Ontologies, Metamodels, and the Model-Driven Paradigm

Uwe Aßmann[1], Steffen Zschaler[1], and Gerd Wagner[2]

[1] Technische Universität Dresden
Institut für Software- und Multimediatechnik (SMT)
http://www-st.inf.tu-dresden.de
{uwe.assmann, steffen.zschaler}@inf.tu-dresden.de
[2] Brandenburgisch-Technische Universität Cottbus
Institute of Informatics
http://www.informatik.tu-cottbus.de
G.Wagner@tu-cottbus.de**

– In memory of Emma Larsdotter-Nilsson who, in search for a thesis in biological modelling [24], died unexpectedly in October 2005 –

Abstract. This paper discusses the role of ontologies, models, and metamodels in the model-driven paradigm. To show how ontologies can be used in model-driven architecture (MDA) and its generalization model-driven engineering (MDE), the paper argues that the main difference of models and ontologies lies in their descriptiveness resp. prescriptiveness. While an ontology is a descriptive model, a model in MDE is specification—that is, a prescriptive model. Therefore, the role of ontologies in model-driven engineering is to describe the existing world, the environment, and the domain of the system (*analysis*), while the role of system models is to specify and control the system under study itself on various levels of abstraction (*design* and *implementation*).
Based on this distinction, we present a scheme combining descriptive ontologies and prescriptive models in the *meta-pyramid*, the multi-level modelling approach of MDE. In this scheme, MDE starts from ontologies, refines, and augments them towards system models, respecting their relationships to prescriptive models on all metalevels. Conceptually, the scheme is a first attempt towards a *megamodel* of ontology-aware MDE.

1 Introduction

Software development centers around the production of several models, going from abstract to concrete (Fig. 1). Step by step, constructs in abstract models are *refined* to more concrete model elements. Roughly speaking, development can be divided into two phases. The *analysis phase* constructs a requirement specification describing all features the user would like to have, building on

** Work partially supported by European Community under the IST programme, contract IST-2003-506779-REWERSE [3].

top of a domain model, a business model, and a context model. Later on, the *design phase* produces an architectural design specification and a detailed design specification. In a last step, this is filled out to an implementation of the software system.

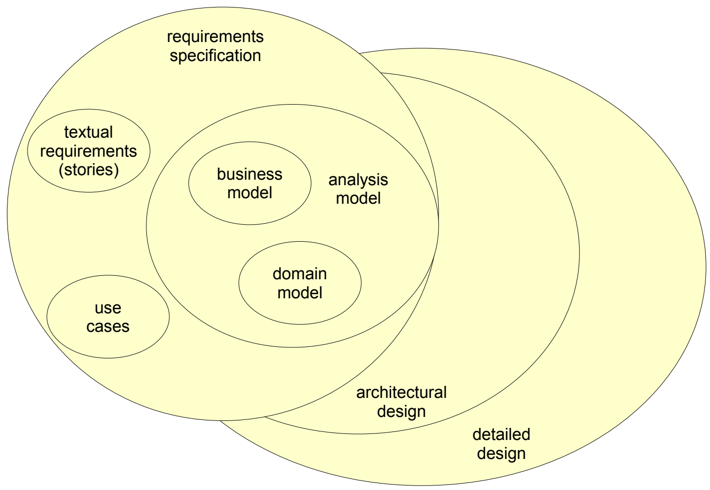

Fig. 1. Models in a typical object-oriented software development process.

Model-driven engineering (MDE) is a variant of this refinement-based software development in which models are no longer loosely coupled, but connected in a systematic way [8, 9]. On the one hand, MDE improves on the software refinement method of the 70es [38] in the sense that more concrete phases are discerned. On the other hand, every phase derives a more concrete model not only by manual refinement, but also by semi-automatic or automatic transformation. To this end, models must be connected, i.e., model elements can be traced from a more abstract model to a more concrete model and vice versa. This is achieved through metamodelling: *metamodels* define sets of valid models, facilitating their transformation, serialization, and exchange, which is a prerequisite for tool support.

In recent years, model-driven engineering has been popularized by a specific incarnation, *model-driven architecture (MDA)*. In this process, one specific type of model information, the *platform information*, plays an important role. In MDA, models differ in how much platform information they contain (Fig. 2). For instance, one platform can be the programming language of the system, another can be the employed libraries or frameworks, a third one can be the binary component model. The designer begins with a high-level model that abstracts from all kinds of platform issues, and iteratively transforms the model to more concrete models, introducing more and more platform specific information. Hence, all information that relates to programming language, frameworks, or component model are added to the platform-independent model by platform-specific extensions.

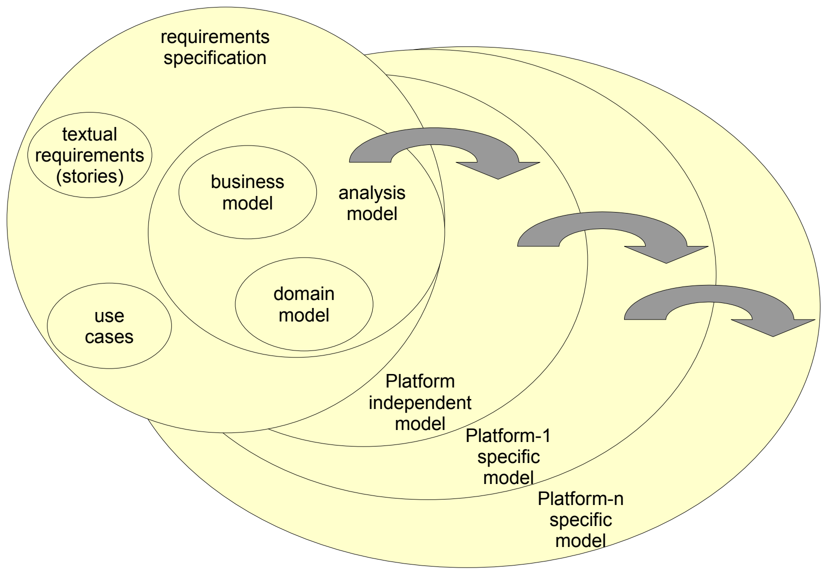

Fig. 2. Models in model-driven architecture.

Essentially, in MDA three types of viewpoints on models are distinguished [27]. The computationally independent viewpoint CI sees the system from the customer's point of view, and manifests in a computation-independent model (CIM). This model is a typical analysis model, since it is expressed in terms of the problem domain.

> "The computation independent viewpoint focuses on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as yet undetermined." [27]

The CIM contains a *domain model*, describing the concepts of a domain and their interrelations, a *business model*, describing a company's rules of business, and, finally, the requirements. The platform-independent viewpoint PI sees the system from the designer's point of view, abstracts from all platforms a system may run on, and results in a *platform-independent model (PIM).* Roughly speaking, a PIM contains an architectural model, adorned with sufficient detail of platform-generic implementation issues. Finally, the platform-specific viewpoint adds platform-specific extensions and results in a *platform-specific model (PSM)*. Either this model can be executed directly, or it is used to generate code.

To arrive at a PSM, the PIM must be extended with platform-specific information, for which it is merged with several platform-specific extensions (Fig. 3). Because the platform-specific extension can be regarded as an aspect that crosscuts the platform-independent information [22], one can speak of *model weaving*. This *MDA pattern*, weaving platform-specific models from PIMs and PSE, can be repeated over several levels. Often, different kinds of platforms are involved and one would like to vary the system over all combinations of these platform instantiations; for example, by having a system with C# and Java, both on the web and GUI-client platforms. The idea of multi-level MDA is to repeat the model weaving process over several levels (Fig. 2), so that on every level, a PSM is re-interpreted as a new PIM for the next platform.

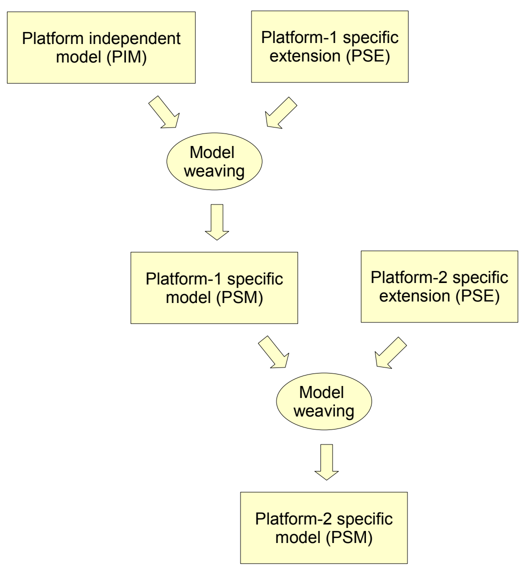

Fig. 3. The MDA pattern: weaving a platform-specific extension as an aspect into a PIM as a base.

A heretic spectator could remark that MDA (and hence MDE) is not a new technology, but just refinement-based software development. However, since MDA discerns platform-specific information as the main criterion for refinement, the entire process is much more structured than the "free-style" refinement of the 1970s. Also, in MDA, all models are graph-based, while standard refinement worked mainly for syntax trees.

Recently, the Semantic Web has popularized another notion of model: *ontologies*. Ontologies are *formal explicit specifications of a shared conceptualization* [16]. They describe the concepts of a domain, similar to the domain model of a CIM. While they are currently used mainly in the Semantic Web, they could be useful also in general software development [1, 7]. But then, the question arises how ontologies should be integrated into MDE, and more specifically, into the process architecture of MDA. And this is what the rest of the paper is about. In Sect. 2, ontologies are compared to general models, resulting in the insight that ontologies describe reality while models specify artifacts. Sect. 3 investigates these relationships in more detail and explains, how the specification relationship `instance-of` can be used to build up a stack of models, the so-called IRDS *meta-pyramid*. Sect. 4 extends the meta-pyramid with ontologies, distinguishing a descriptive dimension. A comparison to related work concludes the paper.

2 Models and Ontologies

In this section, we discuss the fundamental terms 'model' and 'ontology' and investigate their primary commonalities and differences. We begin by looking at definitions of 'model' and 'ontology', go on to discuss a fundamental property of models—namely whether they are descriptive or prescriptive—and finish by showing how this distinction can be applied to distinguish between ontologies and other software models.

2.1 What's in a Model?

Models are representations, descriptions, and specifications of things. Pidd defines:

> "A model is a representation of reality intended for some definite purpose." [30]

Hence, models represent reality (in the following coined by the `is-represented--by` relation).

Models have a *causal connection* to the modeled part of reality: they must form *true* or *faithful* representations so that queries of the model give reliable statements about reality, or manipulations of the model result in reliable adaptations of reality. Pidd characterizes this as follows:

> "A model is an external and explicit representation of a part of reality as seen by the people who wish to use that model to understand, change, manage, and control that part of reality." [30]

Secondly, while models represent reality faithfully, they may *abstract* from irrelevant details. For instance, while models are finite descriptions, they may well describe an infinite language—that is, an infinite set of things or systems. Usually then, abstractions are involved—for example, about the number of elements in the language.

A model can represent many different kinds of realities, e.g., domains, languages or, in particular, systems. Hence, we can distinguish *domain models* from *system models*, models that describe or control a set of systems:

> "A model of a system is a description or specification of that system and its environment for some certain purpose." [27]

where the environment of a system is described by a domain model.

Models can describe structure or behavior. In the former case, models describe the concepts of a reality and their interrelation, the *static semantics* of a domain, its *context-free structure* or *context-sensitive structure*. Wellformedness rules (*integrity constraints*) describe valid configurations of reality.

Example 1. UML class diagrams are frequently used together with an *Object Constraint Language* [25]. The OCL integrity constraints describe valid configurations and interrelationships of classes and objects in an UML class model.

Secondly, while a structural model contains abstractions of a domain or a system and their interrelationships, a behavioral model also specifies their behavior, their *dynamic semantics*. In this case, a model may state assertions on the behavior of things in a domain or of some systems.

Example 2. Modelica is a multi-domain modeling language for simulation, visualization, and controlling technical systems. Hence, it is a prescriptive modeling language for the dynamic semantics of technical systems [12].

2.2 What's in an Ontology?

Recently, the Semantic Web has popularized another notion of model: *ontologies*:

> Ontologies are "formal explicit specifications of a shared conceptualization". [16]

They describe the concepts of a domain, similar to the domain model of a CIM. Since concepts are abstractions and play an important role in models, an ontology is certainly a special kind of model. But what is the exact difference? To answer this question, we have to introduce two other qualities of models.

An important property of ontologies is the so-called *open-world assumption* [18]. It states intuitively, that anything not explicitly expressed by an ontology is unknown. Hence, ontologies use a form of partial description or underspecification as an important means of abstraction. In contrast, most system models underlies the assumption that what has not been specified is either implicitly disallowed or implicitly allowed (*closed-world assumption*), to restrict arbitrary extensions of the system, which could introduce inconsistencies.

It is important to distinguish whether models describe or control reality. If they describe, they monitor reality and form *true*, or *faithful*, abstractions. If they control, they prescribe reality; that is, they specify well-formedness conditions what reality should be like, once it has been constructed. It can also be said that such models are templates or schemas of reality.

Hence, a most fundamental feature of a model is that it can be *descriptive* or *prescriptive* [34]. In the former case, the model describes reality, but reality is not constructed from it. In the latter case, the model prescribes the structure or behavior of reality and reality is constructed according to the model; that is, the model is a *specification* of reality. Favre [10] observes that in a descriptive model truth lies in reality, whereas in a prescriptive model, truth lies in the model itself. Descriptive models are, of course, used in analysis and reengineering, specifications in design and forward engineering. Since most specifications model systems, a prescriptive system model is also called a *system specification*.

Models are abstractions from reality for some purpose [30]. Ontologies are special models. In general, models can be prescriptive or descriptive. Most of the models used in software development and design are of a prescriptive nature in that they form the templates from which the system is later implemented. In contrast, because of their open-world assumption, ontologies are always descriptive models. This is so, because the open-world assumption does not allow for a complete and final description: Anything that has not been said explicitly is unknown. Two very different systems may satisfy an ontology, if they differ in areas not explicitly mentioned in the ontology.

Taking this discussion into account, we can define:

> An ontology is a *standardized, descriptive, structural model, representing reality by a set of concepts, their interrelations, and constraints under open-world assumption.*

This definition deserves some elucidating remarks. When comparing hallmark papers, such as [16] and [34], models and ontologies look very similar. Both provide vocabulary for a language and define validity rules for the elements of the language. Both models and ontologies, use integrity constraints to limit the valid instances of the domain. However, there are also differences. Ontologies are *shared* knowledge; that is, they must be standardized in a certain group of people. Ontologies are not specification models, but descriptive models in Seidewitz's sense. Ontologies do not describe systems, only domains. Hence, in a software engineering process, they play the role of an analysis model, not of a design or implementation model. Ontologies must describe a domain as completely as possible (to be as faithful as possible). With this view we contradict Devedzic: "Generally, an ontology is a metamodel describing how to build models." [7] and Gruber, because he maintains that ontologies are specifications [16].

To summarize: Specification models focus on the *specification*, *control* and *generation* of *systems*, ontologies on *description, standardization* and *concepts (structural models)* of *things*. Both kinds of models have in common the qualities of *abstraction* and *causal connection*. Because ontologies are, thus, somewhat special in the realm of models, we should investigate how we can make use of ontologies together with more common kinds of models.

3 Similarity Relations and Meta-Modelling

The previous arguments make it possible to discern two basic notions of the `is-represented-by` relation between a model and the corresponding part of reality (Fig. 4). In a descriptive model—for example, an ontology—the model describes the world; that is, the world's objects are in relation `is-described-by` with concepts of the descriptive model. In a specification model, the system's objects are created from the model; that is, an object is an `instance-of` a model element. Both relationships are representation relations, one is descriptive, the other is prescriptive. Their generalization `is-represented-by` is a `similarity` relation, in which a causal connection—delivering true and faithful statements—is defined between the represented things and the representing model. Beyond that, more similarity relations can be defined; for example, two things may share features (often expressed as `is-a`—that is, structural or behavioral inheritance), or they may be included in a hierarchy of sets (set inclusion, `subset-of`). In Fig. 4, `is-a` is defined as a sub-relationship of `subset-of`, because inheritance usually has a set-based semantics, namely, that all objects in a subclass are also members of the superclass. Additionally, `is-a` is a sub-relationship of `is-described-by`, because a superclass also describes all objects in a subclass. In contrast, `is-a` cannot be a sub-relationship of `instance-of`, because a superclass cannot necessarily be regarded as a template, schema or specification for a subclass.

3.1 Metamodels

In MDE, the specification relationship `instance-of` plays a special role. When the specification principle is applied repeatedly, models are regarded as the reality or system under study, so that models specifying models can be defined: *metamodels*. Metamodels *represent* and *specify* models; that is, they tell about what are valid ingredients of a model. More precisely:

> "A metamodel makes statements about what can be expressed in the valid models of a certain modeling language." [34]

Hence, a metamodel is a prescriptive model of a modeling language [34]. In general, metamodels are language specifications, not only of modelling, but also of arbitrary languages. In the current stage of MDE, they are mainly concerned with the static semantics—that is, with context-sensitive syntax of models, integrity and well-formedness constraints. However, modeling languages for dynamic semantics could also be applied to construct metamodels.

A language concept or construct in a metamodel is captured by a *metaclass.* While its structure and embedding describes the static semantics of the language constructs, its methods describe the dynamic behavior of the language construct. Usually, metaclasses are assembled in a behavioral metamodel, the *meta-object protocol (MOP)* [23], a reflexive metamodel that describes an interpreter for the language.

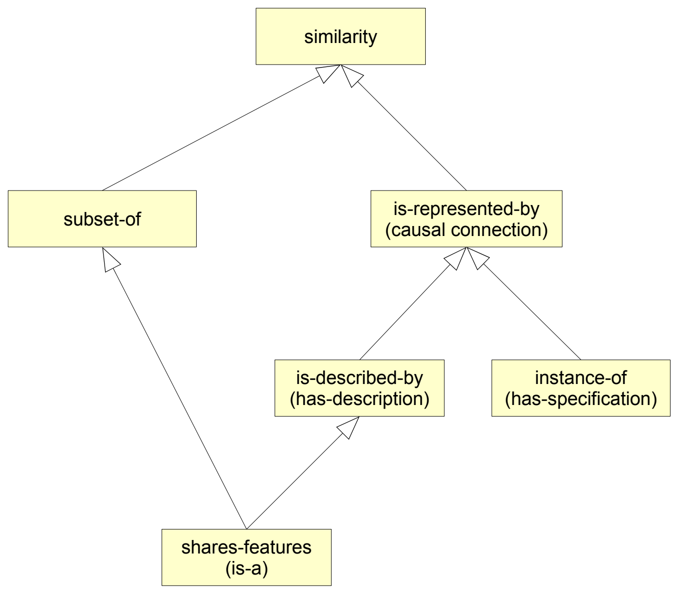

Fig. 4. A classification of `similarity` relations.

A big incentive for metamodeling has been the need of CASE (Computer-Aided Software Engineering) tool vendors to exchange models [26]. Since a metamodel describes, rather specifies, valid instances of a modeling language—models—it enables control over the structure and validiy of models. If two CASE tools agree on the same metamodel, they impose the same structure on their models, so that they can easily exchange them.

A language, described by a metamodel, can have a specific purpose or domain in which it is applied. Such purposes or modeling domains are called the *subject areas* of metamodels [11].

Example 3. For instance, the common warehouse metamodel (CWM) [28] defines a data specification language, a metamodel for data and information system applications. Workflow systems are another special subject area whose data, functions, and tasks can also be described with metamodels [32]. Software processes, being specific workflows, can be metamodeled [13] and used to construct software environments [4].

Subject areas can be organized in hierarchies or partial orders. Then, metamodels in a certain subject area can build on others from lower-level subject areas, so that complex languages can reuse simpler languages [11].

Example 4. The CASE Date Interchange Format (CDIF) has structured its metamodel into several subject areas (Fig. 5). The `Foundation` module contains information about names and relations; the `Common` module defines name aliasing for objects; and the `Data` module describes access paths to data and roles of objects. Based on these, data flow can be defined (`Data Flow` module). Another module specifies facilities for the presentation of objects. Finally, the full integrated metamodel uses all other modules and provides their concepts in an integrated way to the users.

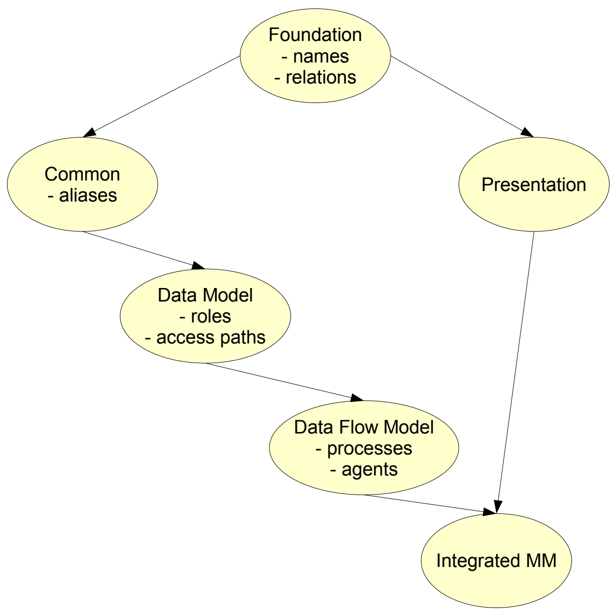

Fig. 5. The subject areas of CDIF and their metamodels in a use relationship.

3.2 Metametamodels

The specification principle can be applied repeatedly. Metametamodels *represent* and *specify* metamodels; that is, they tell about what are valid ingredients of a metamodel. They specify languages, and are thus a form of *language specification languages (metalanguages).*

In order to model anything useful, such a minimal metalanguage should contain the following concepts [11]:

1. classes (concepts)
2. attributes (or properties) of classes, contained in the classes
3. binary relations between classes

Thus, the Entity-Relationship Diagram language (ERD) [5] can be used as a very simple metalanguage. It defines modeling concepts, their attributes and their relationships. Other metalanguages exist that describe other forms of languages, or describe specific aspects:

1. Grammar specification languages—for example, EBNF—specify the concrete or abstract syntax of a text-based language [15].
2. Attribute grammars describe context-sensitive syntax in form of attribution rules of syntax trees [6].
3. Natural semantics can be employed for type systems, but are also able to specify dynamic semantics of systems [21].
4. In SGML [14], markup languages can be defined. XML [37] is a variant of SGML, allowing for defining context-free markup languages.
5. EXPRESS [33], a modelling language in the spirit of UML, is frequently used in mechanical engineering.

3.3 The Meta-Pyramid, the Modeling Architecture of Model-Driven Engineering

Based on the meta-principle, a so-called *meta-pyramid* can be defined, which displays systematically the mentioned stack of models and metamodels [20]. In essence, a meta-pyramid is a specification hierarchy linked by the `instance-of` relation, in which upper-level metamodels in some way specify other sets of lower-level models. Since sets of models can be regarded as languages, the meta-pyramid is a hierarchy of language specifications.

In this paper, we focus on the standard meta-pyramid of OMG, originally put up in ISO Information Resource Dictionary System (IRDS) standard [20] (Fig. 6), which contains 4 levels: M0-level (objects), M1-level (models), M2-level (metamodel or language level), M3-level (metametamodel or language description level). There are alternatives and a debate is going on whether the IRDS meta-pyramid is precise enough, because it is onedimensional, while multidimensional model pyramids exist [2]. However, at the moment, this is the mainstream meta-pyramid of MDE.

Example 5. On level M3, CDIF applies ERD as metametamodel [11]. There are ERD specifications for all subject areas of CDIF. On M3 of the OMG meta-pyramid, a *meta-object facility (MOF)* plays the role of a metametamodel. Essentially, its concepts are similar to those of the ERD.

The stereotypical models of MDA, CIM, PIM, and PSM live on level M1. All of them are specified by metamodels (CIM-MM, PIM-MM, PSM-MM), dialects of UML, enriching the UML core by *profiles* containing markup for model elements (stereotypes and tagged values). While all of these models are prescriptive, i.e., use the `instance-of` relationship, the question remains how ontologies, being models relying on `described-by`, can be integrated into the meta-pyramid. This is the topic of the next section.

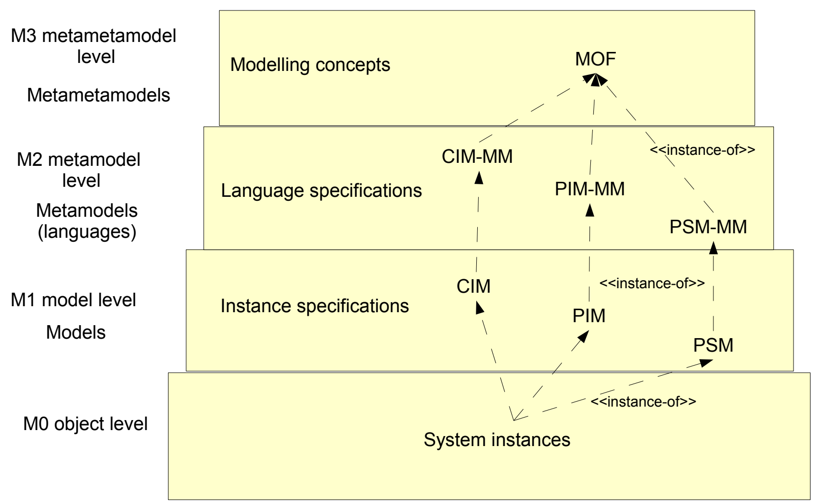

Fig. 6. The meta-pyramid with the MDA-related model types CIM, PIM, PSM.

4 MDE and Ontologies

This section discusses the role of descriptive and structural models, in particular ontologies, in the model-driven process. We propose a first attempt to an ontology-aware meta-pyramid and embed the stereotypical MDA model types (CIM, PIM, PSM) into it. In fact, this delivers a first ontology-aware megamodel of MDE [9]. First, the different role of domain and upper-level ontologies is discussed. Secondly, we postulate that ontologies can also be used as language descriptions. Thirdly, we propose an embedding of parts of the CIM as ontologies into the MDA meta-pyramid (*ontology-aware meta-pyramid*) and discuss its advantages.

4.1 Domain and Upper-Level Ontologies

The basic idea of the ontology-aware meta-pyramid is that most models in MDE are specifications, but can integrate ontologies on different metalevels as descriptive analysis models. Since ontologies differ from specifications due to their descriptive nature, the standard M0-M3 meta-pyramid can be refined from using pure specification models to also using ontologies.

Depending on the metalevel, an ontology may serve different purposes. In fact, there are different qualities of ontologies in the literature. First of all, the word *ontology* stems from philosophy, where it characterizes *Existence*.

> "Ontology is a systematic account of Existence." [16]

We coin such a systematic acount of existence a *World ontology*, a conceptualization of the world, that is, all existing concepts. Usually, a World ontology is split into an *upper-level ontology (concept ontology, frame ontology)*, providing basic concepts for classification and description, and several lower-level ontologies, *domain ontologies* describing domains of the world [17, 36]. Sowa characterizes domain ontologies as follows:

> "The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. The types in the ontology represent the predicates, word senses, or concept and relation types of the language L when used to discuss topics in the domain D." [35]

In contrast, upper-level ontologies can be defined as follows:

> "An upper ontology is limited to concepts that are meta, generic, abstract and philosophical, and therefore are general enough to address (at a high level) a broad range of domain areas. Concepts specific to given domains will not be included; however, this standard will provide a structure and a set of general concepts upon which domain ontologies (e.g. medical, financial, engineering, etc.) could be constructed." [19]

Usually, concepts of the domain ontology *inherit* from concepts in the upper-level ontology. For better interoperability and understanding, some researchers try to create a normalized upper-level ontology, from which all possible domain ontologies may inherit [29]. If a standardized upper-level ontology with modelling concepts existed, all domain ontologies could rely on a standardized concept vocabulary.

With this terminological distinction we can relate the different forms of ontologies to metalevels in the meta-pyramid. Domain ontologies live on level M1, they correspond to models. An upper-level ontology, also a standardized one, should live on level M2, because it provides a language for ontologies. Fig. 7 summarizes this insight, showing both dimensions, descriptive and prescriptive models, on different metalayers.

Interestingly, on the ontology side, inheritance is used as the connecting relation of M1 and M2, and not `instance-of`. We believe that this historic choice, which might have been made unconsciously, has a deep semantic reason in the difference between descriptiveness and prescriptiveness. A concept in a domain ontology on M1 needs to express its `similarity` to a modelling concept of an upper-level ontology (on M2). For this, the `is-a` relationship is sufficiently precise (cf. Fig. 4), and therefore, it has been selected in the ontology world to connect the metalevels. A concept in a specification model, however, has to express *that it has been made from* a specification model, which is clearly a more specific relationship than `is-a`. And this is the reason why in the IRDS world the `instance-of` relationship has been employed.

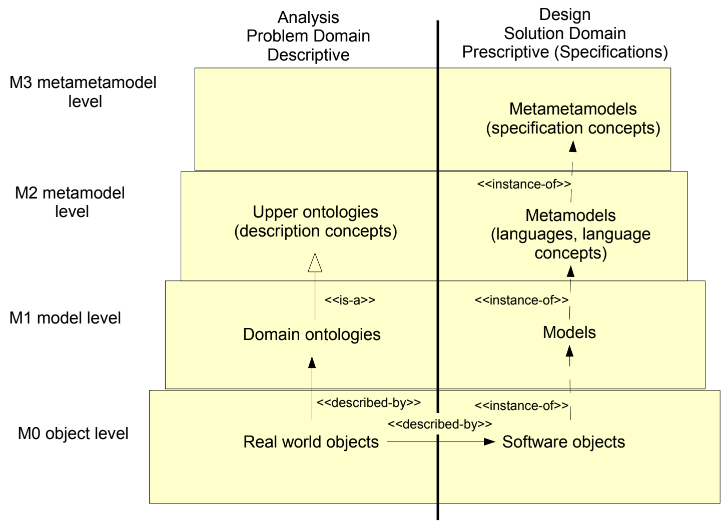

Fig. 7. The meta-pyramid with upper-level ontologies as modelling concepts.

4.2 Specification of Ontology Language

We argue that on level M3 of the descriptive side of the ontology-aware model pyramid, also a specification metalanguage should be employed. The language that describes or specifies an ontology language cannot be descriptive, because ontology languages are not something given, but artificial languages. Hence, a model to represent them should be prescriptive. We argue that the same metalanguage can be used on the ontology as well as on the system model side.

With this additional terminological distinction we can extend the ontology-aware meta-pyramid as follows. Domain ontologies live on level M1, upper-level ontologies live on level M2, while ontology metalanguages live on level M3. Since inheritance is used as a connector of the M1 and M2, it is possible to move language concepts between these levels. Fig. 8 shows the refinement, showing the different interrelationships of the models on different metalayers.

In fact, inheritance is not required in Fig. 8. While, usually, concepts in a domain ontology *inherit* from a concept in an upper-level frame ontology, we suggest that to distinguish them better from concepts in specification models, ontology modelling should causally connect ontological concepts by the `described-by` relationship. This would introduce a parallelism to using `instance-of` on the specification side and retain the basic ontological modelling principle of descriptiveness. Because of the parallel structure to the specification dimension, the advantage of such a meta-pyramid is that easily connections from ontologies to specifications can be made. In particular, this holds for the application of the meta-pyramid in the MDE.

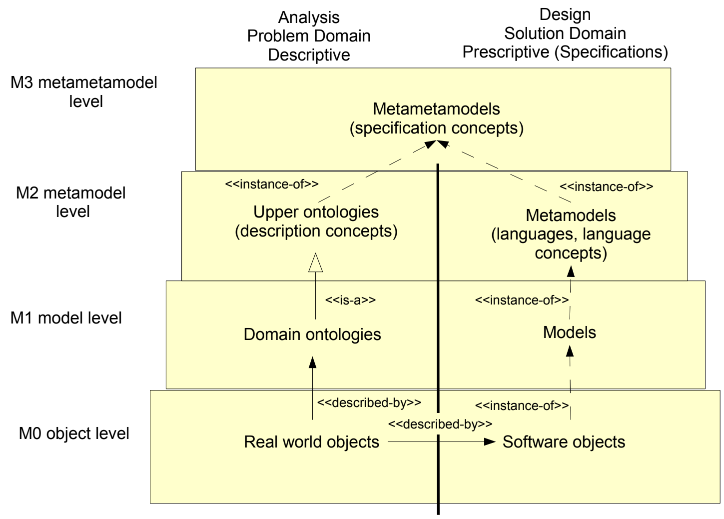

Fig. 8. The ontology-aware meta-pyramid.

4.3 Employing Domain Ontologies in the MDA

This version of a ontology-aware meta-pyramid permits us to group the MDA-based models around ontologies. In particular, the CIM plays a special role.

A CIM contains information about the system from the perspecitive of the system user. It is an *analysis model*. As such, it may contain a domain model, a business model, and requirements (Fig. 1) [27]. The gap between descriptive and prescriptive models concerns the CIM in particular. The domain model of a CIM can be selected to be a domain ontology (CIM-DO in Fig. 9). A business model, capturing business rules for a company that should prevail in all software products, can also be regarded as a domain ontology, namely that of the rules of the company (i.e., a domain ontology for a company, CIM-BO in Fig. 9). However, the parts of the CIM that deal with requirements, cannot be grasped by ontologies, because they *specify* requirements of a system to-be-made. Hence, this specification is grouped in CIM-RM in Fig. 9 as a specification model. This difference is also the reason, why only for CIM-RM, the specification part of the CIM, a metamodel is needed. Concepts of the CIM-DO or CIM-BO *describe* existing things, and may inherit from concepts on the language or concept ontological level. Concepts in CIM-RM, on the other hand, are instances of a CIM metamodel, because they specify parts of functions of a system.

Usually, a CIM is extended towards a PIM by hand, by enriching it with operational model elements. Hence, at least CIM-DO and CIM-BO play the role of standardized analysis models, whose elements can be traced back from the PIM [1]:

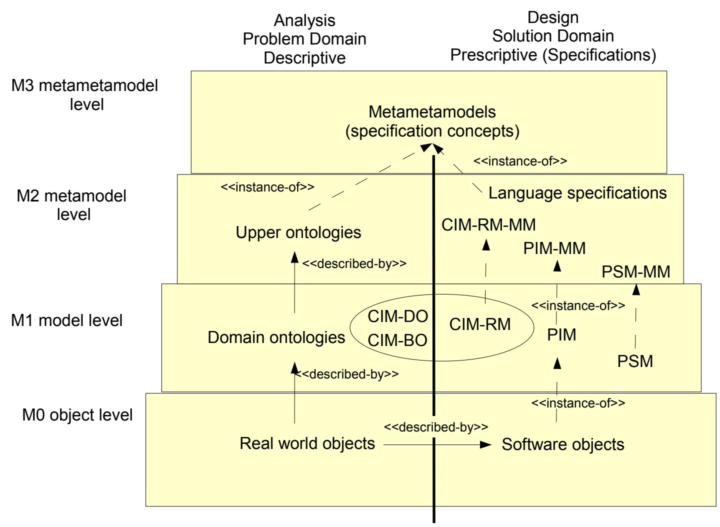

Fig. 9. A proposal for the role of ontologies in meta-pyramid of model-driven engineering and the MDA.

> "In an MDA specification of a system, CIM requirements should be traceable to the PIM and PSM constructs that implement them, and vice versa." [27]

Hence, surprisingly, model-driven architecture can benefit from ontologies, because via the standardized domain and business ontologies, once parts of a CIM, connection to PIM specifications can be made in a clear and systematic way.

The ontology-aware meta-pyramid offers several other benefits. First of all, it suggests a more concrete model-driven software development process. The designer starts from standardized analysis models, ontologies, which may have been defined long before project start. These domain and business models are refined towards design models. First, the requirements are added to yield a complete CIM. This is refined to a PIM and, then, conventionally, via several PSM towards an implementation. Employing ontologies as analysis models should increase the reliability of software products, because these models are well engineered, often used, and hence trustworthy, and avoid the risks of a self-made domain analysis.

Secondly, ontologies as analysis models offer more common vocabulary for software architect, customer, and domain expert. This should improve the understanding of the parties that order and construct software. Then, the standardization of the ontologies improves the interoperability of applications, because applications that use the ontology contain a common core of common vocabulary. Finally, domain and business ontologies can be reused in many software products. In particular, they may form the core of a software product line [1], around which many products are grouped, and from which they reuse domain terminology. Overall, this improves reuse in the software process.

It is also beneficial to make an explicit distinction between descriptive and prescriptive models in the MDA. Modelling becomes easier, because designers and domain experts can always answer the question: where lies the truth? In the model or in the reality? Specification models have to confine themselves to the modeling of *artificial things,* things that are made, while ontologies can focus on the description of *real things*, things that exist. (In particular, this can be seen from the example of the CIM, which in fact contains descriptive and prescriptive models.)

Finally, the ontology-aware meta-pyramid distinguishs conceptual from behavioral models. It seems to be convenient to center software modeling around concepts of a domain, or structure of a domain, while adding behavior to it step by step. In essence, this supports one of the central ideas of MDA, the refinement.

4.4 The Megamodel of Ontology-Aware MDE

The abovepresented ontology-aware meta-pyramid can be called a *megamodel* of ontology-aware MDE.

> "A megamodel is a model that describes a meta-pyramid." [10]

A megamodel stands outside of the meta-pyramid and describes all its levels. It has a global influence on all levels of the meta-pyramid. As such, the presented megamodel sheds new light on the relation of ontologies and metamodels in MDE. Systematically, ontologies can be related to specification models and metamodels in the meta-pyramid. It is important to distinguish the representation relations `is-described-by` and `instance-of`, because then, ontologies can be differentiated from specification models on all levels. As a whole, we propose that

1. An ontology-aware MDA should employ domain and business ontologies as parts of the CIM.
2. An ontology-aware MDE should additionally incorporate a second dimension of ontologies as descriptive models in the meta-pyramid, and maintain interrelations between the descriptive and prescriptive models on all levels.

5 Related Work

One of the works integrating metamodels and ontologies is [31], which extends software process and measurement ontologies with to metamodels from which software can be built. The work demonstrates the usefulness of ontologies in a metamodelling scenario.

Favre dissects the `instanceOf` relation into `representationOf` and `member-of` [8]. A model *represents* a language, and a system is an element of that language. This leads to a *relative* model hierarchy which is not restricted to 4 levels, but in which certain composite patterns denote more complex similarity relations, such as `instance-of` or `described-by`.

The standard aforementioned meta-pyramid is not undebated in the literature. Other pyramids can be described, in particular, if some design principles for meta-pyramids found in the literature are varied. A central role plays the similarity relations: since different notions can be defined, different model hierarchies result.

If every element on level n is instance of exactly *one* element on level n+1, a meta-pyramid is called *strict* [2]. With strict similarity, meta-pyramids must be lists or trees and are essentially one-dimensional. Based on this distinction, [2] defines a non-strict meta-pyramid, consisting of two dimensions arranged in a matrix. One dimension of the matrix is characterized by physical (technical, *linguistic*) instantiation. The linguistic similarity describes the *specification language aspect* of modeling: which language construct is instance of which language concept. Linguistic similarity is distinguished from logical *(ontological)*, which spans up the other dimension, the matrix-like meta-pyramid. Ontological similarity describes similarity of real-world concepts, e.g., that a dog is a mammal, and Fido is a dog. Clearly, this dimension corresponds to our descriptive, ontological dimension. However, [2] does not discern prescriptive vs. descriptive models, nor further different forms of similarity relations. It is future work to combine both approaches; at this point in time, it seems unclear whether a twodimensional matrix-like approach or the presented approach of parallel descriptive and prescriptive dimensions will prevail.

6 Conclusion

Ontologies are no silver bullet. They can be employed in the software process as descriptive standardized domain models, domain-specific languages, and modeling (description) languages. However, they should not be mingled with specifications of software systems. In model-driven engineering, both forms of models are needed and complement each other. It is time to develop appropriate megamodels that clarify the role of ontologies in model-driven engineering. This paper has presented one approach, however, this can be only an intermediate step, because we restricted ourselves to the standard IRDS meta-pyramid. Other, more sophisticated meta-pyramids exist and must be extended to be ontology-aware.

References

1. Uwe Aßmann. Reuse in semantic applications. In Norbert Eisinger and Jan Małuszyński, editors, *Reasoning Web, First International Summer School 2005*, number 3564 in Lecture Notes in Computer Science, Berlin, Heidelberg, New York, Tokyo, July 2005. Springer.
2. Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5):36–41, 2003.
3. Francois Bry and alia. Rules in a Semantic Web Environment (REWERSE). EU Project 6th framework. IST-2004-506779. http://www.rewerse.net.

4. Gerardo Canfora, Félix García, Mario Piattini, Francisco Ruiz, and C. A. Visaggio. Applying a framework for the improvement of software process maturity. *Software - Practice and Experience*, 2005.
5. P. P.-S. Chen. The entity-relationship model - towards a unified view of data. *Transactions on Database Systems*, 1(1):9–36, 1976.
6. Pierre Deransart, Martin Jourdan, and Bernard Lorho. Attribute grammars - definitions, systems and bibliography. *Lecture Notes in Computer Science*, 323, 1988.
7. Vladan Devedzic. Understanding ontological engineering. *Commun. ACM*, 45(4):136–144, 2002.
8. Jean-Marie Favre. Foundations of model (driven) (reverse) engineering: Models. Technical report, ADELE Team, Laboratoire LSR-IMAG Université Joseph Fourier, Grenoble, France, 2004. vol. 1-3.
9. Jean-Marie Favre. Megamodelling and etymology - a story of words: From med to mde via model in five milleniums. In *Dagstuhl Seminar on Transformation Techniques in Software Engineering*, number 05161 in DROPS 04101. IFBI, 2005.
10. Jean-Marie Favre and Tam Nguyen. Towards a megamodel to model software evolution through transformations. *Electr. Notes Theor. Comput. Sci*, 127(3):59–74, 2005.
11. Rony Flatscher. Metamodeling in EIA/CDIF - meta-metamodel and metamodels. *ACM Trans. Model. Comput. Simul*, 12(4):322–342, 2002.
12. Peter Fritzson and Vadim Engelson. Modelica—A unified object-oriented language for system modeling and simulation. In Eric Jul, editor, *ECOOP '98—Object-Oriented Programming*, volume 1445 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 1998.
13. Félix García, Francisco Ruiz, Mario Piattini, and Macario Polo. Conceptual architecture for the assessment and improvement of software maintenance. In Piattini and Filipe, editors, *Enterprise Information Systems IV (ICEIS)*, pages 219–226. Kluwer Academic Publishers, 2002.
14. S. F. Goldfarb. *The SGML Handbook*. OUP, 1990.
15. Gerhard Goos and William M. Waite. *Compiler Construction*. Springer, Berlin, 1984.
16. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
17. Giancarlo Guizzardi, Heinrich Herre, and Gerd Wagner. On the general ontological foundations of conceptual modeling. In *ER*, number 2503 in Lecture Notes in Computer Science, pages 65–78, 2002.
18. I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
19. IEEE. Standard upper ontology knowledge interchange format. Technical report, 2003. http://suo.ieee.org/suo-kif.html.
20. ISO and IEC. Information technology - information resource dictionary system (IRDS). International Standard ISO/IEC 10027, 1990.
21. Gilles Kahn. Natural semantics. Report no. 601, INRIA, February 1987.
22. Gregor Kiczales. Aspect-oriented programming. *ACM Computing Surveys*, 28(4), December 1996.
23. Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, 1991.

24. Emma Larsdotter-Nilsson and Peter Fritzson. Using modelica for modeling of discrete, continuous and hybrid biological and biochemical systems. In *The 3rd Conference on Modeling and Simulation in Biology, Medicine and Biomedical Engineering*. The University of Balamand, may 2003.
25. UML 2.0 object constraint language (OCL) specification, 2003. http://www.omg.org/docs/ptc/03-10-14.pdf.
26. OMG. *XML Metadata Interchange (XMI)*, January 2002. http://www.omg.org/technology/documents/format/xmi.htm.
27. OMG. *MDA Guide*, June 2003. http://www.omg.org/mda.
28. Object Management Group (OMG). Common warehouse metamodel (CWM), February 10 2000.
29. Adam Pease, Ian Niles, and Teknowledge Corporation. Towards a standard upper ontology. In *FOIS*, Ogunquit, Maine, October 2001. ACM.
30. Michael Pidd. *Tools for Thinking - Modelling in Management Science*. Wiley, 2000.
31. Francisco Ruiz, Aurora Vizcaíno Barceló, Mario Piattini, and Félix García. An ontology for the management of software maintenance projects. *International Journal of Software Engineering and Knowledge Engineering*, 14(3):323–349, 2004.
32. A.-W. Scheer. *ARIS - Business Process Frameworks*. Springer, Berlin, 1998.
33. D. Schenck. The *express* language reference manual. Technical Report ISO TC184/SC4/WG1 N466 Working Document, ISO, March 1990.
34. Ed Seidewitz. What models mean. *IEEE Software*, 20:26–32, September 2003.
35. John F. Sowa. Website on Ontologies. http://www.jfsowa.com/ontology/index.htm.
36. John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., 2000.
37. W3C. Extensible markup language (XML) 1.0. Technical Report REC-xml-19980210, February 1998.
38. Niklaus Wirth. Program development by stepwise refinement. *CACM: Communications of the ACM*, 14, 1971.