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

Damir Nešić, Mattias Nyberg

{damirn, matny}@kth.se
ITM/MMK/MDA
Royal Institute of Technology
Brinellvägen 83, 100-44 Stockholm, Sweden

**Abstract.** Developing safety critical, *Software-Intensive Systems* according to the *Product Line Engineering* (PLE) paradigm is a process in which many different engineering artifacts are produced and bound with configuration management data. Different artifact types are maintained by different tools, sometimes even manually, which makes automated analysis involving several types of artifacts a challenging task. Overcoming this issue can be achieved through *data integration* of existing data and a promising technology for robust and scalable data integration is the *Linked Data* (LD) technology. However, because the primary use case for LD is data integration on Internet, its *information modeling* capabilities in an enterprise setting are limited. This paper reports on the experiences from applying the *Multi-Level conceptual Theory*, to the problem of information modeling for data integration in the context of PLE using the principles of LD. Being a *powertype based* framework, MLT allows separation of the class and instance facet of modeled entities thus facilitating practical implementation. Formal definitions of modeling construct are essential for creation of unambiguous models and some of the constructs that MLT defines are particularly useful in a data integration scenario but there are 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.

**Keywords:** Multi Level Modeling, Information Modeling, Product Line Engineering, Linked Data

## 1 Introduction

Large scale development of safety-critical *software-intensive systems* (SIS) is a process involving various engineering disciplines that produce many engineering artifacts across the SIS lifecycle, e.g. software, hardware, various models and documents. The artifact data is usually analyzed in order to ensure their consistency, e.g. in terms of consistent versioning or tracing, and more importantly in order to establish product safety or reliability with respect to standardized

norms. All these analyses are vital for development and deployment of aforementioned SIS. However, it is often the case that different tools maintain different artifacts in different formats. Moreover, some artifacts might be maintained manually and coupled with their great diversity, performing automated analyses described above can be a daunting task [21].

Development of *highly configurable SIS*, also known as *Product Line Engineering* (PLE) [19], brings additional difficulties. The goal of PLE is systematic reuse of engineered artifacts among different product configurations. In other words, performing the analyses described above are always with respect to particular product configurations, i.e. the artifact data is contextualized with respect to product configurations. It is common that besides different types of meta-data, each artifact is labeled with configuration data that has first-class citizen status but is also scatered across different tools and documents [15, 19].

In order to avoid costly migration to a new toolchain and processes with the goal of enabling previously described automated analyses, *data integration* [8] techniques can be used to extract and integrate existing artifacts data from existing tools into a unified representation that allows performing the analyses. Besides the traditional approaches to data integration [11, 8] underpinned by technologies like *relational databases* [12] and *ER* or *UML* information models [12], the idea of *Linked Data* (LD) [4] has in recent years been applied to the problem of robust and scalable data integration on Internet, but also in different engineering domains, primarily through *OSLC* standards [18].

Some important benefits of LD are: a standardized data model, well-defined semantics of the data, support for incremental integration, and scalable and robust data manipulation through web protocols. In an enterprise, all these benefits should support creating structured and well-formed data that conforms to an *information model* that in turn captures all domain constraints allowing above mentioned analyses. However, LD principles are primarily aimed at data integration on Internet, a highly distributed system that is not concerned with data structure or its truthfullness. Consequently, information modeling for LD is not tailored to support the enterprise data-integration use case. As an alternative to information modeling frameworks supporting LD, like *RDFS* and *OWL* languages, but also to traditional *MOF-compliant* modeling frameworks, we have investigated [16] the applicability of *Multi-Level Modeling* (MLM) paradigm [14, 2] for information modeling in the PLE context. More specifically, the *Multi-Level conceptual Theory* (MLT) [7] was evaluated and complemented with PLE-specific concepts in order to support modeling of different artifacts and their configuration data.

In the present paper we report on the experiences from applying the framework presented in [16] on the case of LD information-model creation for data integration in the real industrial PLE context on the case of 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 2 presents relevant PLE and LD concepts followed by the MLT framework with PLE extensions as the framework for LD information modeling. Section 3 presents the details of the LD information-model and its use in the data integration process. Section 4 discuses the benefits and shortcomings of the applied modeling framework and is followed by Section 5 that surveys related work. Finally, Section 6 concludes the paper.

## 2 Background

This section introduces the PLE and LD concepts followed by a brief introduction to the MLT framework and PLE extensions from [16].

### 2.1 Product Line Engineering

The main idea of PLE is to engineer artifacts that realize or describe a product in a way that these artifacts can be systematically reused in 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 configurations could be described as having features: engine, brakes, cab, trailer and optionally another trailer. The features and their mutual dependencies are captured in a *variability model* [19], in the case of features known as the *feature model* [3], that captures all possible product configurations in terms of features. Left part of Figure 1 shows a fragment from a feature model. An example of a dependency could be, if a truck configuration has small brakes then it cannot have a strong engine.

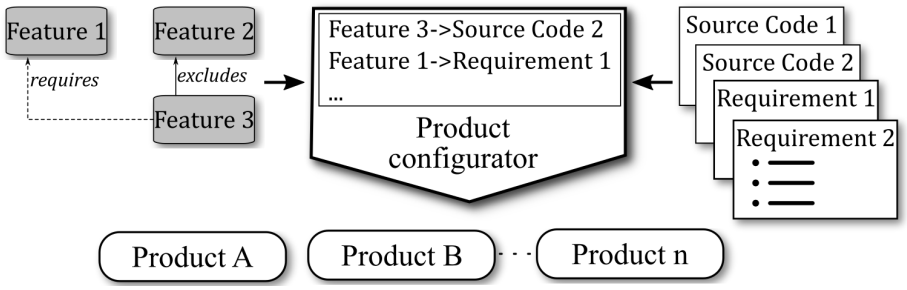


**Fig. 1.** Basic idea behind the Product Line Engineering development paradigm

Once the variability model is established, features can be mapped to one or more artifacts. 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 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 by using the *product configurator*, that based on the selection of features, composes corresponding artifacts into individual products. Figure 1 exemplifies the overall idea of PLE.

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 seven thousand while the number of presence conditions is in the order of millions and they are maintained together with the artifacts in different tools. All of this implies that in order to be able to perform automated analysis over several different types of artifacts with respect to different product configurations, it is necessary to integrate the artifact data from different sources.

### 2.2 Linked Data

Linked Data is a set of principles for publishing and structuring data on Internet. The principles can be summarized as: each entity is identified by a *Uniform Resource Identifier* (URI) and is called a *resource*; the result of any operation over the resources is always presented in a standardized format, i.e. *Resource Description Framework* [24] (RDF); whenever possible, resources should have links, also URIs, to other resources. The main technologies used for creation of LD are the aforementioned RDF data model and its data modeling extension *RDFS Schema* [24], a query language called *SPARQL* [24], and an inference language called *Web Ontology Language* [24] (OWL). There are also other standardized technologies which are not relevant for the present paper.

Publishing LD is a process in which the data from existing sources is assigned with URIs so that each piece of data, regardless if it is atomic or complex, can be serialized into RDF according to an *LD schema* [23]. LD schema is an information model of the published data that is usually expressed in the *RDF Schema* (RDFS) language or *Web Ontology Language* (OWL) language, which define *LD vocabularies* for 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 data expressed in RDF. The RDFS language defines a vocabulary with concepts of class, relation specialization, grouping of resources into containers, and definition of mostly string-valued attributes. Interestingly, although not stated explicitly, RDFS is underpinned with 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. Furthermore, both RDFS and OWL assume the *open-world assumption* [5] (OWA), i.e. any information that is not stated is just unknown but not false.

An example of a reusable vocabulary is the *Friend Of A Friend* (FOAF) vocabulary which defines concepts about people and their relationships. The idea is that whenever some LD is published, the terms from existing vocabularies should be reused. For example, whenever a piece of LD expresses that a person has a certain role, e.g. professor, employee etc,. the definition of *Person* from the FOAF vocabulary should be used.

The appeal of LD for enterprises lies in the robust and generic web-based principles for data exchange and querying, the possibility to reuse existing vocabularies, and incremental development because adding new entities to the information model does not falsify the previous one. The basic idea of data integration implemented in Scania CV AB, following the LD principles, is shown in Figure 2.

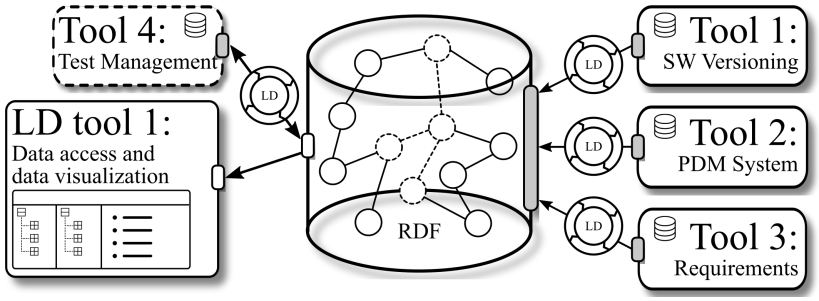


**Fig. 2.** Illustration of data integration in Scania CV AB based on LD principles

Figure 2 illustrates that various artifacts from existing tools are published as LD and stored in a central database that can then be used for different types of analyses across the product lifecycle. Grey filled interfaces represent *adapters* that create URIs for tool artifacts and their attributes. Currently, data integration 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 purpose build application, here *LD tool 1* that can directly interpret LD. The evolution of the data integration is to enable both publishing and consuming LD by existing tools, like exemplified on the case of *Tool 4*.

The OWA-based modeling frameworks that support publishing LD work well in the case of Internet because the complete data is never known and arbitrary changes can occur. On the contrary, enterprises want to analyze their data in order to establish certain properties about the product, make business decisions, check if established processes are followed, all of which requires an information model that precisely captures all possible types or concepts, relations, and constraints in the enterprise. To facilitate this, the information model should follow the *closed-world assumption* where any unstated information is false. Furthermore, as noted in [17], in data integration instances from one tool can be classes

in another tool and once this data is integrated into a single database, the information model should be able to capture this fact, i.e. it must be expressed using MLM concepts. Since both RDFS and OWL languages are OWA based, and they do not provide any support for MLM, a different information modeling framework is needed in order to leverage the benefits of LD for enterprise data integration.

### 2.3 Multi Level Theory for Data Integration in PLE

This section briefly introduces the *Multi-Level conceptual Theory* (MLT) concepts togeter with PLE extensions that were introduced in [16]. Detailed explanations of all the concepts are presented in the next section in conjunction with the information model created using the MLT framework.

The MLT framework differentiates between three primary concepts. These are *types*, *individuals*, and *attributes*. Types and individuals are commonly referred to as *entities*. 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 of the higher order type specializations. Order types are called *IndividualOT*, representing types whose instances are individuals that cannot be instantiated further, *FirstOT*, whose instances are specializations of the *IndividualOT*, *SecondOT* whose instances are specializations of the *FirstOT* and so on. MLT is a *powertype-based* MLM framework, i.e. unlike the deep instantiation frameworks [14], the type-facet and the instance-facet of a type are modeled on two different but adjacent abstraction levels.

Attributes are used to represent properties of types and instances of types. Semantically, attributes represent relations between two sets; either a between a type and a data-type or between two types. In the latter case, this corresponds to relations between types. Syntactically, this distinction is reflected in the visualization of the MLT model. Attributes are visualized similar to attributes in traditional class modeling while the attributes corresponding to relations are visualized as directed associations between classes. An important aspect of the MLT framework is that all constructs are defined in *first-order-logic* which leads to unambiguous models.

MLT framework differentiates between *basic* and *structural* relations. Structural relations are relations that hold between types while the basic relations are relations that hold between instances of types. Work in [16], discusses and disambiguates basic relations between different artifacts that are *product-configuration specific* (PCS) and that inherently occur in the PLE context. Each PCS relation is the consequence of previously mentioned artifact-presence conditions that specify the subset of all product configurations in which the artifact can be used. In other words, each presence condition defines the so-called *product group type*, whose instances are specific product configurations. In this way, presence conditions that are commonly just syntactical artifact annotations, become first order citizens in the MLT information model.

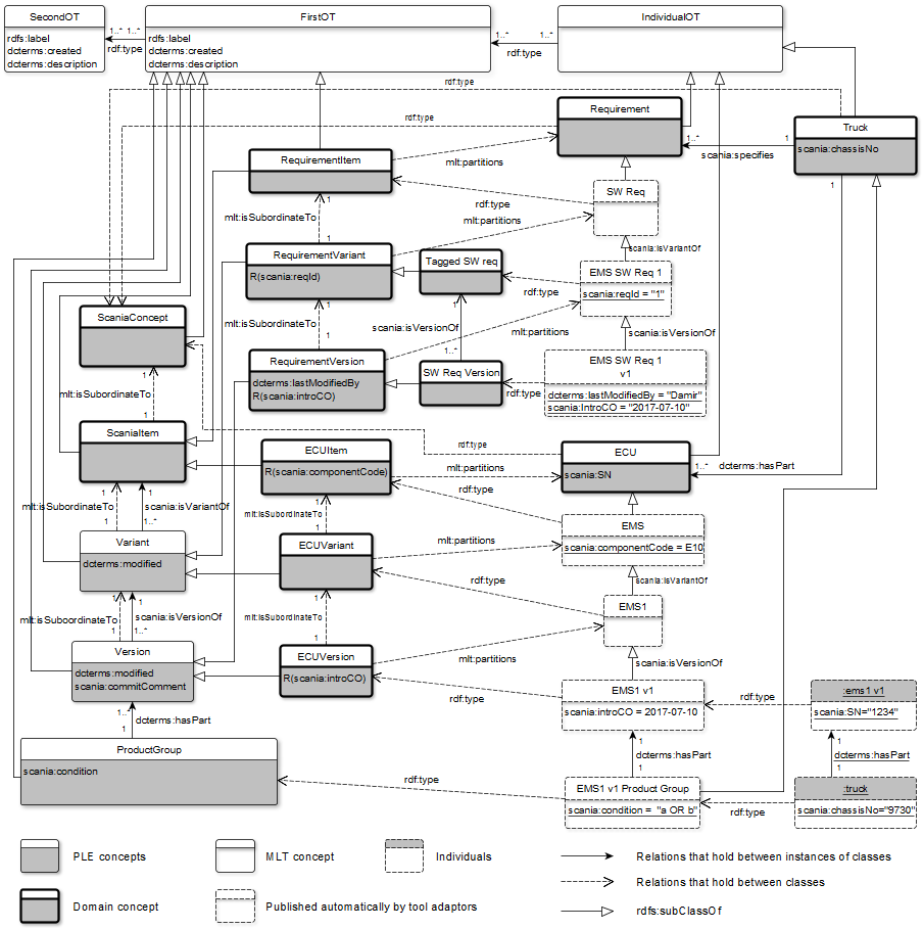


**Fig. 3.** MLT information model for data integration according to LD

## 3 Information Modeling Using Extended MLT

The model in Figure 3 captures the details about requirement artifacts from the requirements tool and the *Product Data Management* (PDM) tool regarding *Electronic Control Units* (ECUs), i.e. embedded computers that are a part of each vehicle. The complete model is several times 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 data integration in the form of LD, 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, `dcterms:description` is a shorthand for `http://purl.org/dc/terms/description` where the attribute *description* is defined. The prefix `scania` is used in the case when these terms were defined for

the purpose of creating the scania-specific information model. An underscored attribute is an attribute of the entity while a non-underscored attribute is an attribute of the instances of the type which is labeled by it. Figure 3 exemplifies underscored attributes that have special significance while it omits common attributes like `rdfs:label`, `dcterms:description`, and `dcterms:created`.

The `rdf:type` relation is the RDFS vocabulary term equivalent to the *instance of* relation. It should be noted that all types specializing type `FirstOT` are instances of the type `SecondOT` but the *instance of* relations are omitted in order to reduce clutter. Types with dashed boarder are examples of types that are added automatically by the adapters while the remaining types are part of the information model.

### 3.1 Created Information Model

In Figure 3, types specializing the `IndivudalOT` type capture the information that each instance of the type `Truck` is *specified* by one or more instances of type `Requirement` and that each instance of the type `Truck` *has* one or more parts which are instances of type `ECU`. Types `Requirement`, `ECU`, and `Truck` are instances of type `ScaniaConcept` which represents stable concepts in the domain whose definitions change very rarely.

Types `Requirement` and `ECU` are *partitioned* by types `RequirementItem` and `ECUItem`, following the so-called the *type-object* pattern recognized in [14]. The *partitions* relation, based on the *powertype* relation, implies that all specialization of the partitioned type are pairwise disjoint instances of the partitioning type. For example, Scania currently has around 80 different ECUs that are instances of type `ECUItem`. Type `ECUItem` is a specialization of the type `ScaniaItem` which represents domain concepts that change on a yearly basis. The relation between types `ScaniaItem` and `ScaniaConcept` is the `isSubordinateTo` relation and it implies that each instance of type `ScaniaItem` must be a specialization of an instance of `ScaniaConcept`.

Type `ECUItem` has an attribute called `scania:componentCode` that is a *regularity attribute*, denoted by placing the attribute in parenthesis preceded by the letter *R*. According to the MLT framework, a regularity attribute is an attribute such that each attribute value is unique to the instance that assigns it a 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 belongs to a different specialization of type `ECU`. The attribute `scania:componentCode` is a sticker on physical individuals used to differentiate between instances of the type `ECUItem`, e.g. EMS and other, in order to connect proper cabling on the assembly line.

There are only several basic relations 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`, `scania:isVariantOf`, and `scania:isVersionOf` are defined for the need of this particular model. The complete model defines more basic relations. For example, there are additional

types specializing types `RequirementVariant` and `RequirementVersion` along with basic relations defining traceability and decomposition relations.

As mentioned, the original MLT was framework was extended in [16] in order to support modeling of PLE concepts. In Figure 3, type `EMS` specializes the type `ECU`. Similarly, type `EMS1` specializes type `EMS` and type `EMS1 v1` specializes type `EMS1`. The structuring of types in a specialization hierarchy is a reoccurring pattern in the model and is a consequence of the PLE approach which induces creation of variants of different artifacts that are a part of different product configurations. Type `EMS1` represents one out of eight *variants* of EMS artifact while the type `EMS1 v1` is the most specific kind of an `EMS` ECU and it represents one out of ten particular *versions* that are actually instantiated into parts that are built into individual trucks. This is exemplified by representing a particular individual `:ems1 v1` which is an instance of type `EMS1 v1` with a particular value for the attribute *serial number*, i.e. `scania:SN="1234"`.

Being a reoccurring pattern, the previously mentioned specialization hierarchy pattern can be captured on the `FirstOT` level. As previously described, relations `partitions` and `isSubordinateTo` force the structuring of instances of types related by these relations into specialization hierarchies. For each set of types that have the same supertype, e.g. `EMS1,...EMSn` and `EMS`, there is a `isSubordinateTo` relation between the types whose instances are placed in the specialization hierarchy, e.g. `ECUVersion` and `ECUVariant`. Furthermore, in order to capture relations between variants and versions of any type of artifacts, types `Variant` and `Version` are defined together with a `scania:isVersionOf` relation between them and `scania:isVariantOf` relation between types `Variant` and `ScaniaItem`. In order to make the notation more succinct, the `rdfs:subClassOf` relation that is the consequence of the subordination relation, and the `isVariantOf` and `isVersionOf` relations are represented by the same graphical symbol, the specialization arrow.

Finally, type `Product group` and its instances capture the product configurations in which an artifact can be used. If we recall the principles of PLE described in Section 2.1, artifacts are labeled with presence conditions written over the set of features and during product derivation the selected features entail truthfulness of some of the presence conditions which in turn leads to the composition of a product configuration composed of artifacts that are labeled with presence condition evaluated as true.

As outlined in Section 2.3 and according to [16] a presence condition of type `EMS1 v1` defines a type that is a *specialization* of the type `Truck`, i.e. the type that corresponds to all product configurations, and represents product configurations that can be described with features entailing the truthfulness of the `EMS1 v1` presence condition. Each such type is a specific *product group*, e.g. `EMS1 v1 Product Group`, and is an instance of the type `ProductGroup` defined on the `FirstOT` level. The relation `dcterms:hasPart` between types `EMS1 v1 Product Group` and `EMS1 v1` is a *Product-Configuration Specific* (PCS) relation. In Figure 3, the PCS relations is modeled between type `ProductGroup` and type `Version` but the full model also contains relations between type `ProductGroup`

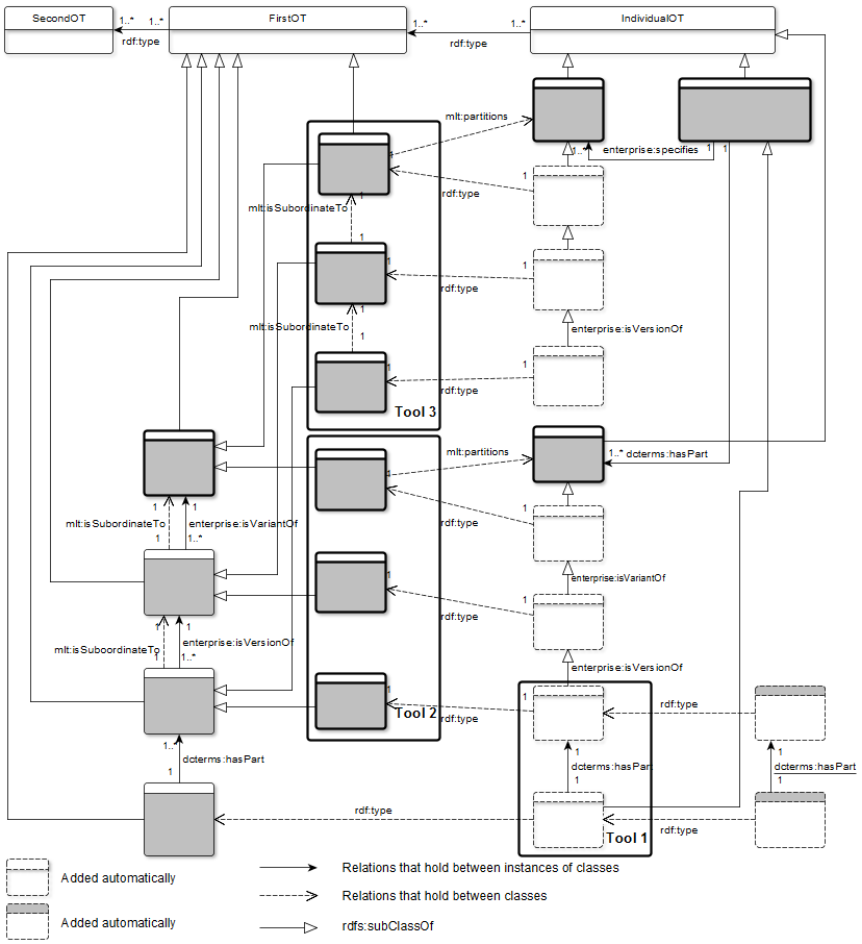


**Fig. 4.** Integrating data from different tools according to the information model

and types `Variant` and `ScaniaItem`. In other words, instances of types `Variant` and `ScaniaItem` can also be specific for certain product configurations.

### 3.2 Using the Information Model

Figure 4 omits most of the details from the information model shown in Figure 3 in order to illustrate the generalized usage of the information model for data integration according to LD.

As previously discussed, the types in the information model represent various engineering artifacts across the product lifecycle. According to Section 2.2 and Figure 2, in order to transform the artifact data into LD, it is necessary to implement adapters for the tools maintaining the artifacts that will transform

the artifact data from the internal format into LD 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 different tools and they are published as LD by the different tool adapters. The need for an MLM based approaches is illustrated by types owned by *Tool 1* which are instances of types owned by *Tool 2*. Because MLT is a powertype-based approach that forces strict stratification of modeled entities into abstraction levels with *instantiation* relation between them, the implementation of adapters using traditional programming languages with two-level concepts was not difficult. Each adapter was responsible for a set of types and their instances, while the types not owned by any existing tool are published by an additional *"virtual"* adapter. The number of types published by the adapters is in the order of magnitude of tens of thousands. The number of individuals published by the adapters corresponds to the number of products and their constituting parts which yield millions of individuals. It should be noted that these numbers of LD resources were reached after several increments of the information model and that initial numbers were much smaller. However, incremental integration and evolution of the information model is one of the strengths of LD and the incremental approach was beneficial for gradual adoption and structuring of domain knowledge.

Currently, the first version of data integration and tool adapters from three tools is in place. 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, change impact analysis etc. 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* [9].

## 4 Discussion

The case introduced in the present paper indicates that capturing a complex information model for data integration in PLE context 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 *partitions* and *subordination* relation 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 [14] compared to the *deep instantiation* approaches. However, because the end goal is data integration and the majority of types are created automatically by tool adapters, this does not create significant

overhead for the modeler. Moreover, the separation into two types, the partitioning and the partitioned type, provides a type where instance-facet attributes can be declared and a type where type-facet attributes can be declared.

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 [6], it was not implemented and MLT models had to be created and debugged manually. As a part of the future work with the *Lyo toolchain*, we are extending its graphical modeling tool to support the MLT framework with extensions from [16].

Being a *powertype-based* 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 types, like `EMS1 v1`, therefore it can 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 one of the ECU related types 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. In the absence of deep instantiation capability, we have resorted to the solution presented in Figure 3 which yields a different result but it is sufficient for the current use case of the integrated data.

The second example concerns a use case for dual-deep instantiation. For example, individual `ems1 v1` could have an attribute `scania:assembledBy` whose value is *scania employee*. Simultaneously, any type on any abstraction level has an attribute `dcterms:created` whose value should also be *scania employee*. 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 type `ScaniaEmployee` is not defined in the information model but in the future it must be incorporated even though MLT does not support dual-deep instantiation.

## 5 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 [10], standardized IT management frameworks for enterprise infrastructure modeling, evolution and decision making are surveyed and common obstacles and prospects for improvement are identified. Following the survey, a multi-level modeling languages *XMF* and $FMML^x$ are evaluated against the previously defined obstacles. 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 from multi-level and two-level languages.

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 [13] treats the problem of interoperability between information systems, a similar problem to the one discussed in the present paper. The authors propose a new language that extends the concepts of *specialization* and *instantiation* in order to increase language expressiveness. The extensions are formally captured and then evaluated against a set of criteria such as modularity, level stratification and etc.

## 6 Conclusions

Constant increase of product complexity in PLE development of SIS forces enterprises to perform ever more complex analyses of different artifacts across the product lifecycle in order to check consistency or prove safety or reliability. One way to enable automated analysis of artifact data is *data integration* of existing artifacts into a unified representation. This paper has reported on the experiences from applying an MLM framework, particularly the MLT framework, for creation of an information model for data integration according to LD principles in the PLE context on the real industrial case of the heavy vehicle manufacturer, Scania CV AB. MLT constructs like *regularity attributes*, *partitioning*, and *subordination* have been particularly useful for the creation of the information model but some concepts in the domain still require the absent deep and dual-deep instantiation concepts. Being a powertype-based MLM approach, MLT has forced clear separation of modeled entities into abstraction levels which has facilitated adapter implementation using traditional programming languages. A significant aspect of the MLT framework is the fact that all the modeling constructs are defined in first-order-logic which facilitates creation of unambiguous models. However, the lack of tool support prevents using the formal definitions 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* 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 the PLE extension of MLT framework that supports Linked Data principles.

## 7 Acknowledgments

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

## 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. Bizer, C., Heat, T., Berners-Lee, T.: Linked Data: The Story so Far. IGI Global (2011)
5. Brachman, R., Levesque, H.: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers Inc. (2004)
6. 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)
7. Carvalho, V.A., Almeida, J.P.A.: Toward a well-founded theory for multi-level conceptual modeling. Software & Systems Modeling (2016)
8. Doan, A., Halevy, A., Ives, Z.: Principles of Data Integration. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2012)
9. El-Khoury, J., Gurdur, D., Nyberg, M.: A model-driven engineering approach to software tool interoperability based on linked data (2016)
10. Frank, U.: Designing models and systems to support IT management: A case for multilevel modeling. In: MULTI@MoDELS (2016)
11. Halevy, A., Rajaraman, A., Ordille, J.: Data integration: The teenage years. VLDB '06 (2006)
12. Halpin, T., Morgan, T.: Information Modeling and Relational Databases. Morgan Kaufmann Publishers Inc. (2008)
13. Jordan, A., Mayer, W., Stumptner, M.: Multilevel modelling for interoperability. In: MULTI@MoDELS (2014)
14. Lara, J.D., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM Transactions on Software Engineering Methodology (2014)
15. Nešić, D., Nyberg, M.: Multi-view modeling and automated analysis of product line variability in systems engineering. In: SPLC '16 (2016)
16. Nešić, D., Nyberg, M.: Modeling product-line legacy assets using multi-level theory. In: REVE@SPLC '17. To appear. (2017)
17. Neumayr, B., Jeusfeld, M.A., Schrefl, M., Schütz, C.: Dual Deep Instantiation and Its ConceptBase Implementation (2014)
18. OASIS consortium: Open services for lifecycle colaboration (2017), http://open-services.net
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