that based on the selection of features, composes appropriate artifacts into individual products.

In Scania CV AB, there are several tens of different types of artifacts and they are maintained either manually, in hand-written documents, or in multiple in-house and third-party tools. The number of features is around ten thousand while the number of presence conditions is in the order of millions and they are maintained together with the artifacts in individual tools.

### B. Linked Data

Linked Data refers to the set of principles for structuring and publishing data on Internet. The principles can be summarized as: each real life entity, digital or physical, is identified by an *Uniform Resource Identifier* (URI) and is called a *resource*; the result of any operation over resources is always presented in *Resource Description Framework* [24] (RDF) format; resources should have links, also URIs, to other resources. The main technologies used for publishing LD are the aforementioned RDF data model, its data modeling extension *RDFS Schema* [24], and accompanying ontology language called *Web Ontology Language* [24] (OWL). Additional standardized LD technologies exist [24].

Publishing LD is a process in which the data from existing sources is assigned with URIs so that each data fragment, can be serialized as RDF according to an *LD schema* [23]. An LD schema is an information model of the published data that is usually expressed in RDFS or OWL language which define constructs that can be used for LD information modeling. Unlike in traditional data integration where a high level modeling language describes the overall data integration schema, in LD *"the data schema is represented with the data itself"* [23], i.e. RDFS and OWL are syntactically the same as the RDF data model. The RDFS language defines the concepts of a class, relation specialization, grouping of resources into containers, and some frequently used string-valued attributes. Interestingly, although not stated explicitly, RDFS is underpinned by the concept of an unlimited number of abstraction levels, similar to MLM approaches. The OWL language defines a richer vocabulary with concepts like class disjointness, relation cardinality, inverse relations and others, but it does not support MLM concepts and its RDF serialization is cumbersome. Furthermore, both RDFS and OWL assume the *open-world assumption* [6] (OWA), i.e. any information that is not stated is not false. This can lead to inconclusive analysis operations of the RDF data which is not desirable in an enterprise.

There are several benefits from using LD in an enterprise: use of robust, generic web-based principles for data exchange and querying, the possibility to reuse already published LD, and incremental data integration because adding new entities

to the information model does not falsify the previous one. The basic idea of LD-based data integration implemented in Scania, is shown in Figure 2.

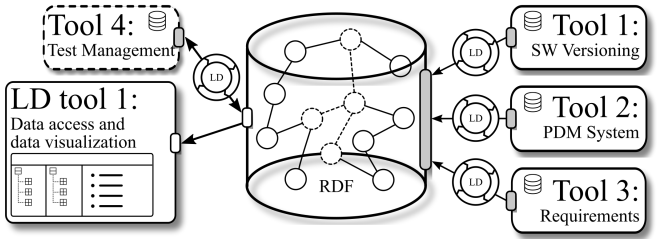


Fig. 2. Illustration of data integration in Scania CV AB based on LD principles

Figure 2 illustrates that artifacts from existing tools are published in RDF format and stored in a central database that can then be used for designing cross-tool artifact analyses. Grey interfaces represent *adapters* that implement LD principles and publish artifacts data from existing tools in RDF format. Currently, data integration in Scania is limited to three tools: *software versioning* tool, *product data management* tool and the *requirement specification* tool. Once stored in the central database, the data can be accessed through a purpose-build LD-based application, i.e. *LD tool 1*. The evolution of the presented data integration approach is to also enable consuming LD by existing tools, like exemplified on the case of *Tool 4*.

Designing adapters requires an information model that structures the relevant types of data objects from existing tools into a LD representation, potentially enriched by relations and attributes that do not exist in source tools. Since the information model represent the interface between data models in existing tools and RDF data model, the information model must treat two issues. Firstly, in all considered cases, the existing tools were not OWA-based while RDF is. Consequently, all the constraints that implicitly exist in non-OWA frameworks, e.g. object disjointness or completeness, must be explicitly captured and published in LD. Secondly, and more importantly, data objects which are instances in one tool can be classes in another tool, also noted in [18], and the fact that certain data objects exhibit this dual nature must be captured.

Since both RDFS and OWL languages do not provide support for expressing the class-instance nature of certain data objects, i.e. they are not based on MLM principles, and they assume OWA-based modeling, a different information modeling framework was needed in order to leverage the benefits of LD for enterprise data-integration.

### C. Multi Level Theory for Data Integration in PLE

This section briefly introduces the *Multi-Level conceptual Theory* (MLT) concepts togeter with interpretations of PLE concepts that were introduced in [17]. Detailed explanations of relevant concepts are presented in the next section on the example of the information model created using the MLT framework.

The MLT framework differentiates between three primary concepts. These are *types* that correspond to UML classes, *individuals* that correspond to UML objects, and *attributes* that correspond to UML associations. Types are semantically interpreted as sets and they are organized into an arbitrary number of abstraction levels where each level is represented by an *order type*. Each type declared in an MLT model is a specialization of an order type and an instance of the immediately higher order type or some type specializing the higher order type. In MLT, order types are called *IndividualOT*, representing types whose direct instances are individuals that cannot be instantiated further, *FirstOT* whose direct instances are types specializing the *IndividualOT*, *SecondOT* whose direct instances are specializing the *FirstOT* and so on. MLT is a *powertype-based* MLM framework, i.e. unlike in deep instantiation [15], the type-facet and the instance-facet of a type is explicitly modeled on two adjacent abstraction levels.

Attributes are used to represent properties of types and instances of types. Semantically, attributes represent binary relations; either between a type and a data-type, e.g. *string*, or between two types declared in the model. This distinction is reflected in the visualization of the MLT models where in the former case, attributes are visualized similar to attributes in UML class diagrams while in the later case the attributes are visualized as associations between types. The attributes relating two types declared in the model are referred to as *relations* and MLT separates them into *basic* and *structural* relations. Structural relations are predefined and they relate two types whether any arbitrary basic relation relates instances of types and must be declared by the modeler.

Work in [17], interprets and disambiguates basic relations between different artifacts that are *product-configuration specific* (PCS) and that inherently occur in the application phase of PLE. Each PCS relation is the consequence of previously mentioned presence conditions that specify sets of product configurations in which a particular artifact can be used. Furthermore, the approach discusses the structuring of versions and variants of different types of artifacts and their relations to product configurations and corresponding presence conditions. The main result of the work is the transformation of presence conditions, that are usually just syntactical annotation, into first-class citizens with well-defined semantics. This allows publishing presence conditions as non-string RDF data that can be analyzed using standard LD technologies.

## III. Information Modeling Using MLT

The model in Figure 3 captures the details about *requirements* and *Electronic Control Units* (ECUs), i.e. embedded computers that control vehicle behavior. The complete model is significantly larger and it includes more artifact types but the excerpt in Figure 3 captures all relevant model aspects.

Because the information model is used for LD-based data integration, the attributes of types and relations between types are reused from various LD vocabularies. The notation *prefix:name* represents a shorthand for *vocabularyURI/name*. For example,

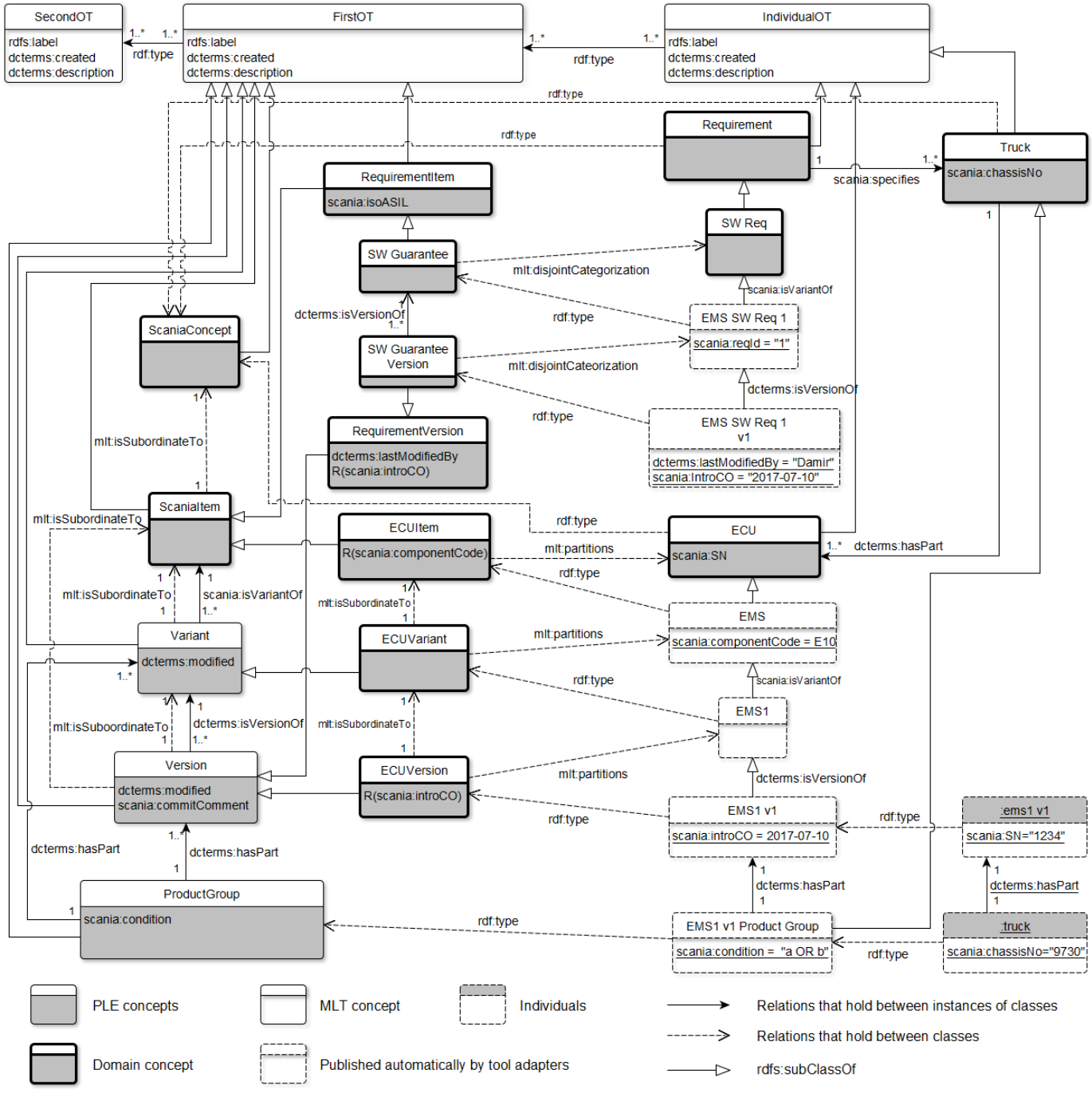


Fig. 3. MLT information model for LD-based data integration

dcterms:description attribute is a shorthand for the full URI http://purl.org/dc/terms/description where the attribute *description* is defined. The prefix scania indicates a vocabulary that contains Scania-specific definitions. An underscored attribute is an attribute of the entity labeled by it while a non-underscored attribute is an attribute of instances of the type that is labeled by it. Figure 3 omits the values of attributes common to all entities, i.e. rdfs:label, dcterms:description, and dcterms:created.

The rdf:type relation is the RDF equivalent of the *instance of* relation. It should be noted that all types specializing type FirstOT are instances of the type SecondOT but the rdf:type relations are omitted in order to reduce clutter.

### A. Different Aspects of the Information Model

The information model in Figure 3 captures several different types of information. The three types in the top part of Figure 3 represent the MLT order types which structure the model into abstraction levels. Because all entities in the model are either direct or indirect, through specialization relations, instances of these three types, they model the common attributes of all entities in the information model. Direct instances of the types on the IndividualOT abstraction level are the real world individuals, e.g. :ems1 v1 and :truck.

The information model in Figure 3 also contains some example RDF data that illustrates the relations between the resources published by LD adapters and types in the MLT model. All entities with a dashed border are examples of RDF data. Two main aspects captured by the information model are the PLE aspect that describes all artifacts and their properties from the PLE point of view and the use of MLT constructs for structuring and capturing the constraints over the published RDF data.

*1) The PLE Aspect:* Types and relations in the left part of the information model, i.e. ScaniaConcept, ScaniaItem, Variant, Version, and ProductGroup, classify all the artifacts from individual tools into these five types and also prescribe the mandatory structural and basic relations between them. Instances of type ScaniaConcept represent abstract, stable concepts in the domain whose definitions change very rarely. Instances of type ScaniaItem represent different logical realizations of concepts. Instances of type Variant represent refinements of instances of type ScaniaItem according to different criteria, e.g. artifact generation, new product variant etc. Instances of type ScaniaVersion represent particular engineering artifacts that can be used to construct or describe a particular product configuration. The only structural relation between these types is the isSubordinateTo relation which implies that instances of types related by it must be in the specialization relation, i.e. when serialized into RDF, specialization corresponds to the rdfs:subClassOf relation. Structuring instances of these four types into specialization hierarchies reflects the process of incremental refinement during the engineering of these artifacts. Moreover, relations dcterms:isVersionOf and scania:isVariantOf are prescribed between the mentioned types in order to enrich the RDF data with traceability links.

Instances of type ProductGroup represent product configurations defined by different presence conditions for the purpose of describing product configurations in which particular artifacts can be used. Unlike in Section II-A, where presence conditions were propositional formulas, in the information model presence conditions are represented by types *defined* by these propositional formulas and these types are instances of the type ProductGroup. For example, type EMS1 v1 Product Group has instances that are individual product configurations, each containing an individual part that is an instance of typeEMS1 v1. In other words, each instance of the type EMS1 v1 Product Group can be represented by a set of features that entail the truthfulness of the presence condition of the type EMS1 v1. The relation dcterms:hasPart between type ProductGroup and Version and Variant indicates that any type that is an instance of types Version or Variant must be related to a particular product group that defines the artifacts inclusion into particular product configurations. In the complete model, even the instances of the type ScaniaItem can be related to an instance of the type ProductGroup and the relation between them can be different, e.g. *testedBy* in the case of test cases.

*2) Use of MLT-specific constructs:* In Figure 3, types specializing the order type IndivudalOT capture the information that each instance of the type Requirement *specifies* one or more instances of the type Truck and that each instance of the type Truck *has* one or more parts which are instances of type ECU. Furthermore, a particular type of requirement is a *software requirement* and therefore the type SW Req specializes the type Requirement.

Type ECU is *partitioned* by the type ECUItem, following the so-called *type-object* pattern recognized in [15]. The *partitions* relation, based on the *powertype* relation, implies that all specializations of the type ECU are pairwise disjoint and are the only instances of the type ECUItem. For example, Scania currently has around 80 different ECUs that are instances of type ECUItem. Type ECUItem specializes the type ScaniaItem whose meaning was previously described. Unlike the type ECU, type SW Req is *disjointly categorized* by the type SW Guarantee. The disjointCategorization relation implies that all instances of the type SW Guarantee are pairwise disjoint specializations of the the type SW Req but not the only ones. For example, in contract-based requirements specification, there are additional requirement types such as *SW Assumption* whose instances also specialize the type SW Req.

Type ECUItem has an attribute called scania:componentCode that is an MLT *regularity attribute*, denoted by placing the attribute in parenthesis preceded by the letter *R*. The attribute scania:componentCode represents a sticker placed on physical ECUs and it is used to differentiate between instances of the type ECUItem in order to connect proper cabling on the assembly line. According to the MLT framework, a

regularity attribute is an attribute such that each attribute value is unique to the instance that assigns it the particular value. In Figure 3, the attribute `scania:componentCode` is assigned with a value `S8` by the type `EMS` which is the Engine Management System and also an ECU. Any other value of the `scania:componentCode` attribute must belong to a different specialization of type `ECU`.

As mentioned, the aim of the work in [17] was to interpret PLE concepts within the MLT framework in order to enable capturing complex and semantically well-defined information models in PLE. Given the previously described PLE concepts in the information model, all other types on the `FirstOT` abstraction level whose instances will be published as RDF data specialize the introduced PLE concepts and leverage the MLT structural relations in order to define the semantics and capture the constraints over RDF data. Each instance of the type `ScaniaConcept` is at the top of a specialization hierarchy that captures levels of refinement, in other words levels of variation, of artifacts in the application phase of PLE. The specialization hierarchies are enforced trough `isSubordinateTo` relations. For example, type `EMS` specializes the type `ECU`. Similarly, type `EMS1` specializes type `EMS` and type `EMS1 v1` specializes type `EMS1`. Furthermore, all types in the specialization hierarchies are disjoint, and in some cases complete, which is captured through the `partitions` and `disjointCategorization` relations. MLM capabilities of the MLT framework are essential for expressing different types of information on different abstraction levels.

Regarding basic relations, there are only a few between types in Figure 3. Besides the `dcterms:hasPart` basic relation which is reused from the *Dublin Core Meta Data* vocabulary, other basic relations like `scania:specifies` and `scania:isVariantOf` are defined for the need of this particular model.

### B. Using the Information Model

This section illustrated the usage of the MLT information model for data integration according to LD principles. To that end, Figure 4 omits most of the details from the model shown Figure 3 and only shows the model structure.

As previously discussed, the types in the information model represent different engineering artifacts across the product lifecycle. According to Section II-B and Figure 2, in order to transform the artifact data into RDF data, it is necessary to implement adapters that will transform the artifact data from the internal tool formats into RDF format.

In Figure 4, black boxes indicate that a particular tool maintains instances of the indicated classes. The entities with a dashed border, either types or individuals, are instances of the types owned by a tool and they are published as RDF data by tool adapters. The need for an MLM based approach is illustrated by types owned by *Tool 1* which are instances of types owned by *Tool 2*. Because MLT forces strict stratification of modeled entities into abstraction levels with *instantiation*, i.e. `rdf:type`, relation between them, the implementation of

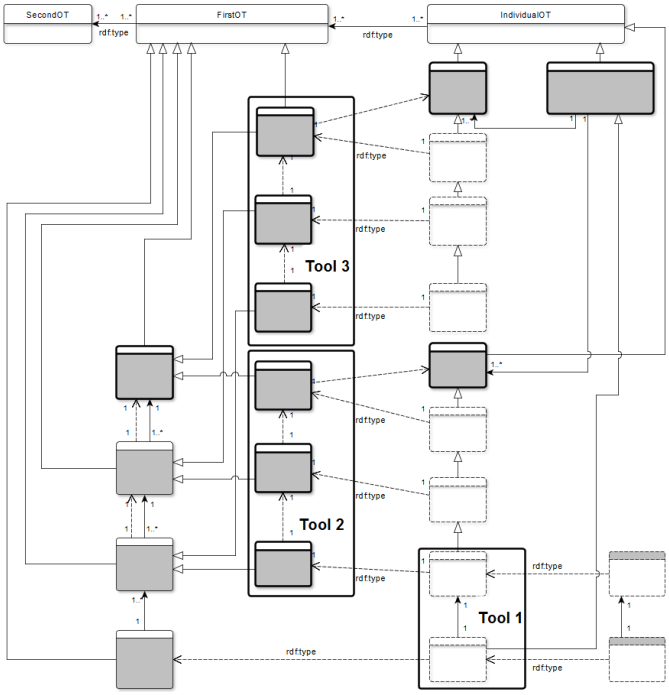


Fig. 4. LD adapter construction based on the MLT information model

adapters using traditional programming languages with two-level concepts was possible. Each adapter was responsible for a set of types and their instances, while the types not owned by any existing tool were published by an additional *"virtual"* adapter. The "virtual" adapter also created the links between RDF data created by tool adapters on the RDF level in order to create the MLM model that conforms to the MLT information model. The number of types published by the adapters is in the order of magnitude of tens of thousands while the number of individuals is far greater. It should be noted that these numbers were reached after several increments of the information model and that initial numbers were much smaller. However, incremental data integration is one of the strengths of LD and the incremental approach was beneficial for gradual adoption and structuring of the domain knowledge.

As mentioned earlier, the first version of LD-based data integration and tool adapters is in place for three tools. As of now, the primary usage of the integrated data is for different stakeholders to visualize, navigate, and explore relations between different types of artifacts using an in-house developed tool called *Search & Browse*. Part of the future work is to define and implement standardized analysis operations over the data that target different use cases like consistency checking or change impact analysis. Also, in order to automate adapter development and data validation, future versions of the information model shall be created using a model-driven approach based on the *Lyo Toolchain* tool [10].

## IV. Discussion

The case introduced in the present paper indicates that creating information models for data integration requires the

use of an MLM framework because instances from one tool can be types in another tool. The particular MLM framework that was used in the present paper, the MLT framework, offers many benefits but there are also some shortcomings.

Most importantly, MLT defines the *partitions*, *disjoint categorization*, and *subordination* relations which were essential for capturing the *item-variant-version* pattern. Furthermore, *regularity attributes* are a succinct way to express the constraint that different attribute values lead to the creation of new types. The fact that MLT is a *powertype-based* MLM framework means that for each `partitions` relation, there is an additional type defined [15] compared to *deep instantiation* approaches. However, because each type in the information model represents a data fragment, it is desirable to explicitly define all of them. Moreover, the separation into two types, provides a type where instance-facet attributes can be declared and a type where type-facet attributes can be declared which was helpful in explaining the MLM concepts to employees in Scania.

Regarding the practical aspects of using the MLT framework, the absence of tool support was the biggest challenge. Although a MLT-UML profile was suggested in [7], it was not implemented and MLT models had to be created and debugged manually. A part of the future work with the *Lyo toolchain* is about extending its graphical modeling tool based on results from [17].

As a purely powertype MLM approach, MLT does not support expressing some information. There are two examples where *deep instantiation* and *dual-deep instantiation* are needed. First example concerns the `componentCode` regularity attribute. As mentioned earlier, `componentCode` is a sticker on physical individuals which are instances of version artifacts, e.g. `EMS1 v1`, and therefore it should be considered as the attribute of individuals. However, because types like `EMS1 v1` are added by the adapters, i.e. they are not present in the information model, the `componentCode` attribute must be modeled as an attribute of instances of the `ECUItem` type on the `FirstOT` level. If MLT framework had supported deep instantiation, the *componentCode* attribute could be modeled as an attribute of type `ECUItem` with *potency=2* and then all specializations of the type `EMS` would inherit that attribute thus yielding the desired result.

The second example concerns the use case for dual-deep instantiation. For example, individual `ems1 v1` could have an attribute `scania:assembledBy` whose value is of type *ScaniaEmployee*. Simultaneously, any type on any abstraction level has an attribute `dcterms:created` whose value should also be of type *ScaniaEmployee*. In this scenario, two different attributes on two different abstraction levels are related to the same type which is basic idea of dual-deep instantiation. Currently, the information model does not treat this issue in any way because the type `ScaniaEmployee` is not modeled.

## V. Related Work

Reports about applications of *Multi-Level Modeling* approaches on real cases are still rare, particularly in areas other than model-based software development including software architectures and *domain-specific languages*.

In [11], standardized IT management frameworks for enterprise infrastructure modeling, evolution and decision making are surveyed and common chanllenges and prospects for improvement are identified. Following the survey, multi-level modeling approach was used to construct a language that tackles identified challenges. The report in [22] also looks at enterprise architecture modeling using a modeling language developed during industrial projects. The language *Txture* uses both multi-level modeling concepts and traditional two-level modeling concepts and the authors claim that a language with enough expressiveness for capturing complex domains must support concepts both paradigms.

Work in [1] tackles the problem of mapping domain specific concepts to concepts from automotive safety standards by introducing a mapping layer which leads to a multi-level model. In the absence of an adequate MLM framework for the presented problem, the paper introduces the *DeepML* language that combines constructs from several MLM frameworks. The approach in [14] is motivated by the problem of interoperability between information systems, a similar problem to the one discussed in the present paper. The authors propose additional disambiguation of instantiation and specialization relations in order to facilitate tool interoperability but they do not apply their approach to a realistic case study. However, the introduced extensions are formally captured and then evaluated against a set of criteria such as modularity, level stratification and etc.

## VI. Conclusions

Constant increase of product complexity in PLE development of SIS requires seamless access to well-structured artifacts data with the purpose of making decisions or ensuring different properties of developed SIS. One way to enable such operations is *data integration* of existing artifacts data into a unified representation. This paper has reported on the experiences from applying an MLM framework, specifically the MLT framework, for the creation of an information model for LD-based data integration used in a real industrial case from the heavy vehicle manufacturer, Scania CV. MLM capabilities of MLT enabled capturing several aspects of the considered data while the relations *partitioning*, *disjoint categorization* and *subordination* have enabled expressing constraints and structuring published LD with well-defined semantics. Being a powertype-based MLM approach, MLT forces clear separation of modeled entities into abstraction levels which has facilitated adapter implementation using traditional programming languages. Although MLT provides formal definitions of all of its constructs, the lack of tool support prevented using them in an automated fashion. As an integration technology, LD has proven useful primarily in two aspects. Firstly, the ability to reuse definitions of attributes like *creator*, *description*, or

*hasPart* relation were a significant time-saver. Secondly, the possibility to incrementally integrate data allowed gradual adoption and structuring of domain knowledge. Future work is targeted towards providing tool support for model-driven LD adapter generation based on information models created using MLT framework.

## REFERENCES


[1] Al-Hilank, S., Jung, M., Kips, D., Husemann, D., Philippsen, M.: Using multi level-modeling techniques for managing mapping information. In: MULTI@MoDELS (2014)

[2] Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software & Systems Modeling (2008)

[3] Batory, D.: Feature models, grammars, and propositional formulas. In: SPLC '05 (2005)

[4] Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A.: A survey of variability modeling in industrial practice. In: VaMoS '13 (2013)

[5] Bizer, C., Heat, T., Berners-Lee, T.: Linked Data: The Story so Far. IGI Global (2011)

[6] Brachman, R., Levesque, H.: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers Inc. (2004)

[7] Carvalho, V.A., Almeida, J.P.A., Guizzardi, G.: Using a well-founded multi-level theory to support the analysis and representation of the powertype pattern in conceptual modeling. In: CAISE '16 (2016)

[8] Carvalho, V.A., Almeida, J.P.A.: Toward a well-founded theory for multi-level conceptual modeling. Software & Systems Modeling (2016)

[9] Doan, A., Halevy, A., Ives, Z.: Principles of Data Integration. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2012)

[10] El-Khoury, J., Gurdur, D., Nyberg, M.: A model-driven engineering approach to software tool interoperability based on linked data (2016)

[11] Frank, U.: Designing models and systems to support IT management: A case for multilevel modeling. In: MULTI@MoDELS (2016)

[12] Halevy, A., Rajaraman, A., Ordille, J.: Data integration: The teenage years. VLDB '06 (2006)

[13] Halpin, T., Morgan, T.: Information Modeling and Relational Databases. Morgan Kaufmann Publishers Inc. (2008)

[14] Jordan, A., Mayer, W., Stumptner, M.: Multilevel modelling for interoperability. In: MULTI@MoDELS (2014)

[15] Lara, J.D., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM Transactions on Software Engineering Methodology (2014)

[16] Nešić, D., Nyberg, M.: Multi-view modeling and automated analysis of product line variability in systems engineering. In: SPLC '16 (2016)

[17] Nešić, D., Nyberg, M.: Modeling product-line legacy assets using multi-level theory. In: REVE@SPLC '17. To appear. (2017)

[18] Neumayr, B., Jeusfeld, M.A., Schrefl, M., Schütz, C.: Dual Deep Instantiation and Its ConceptBase Implementation (2014)

[19] Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering. Foundations, Principles, and Techniques. Springer-Verlag Berlin Heidelberg (2005)

[20] v. Rhein, A., Grebhahn, A., Apel, S., Siegmund, N., Beyer, D., Berger, T.: Presence-condition simplification in highly configurable systems. ICSE '37 (2015)

[21] Sudarsan, R., Fenves, S., Sriram, R., Wang, F.: A product information modeling framework for product lifecycle management. Computer-Aided Design (2005)

[22] Trojer, T., Farwick, M., Haeusler, M.: Modeling techniques for enterprise architecture documentation: experiences from practice. In: MULTI@MoDELS (2014)

[23] W3C Consortium: Best practices for publishing linked data (2017), `https://www.w3.org/TR/ld-bp`

[24] W3C Consortium: Semantic web (2017), `https://www.w3.org/standards/semanticweb`

[25] OASIS consortium: Open services for lifecycle colaboration (2017), `http://open-services.net`