On Abstraction in the OMG Hierarchy

Systems, Models, and Descriptions

Andreas Prinz
Themis Dimitra Xanthopoulou
andreas.prinz@uia.no
themis.d.xanthopoulou@uia.no
Faculty of Engineering & Science,
University of Agder
Grimstad, Norway

Terje Gjøsæter
terje.gjosater@uia.no
Faculty of Social Sciences,
University of Agder
Kristiansand, Norway

Birger Møller-Pedersen
birger@ifi.uio.no
Department of Informatics,
University of Oslo
Oslo, Norway

ABSTRACT

The Model-Driven Architecture (MDA) uses a metadata hierarchy with several layers that are placed on top of each other. The traditional view is that the layers provide abstractions related to models in languages defined by meta-models. Over the years, it has been difficult to define a consistent understanding of the layers. In this paper, we propose such a consistent understanding by clarifying the relations between the different elements in the hierarchy. This is done based on the Scandinavian approach to modelling that distinguishes between systems and system descriptions. Systems can be physical, digital, or even mental, while descriptions can be programs, language descriptions, specifications, and diagrams. We relate descriptions and systems by explaining where semantics of objects originate and how they apply in the hierarchy.

KEYWORDS

System, Model, Description, Abstraction, Semantics, Instantiation.

ACM Reference Format:
Andreas Prinz, Themis Dimitra Xanthopoulou, Terje Gjøsæter, and Birger Møller-Pedersen. 2022. On Abstraction in the OMG Hierarchy: Systems, Models, and Descriptions. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3550356.3561573

1 INTRODUCTION

Model-driven development as seen from the point of view of the object management group (OMG) is closely related to the model-driven architecture (MDA) [20]. MDA is based on different kinds of models and transformations between these. Models are formulated in modelling languages defined by meta-models. The general connection between models, meta-models, and the objects related to them is captured in the four-layer meta-modelling architecture, see Table 1. For readers more familiar with programming languages we have included how the architecture applies to grammars, including program, grammar and EBNF along with UML model, UML meta-model (defining the language UML), and MOF as a subset of UML in which UML is defined.

Table 1: The OMG view of the four-layer meta-modelling architecture

OMG Layer	UML Example	Grammar example
M3 = meta languages	MOF	EBNF
M2 = languages	UML metamodel	Java grammar
M1 = models	UML model	a program
M0 = instances	objects of classes	a run

The lowest layer of the OMG hierarchy is called M0 and it contains all sorts of *objects*. There can be physical objects like (in case of a travel agency system) passengers and flight seats, and there can be digital objects, such as objects that represent or model real passengers and seats in an airplane, and flight bookings.

In the layers M1,M2,M3 we find descriptions of those objects (often referred to as models), descriptions of how models are expressed (called meta-models), and descriptions of how meta-models are expressed (called meta-meta-models) respectively. An example of M1 would be a model of a passenger. On M2, we find languages in which to describe models - in an OMG context typically UML, but other languages as Java or Python are also possible. The layers are called Mx because they are intended to be *modelling* layers.

This architecture seems simple enough, but still the details of its understanding are far from simple. This leads to a gap between programmers and modellers, where programming and modelling are seen as different activities that are difficult to bring together. A common understanding between the two is needed [15].

The real relationship between the layers has been source of a lot of debate [1–3, 5, 6, 8, 11, 13, 16, 18, 21] and final agreement has not been achieved. The main challenge is the possibility of describing objects in UML, which would somehow be both on M0 and M1. Therefore, in the last OMG standards, the use of the architecture has been avoided, see also Section 6. Still, the relation between layers is important and a consistent understanding would be needed.

As Atkinson and Kühne observed in [4], the elements in the hierarchy can have different roles. For example, meta-models can be language descriptions, ontological descriptions, or perspectives on reality. We will follow these directions in more detail later when we discuss abstraction in the hierarchy.

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10.
https://doi.org/10.1145/3550356.3561573

In this paper, we sort out the different roles of the elements in the architecture by looking at the instantiation (based on semantics) relationships between layers and the abstraction relationships within layers. Our goal is to provide a simple and consistent understanding of the elements in the hierarchy and their relationships. We present both an understanding and an explanation of the layers and how they relate. In addition, we also look at the reality of the objects in the hierarchy. Important for our understanding is the distinction between *systems* (and elements of systems) and *descriptions* of systems (and elements thereof).

In Section 2, we discuss reality and introduce the notions of system and perspective, while in Section 3 we cover models, together with the abstraction relation. We then introduce the notions of descriptions and semantics (Section 4), and how descriptions lead to systems; followed by a discussion (Section 5) of the OMG hierarchy based on those relations. Finally, in Section 6 we discuss our findings, and in Section 7 we summarise and conclude.

2 SYSTEM AND PERSPECTIVE

A *system* can be a part of reality or, as it happens in computer science, it can be an application within an application domain made by systems and software engineering. The aforementioned application has a purpose and required functionality. From [22] we

Figure 1: Sample system 1: Mini

borrow the example of a Mini as a system, see Figure 1. We add a travel agency as another example of a system, see Figure 2. The

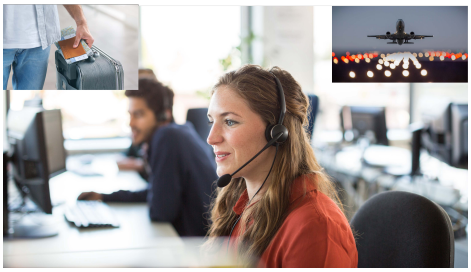

Figure 2: Sample system 2: Travel agency

travel agency is an example of a real system of which we would like to make a (simulation) model. Applications (e.g. a travel agency software system) are also part of reality, and during engineering it is useful to make models of parts of the application domain, of elements of reality that will be influenced by the application, and even of the application itself (planned systems).

In our understanding, systems can be dynamic, have a structure, and boundaries. In our context, dynamic means that a system evolves over time and changes its state. Moreover, the system structure is given by contained objects and this is evolving as well. System boundaries imply that we can distinguish elements from inside and from outside the system.

In order to be able to comprehend and work with systems we identify boundaries of the system, parts of the system, and properties of these parts. Following the object-oriented approach (see below), these parts are objects, and it leads to the following definition of system, see also [15].

Definition 2.1 (System). A system is a changing set of executing objects and their properties. These objects interact with each other and with entities in the environment of the system resulting in changes of the objects and properties. Objects may be existing entities like devices, and they may be entities that have to be made as part of the systems development. This way, a system is a set of possible executions, i.e. a set of object configurations that exist at different time points.

In order to understand systems we use the notion of perspective. Typically, we understand that systems have a relatively stable structure of their included objects. Such a definition can work in an object-oriented setting, where we have objects with properties. However, reality in general does not have such an implied structure and identification of objects and parts of reality is typically man-made. It is given by the way we look at reality, not by reality itself. For this paper, we define perspective as the structure we apply on reality.

Definition 2.2 (Perspective). A system is a part of reality perceived using a *perspective*, which is a structure imposed on reality.

Perspective is closely related to theory [17], which provides a shared perspective for a field of discourse. In our context, perspective can be a theory, but also a more individual way of looking at reality.

In general, it is possible to employ different *perspectives* on the same part of reality. This is both done to reduce the complexity of reality and to view reality related to different purposes. Each perspective can be thought of as a pair of glasses that is used to observe reality. Figure 3 illustrates that a system is a part of reality with a perspective that works as a filter only showing the relevant aspects.

We distinguish between two kinds of perspectives. First, there is a basic perspective related to our way of looking at the world and answering the question: What are the things we expect to see in reality? We have chosen to employ a basic (philosophical) perspective based on phenomena and concepts. This perspective is the basis for object-orientation when we in short consider that phenomena and concepts are represented by objects and classes, respectively.

Using an object-oriented perspective does not mean that we include all possible objects in the perceived system (existing or planned). That is to say, a lot of objects are ignored. This ignorance leads to a second kind of perspective which is based on the purpose

Figure 3: Reality seen through different glasses with specific perspectives on the Mini; a physical view with atoms, the electrical system, and the engine.

of the system, i.e. why we are looking at the system, see again Figure 3. In this paper, we consider the object-oriented perspective as given and we mostly discuss the second kind of perspective based on the purpose of the system.

From our point of view, perspective is not the same as abstraction. It would be possible to consider perspective to be some kind of abstraction. After all, perspective reduces reality to something manageable. We think that this is not completely correct, as a system is not only reduced reality. This is because without the perspective, reality is not even observable. So, it is difficult to say that perspective is an abstraction of the unstructured reality, exactly because there is no structure in reality per se.

However, it is well possible to have abstractions between different perspectives of the same reality. For example, the car audio view of the Mini can be an abstraction of the general electrical view of the same Mini. This way, perspective is the realisation dimension as observed by Jean-Marie Favre in [12]. Reality is then the realisation of the systems. Even though it is a dimension, there are only two end points in this dimension: reality and systems. Please remember that there is only one reality, but there can be several systems based on perspective and on the selection of part of reality. We will discuss possible (abstraction) relations between systems in the next section.

3 MODEL AND ABSTRACTION

A model in our sense is an abstraction of a system, which we call 'referent system'. In a way, the model refers to the referent system. Sometimes, the referent system is also called target system. A model is also a system in itself. A system becomes a model by having a relationship to the original system it is a model of.

Definition 3.1 (Model). A model is a *system* that is in the model-of relationship to a referent system, existing or planned, where the model-of relationships means that the model is *analogous* to and *more focused* than the referent system.

Two systems are analogous when they use a similar perspective, and a system is more focused than another (analogous) system, if it uses fewer objects and properties.

Please note that this definition of model is independent of the purpose of the model (decision making, system creation, understanding, etc.). The purpose is taken care of by the perspective used for the systems.

This definition might seem trivial, but we will discuss in the next Section 4, which artefacts we do not consider to be models. In particular, a UML diagram is part of a model according to OMG but is *not* part of a model according to our definition of model, but rather part of a *model description*. We discuss descriptions in Section 4.

Staying with our examples, Figure 4 shows a (physical) model of the Mini from Figure 1, made with the purpose of being played with by children. It is easy to see that both are systems, and that they

Figure 4: A Lego Mini as a model of the Mini

are analogous, which means that their perspective is overlapping. The Lego Mini is more focused in that it has less detail than the original. A similar case can be made for an online booking portal as shown in Figure 5 which is a model of the travel agency from Figure 2. Again, both are systems with matching perspectives, and there is some extra focus in the model.

By our definition, each model is a system, but not every system is a model. As a model is a system itself, it is a set of all possible

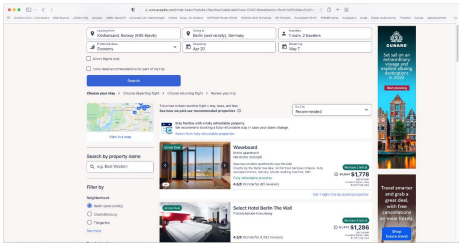

Figure 5: Expedia as a model of the travel agency

executions by our definition of system. In [22], the same argument is made, i.e. that models are program executions. We apply this understanding to the meta-model architecture, which is not addressed in [22]. In particular, we use this understanding to identify different kinds of abstraction in the meta-model architecture.

Thalheim introduces the following properties of models in [26].

(1) well-formed
(2) analogous
(3) more focused (simpler, truncated, more abstract, reduced)
(4) dependable (fit for its purpose)

We will come back to well-formedness in the next Section, while analogy and focus are already given in the definition. Dependability of the model is part of the model being a system, given by the perspective of the model. In fact, the focus property is optional for being a model, as it is already implied in the model and the referent system being systems. The application of a perspective already implies focus.

For the *general notion of model*, it is generally agreed that a model is an abstraction of an existing or planned system. Therefore, only a subset of the properties of a system are represented in the model. One of the uses of models is to *experiment* with the models instead of with the real systems and deduce properties for the systems, as long as the experiments are possible in the abstracted view of the model. This idea of model works for material, mathematical or digital models. It is obvious that experiments are related to the executions of the system, for example validation is the check whether the executions of a model and its related system match.

A kid can experiment with a Lego Mini car to imitate driving, thereby interacting with the model (the Lego Mini car), its position, orientation, and velocity, while imitating how a car turns around. Playing with the car simulates the execution, which is often matching the possible executions of the Mini system from Figure 1. Of course, the kid can let the Lego Mini fly, which would not match with the real Mini.

Similarly, the booking portal acts as a model of a physical travel agent. It also functions as a system in itself, facilitating much of the same functionality as the physical travel agency.

This way, physical models are systems, because it is their behaviour (their executions) that makes them systems. Scale models are also systems, but often with a fixed object structure.

Please observe that the abstraction property (more focused) of a model is heavily dependent on the perspectives of model and referent system. This way, simplification is somewhat outside the model-of relation. Anyway, the relation between model and referent system is an abstraction relation and it is characterised by the model being analogous, and more focused than the referent system.

The importance of analogy is also identified in [19] by stating "... abstraction is the result of recognising and isolating what different concrete concepts have in common."

4 DESCRIPTION, INSTANTIATION, SEMANTICS

More often than not, systems are coming to exist based on system descriptions. The most obvious case is programming, where the system is the set of executions, and the description is a program. Alternative kinds of descriptions could be specifications, sets of statements, diagrams, or formulae.

In many cases, models are understood as being descriptions of (referent) systems. Here we can remember the characterisation as [26] that a model is well-formed. Well-formedness is a clear property of a description, not of a system. It would not apply to scale models, for example, but to descriptions of those. Also the UML language specification [24] considers a model to "make some statements of interest about that system ... from a certain point of view and for a certain purpose".

In our approach, the description is *not* the model, but describes the model. For example, Figure 6 provides a partial description for

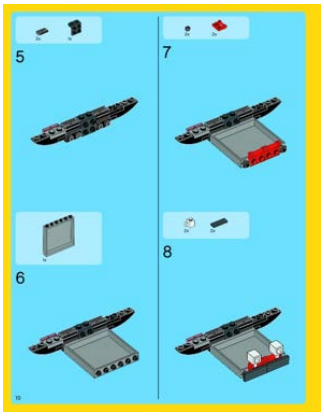

Figure 6: Partial description of the model of the Mini.

building the Lego-Mini. It is obvious that the description is not the system itself (the Lego Mini), but it provides a way to create the system. In the same sense, the description of a system is *not* a model of the system, but a way to create another system that then is the model.

This might be easier to understand by looking at Figure 7, which provides a partial description of the database structure of the booking portal. This description is obviously not the database itself, but a way to create and handle the database. The database itself is then the model of our intended reality.

We could even use the term "prescription", as the system description leads to (prescribes) a system as shown in Figure 8. Systems can be made using various kinds of *system descriptions* from which their executing objects are created. A similar argument can be made for mathematical models.

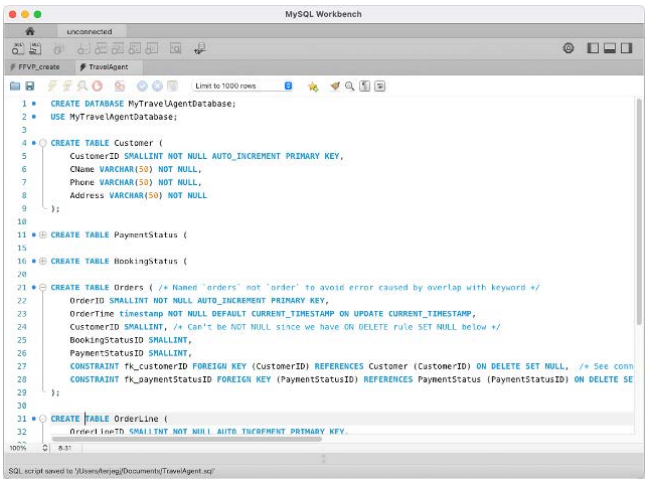

Figure 7: Description of the database of the booking portal.

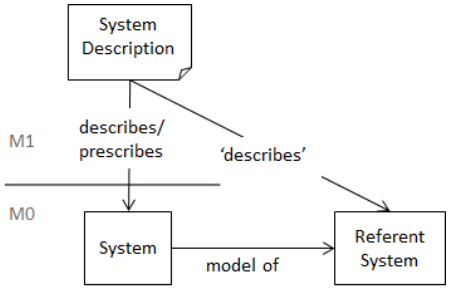

Figure 8: Prescriptions lead to systems, which can be models.

It is very common to associate the description with its implied system, for example the database description is often considered as if it were already the database that it describes. In this paper, we follow the Scandinavian approach to modelling [14] by distinguishing between description and implied system.

Among the descriptions, programs (or specifications) are *descriptions* of systems as possible program executions. A program execution will consist of a changing structure of computational objects according to the descriptions of objects and classes in the program.

Definition 4.1 (System description). A system *description* is a well-formed description of all possible static structures that may exist during run time of the system together with their possible dynamic development.

A description is given in a language as shown in Figure 9. Well-formedness means that the description adheres to the rules of the language. Sometimes, this adherence is called static correctness. Please note that the language can be both formal or informal, leading to different formality levels of the well-formedness.

As can be seen in the lower part of Figure 9, a system description always leads to a system, which is a set of possible executions. The system does not need to be a model if there is no related referent system. This connection between a system description and the system itself is typically called *instantiation*, see also [9, 12]. It will not only work on the system as a whole, but for all the elements of the system as well. Instantiation is based on the semantics of the language in which the description is made, which provides a set of possible instances[1]. Instantiation will select one instance from the set of possible instances. This relation is the classical "meta"-relation as for example stated in [8]. It is also closely related to the mathematical 'interpretation' relation. This leads to the following definition of semantics.

Definition 4.2 (Semantics and instantiation). Semantics is the relation between a (system) description and its prescribed possible systems. Selecting one specific system from the possible systems is called instantiation.

How does this magic of instantiation work? The answer is, that it is defined by the language (which we call layer L) that is used for the description (called layer D), see Figure 9. Again, the semantics can be formal or informal, depending on the language. Each lan-

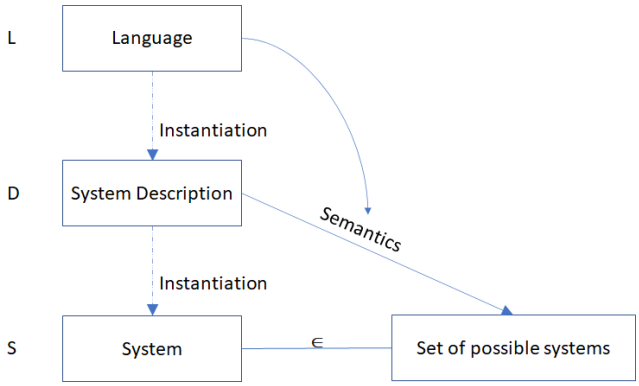

Figure 9: Semantics of the language relates system description and system.

guage defines the semantics of its elements, e.g. what are instances of classes, modules, methods, and variables. This is the known aspect of compilers called runtime environment (RTE). RTE states describe the situations we can encounter at runtime and runtime state changes describe the dynamics at runtime. How RTE states and state changes can be described for formal languages is shown in [25].

This means that the instantiation semantics of a language (RTE structure) is defined on layer L as a mapping from any description on layer D to appropriate structures on layer S (for systems). The execution semantics of a language assembles RTE structures into sequences of structures forming runs. Again, this is a part of the language description on layer L which defines the possible runs for any description on layer D, leading to runs on layer S as depicted in Figure 9. This three-layer structure L-D-S is always there when we talk about instantiation.[2]

[1]Please note that the kind of instances depends on the kind of description. Instances of classes are objects, while instances of system descriptions are sets of possible runs.
[2]The L-D-S structure can be seen as a way of deep instantiation [7], because the instantiation between D and S is described already in L. In this understanding, instantiation from the language level is always deep.

The UML 2.4.1 Infrastructure specification [23] agrees by stating that when dealing with meta-layers to define languages there are generally three layers that always have to be taken into account:

(1) the language specification, or the meta-model,
(2) the user specification, or the model[3], and
(3) objects of the model[3].

From the relation between system description and system it is clear that the system description is *not* an abstraction of the system - rather, it is a *description* of the system. There is some abstraction in the diagonal relation in Figure 8, but the plain abstraction relation is horizontal.

5 HIERARCHY

OMG has defined a model-driven architecture (MDA) for using models in the development process, and together with this the notion of models and meta-models, see Figure 1 and [20]. After the discussion in Section 4, we know that the relation between M1 and M0 is the instantiation relation, where M1 contains descriptions and M0 contains instances according to these descriptions. In this context, the descriptions on M1 are fixed (read-only).

Essential in our understanding of the modelling layers is that of semantics and instantiation, and this is further coupled with our understanding of systems and descriptions. As we shall see in the sequel, this understanding applies to all layers and not just the M1-M0 crossing, see Figure 10.

The key observation is that a description (on M1) is realised by objects (still on M1 - i.e. instances of meta-classes at M2) and thereby part of a system. This system is not the one being described by the description, but rather the system where the description itself can be handled by changing the objects representing the description, e.g. editing. The objects on M1 are instances of the language description, which is on M2. In Figure 10, there are objects on the right-hand side, and descriptions on the left-hand side. The connection between right-hand side and left-hand side is realisation (or representation). According to our discussion in Section 2, there would be even a far left side which contains the reality underlying the systems and objects. This way, the horizontal dimension in Figure 10 is the realisation dimension.

In the following we indicate with (L), (D) or (S) how the layers of Figure 9 correspond to the various OMG layers. Please note that the instantiation goes one layer down from the left to the right. In summary, that means for the M0-M1 crossing, see also Figure 10.

- Descriptions on M1 (D) are represented by objects (of classes from M2 (L)). These descriptions describe systems.
- A system description on M1 (D) is instantiated to a system on M0 (S), with all its possible runs, based on the possible objects in RTE states.
- Objects on M0 (S) are generated from the descriptions on M1 (D), according to the semantics of the language defined on M2 (L).

A similar situation appears for the crossing between M2 and M1, with semantics from M3.

- Descriptions on M2 (D) are represented by objects (of classes from M3 (L)). These descriptions describe languages.
- A language description on M2 (D) is instantiated to a system on M1 (S) - typically called IDE, based on the possible specifications (system descriptions that can be created in the described language). This way, a language (instance of a language description) is a set of system descriptions.
- Objects on M1 (S) are generated from the descriptions on M2 (D), according to the semantics of the (meta-)language defined on M3 (L).

As an example, in Figure 10 the description on M2 defines UML, in terms of objects of the (meta-)classes on M3 (MOF). The corresponding system on M1 is the system (the language) generated from the description of UML (i.e the UML language as a system), given the semantics of MOF. The different runs of this system will have (run time) objects that represent different UML specifications on M1.

Because the semantics of MOF is more or less plain instantiation, different runs of the UML language system will just produce different object structures representing different UML specifications on M1. Runs of the UML system will typically be performed by a tool for handling UML descriptions (editors and the like).

We can apply the same situation again for M2, M3, and M4. Obviously, the languages on M4 are the same as the languages on M3, as those are languages to define languages. Therefore, we would normally stop at M3 and consider the upper layers to be repetitions of M3, i.e. M3=M4=M5=M6=... This method is normally called bootstrapping, for example when writing a Java compiler in Java.

There is also a similarity between the layers M2 and M3, as both of them contain descriptions of languages. All meta-languages are also languages by definition. Still, not all languages are meta-languages and therefore M3 is a subset of M2, and not the same.

6 DISCUSSION

In this section, we will discuss several aspects of our approach. First, we look into the difference between a model and a description of a model, then we consider the difference between meta-data and meta-models, and finally, we discuss the importance of well-defined terms.

6.1 Is a Description a Model?

In our approach, every model is a (model) system in itself, which becomes a model by having an abstraction relationship to a referent system. The (model) system can be physical and tangible, but it can also be digital or mental. Similarly, also the referent system can be physical, digital, or mental. This way, a model is *always* on the same modelling layer as the referent system. We compare this to the OMG idea in Table 2. Obviously, there is no difference in the idea of M0. The main difference is that according to our understanding, all layers above layer M0 are description layers. Their meaning is given by instantiation and is placed on the layer below, see again Figure 8.

Distinguishing between models and model descriptions allows a unified handling of models with and without descriptions. In the general understanding of modelling, most people would agree that physical models (e.g. the Lego Mini) and systems (e.g. the real Mini) are obviously on the same layer. A system being a model is just

[3]Note that this is OMG notation, where 'model' is in fact a 'model description'.

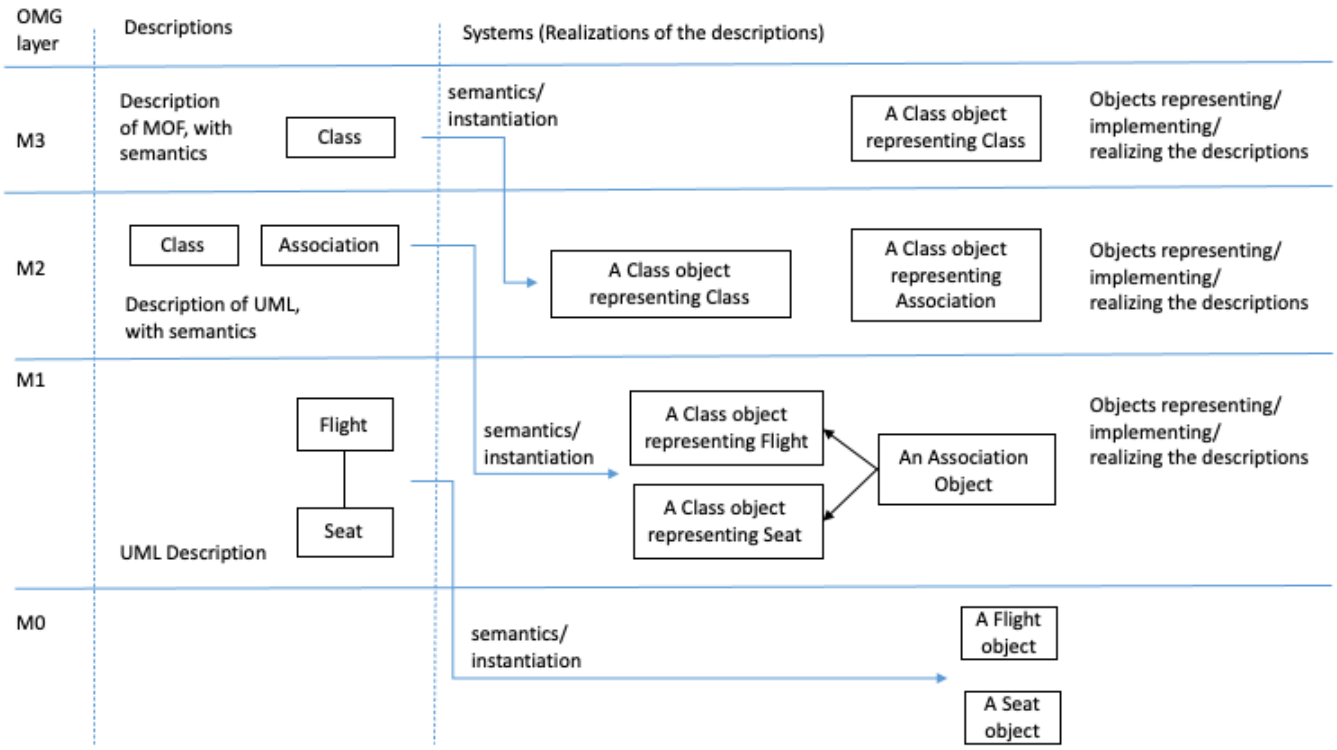

Figure 10: Descriptions are also systems.

Table 2: Our view compared with the OMG view of the four-layer architecture

Layer	OMG terms	Our terms
M3	meta-languages	descriptions of meta-languages
M2	languages	descriptions of languages
M1	models	descriptions of systems & models
M0	objects of classes	systems, models, objects

as physical: it contains objects with identity, state and potential actions, and these model/represent elements of the real systems.

When models are created from descriptions, we tend to call the descriptions for models and even think of them as models. In this view, we consider architectural drawings to be models of the buildings, UML diagrams to be models of the systems, and music sheets to be models of the music.

However, when we look more closely, we see that the real connection is *not* from the description to the referent system, but from the description to the implied system (by instantiation semantics) and then to the referent system (by abstraction), see again Figure 8. The model description is connected to the referent system indirectly via an instantiation and a model-of.

Separating model-of from instantiation also clarifies the difference between linguistic and ontological instantiation as introduced by [5]. We already discussed linguistic instantiation, which is the relation based on semantics. Ontological instantiation is the intra-layer relation between descriptions of classes (e.g. in class diagrams) and descriptions of objects (e.g. in object diagrams). As the relation is between descriptions of classes and descriptions of objects (not between a class and an object), this is not instantiation at all. Instead, it is a relation between the semantics of a class (which is an object) and the semantics of an instance specification (which is also an object).

6.2 Meta-data or Meta-model?

Even though the four-layer architecture should have a prominent place in the OMG standards, binding together languages and technologies, it does in fact only appear as a side-remark in the OMG standards. Moreover, its role is played down more and more for the new versions of the central standards UML [24] and MOF [10].

We see a major problem in that the architecture is designed to be a *meta-data* architecture, as stated in [10].

> One of the sources of confusion in the OMG suite of standards is the perceived rigidness of a 'Four layered metamodel architecture' that is referred to in various OMG specifications. Note that key modeling concepts are Classifier and Instance or Class and Object, and the ability to navigate from an instance to its metaobject (its classifier). This fundamental concept can be used to handle any number of layers (sometimes referred to as metalevels).

Handling meta-data instead of meta-models does not take into account the dynamic aspect of systems. With meta-data, we only handle (static) snapshots and attach meta-information to them. This leads to a bottom-up approach with focus on extensibility, because the basic operation is attaching meta-data to data, which goes from objects to classes. Related to Figure 9, we see that the OMG focus removes the upper part (the language), in particular the semantics. Instantiation semantics is fixed in this approach to be connection between object and class. This is the reason that MOF can be used in a two-layer architecture.

Handling languages requires a different approach, where the starting point is the language and well-formedness is essential. Such an approach is more top-down and not overly interested in extensibility. For meta-modelling, the main operation is instantiation, which is going from classes to objects. When meta-modelling is involved, it is essential to have at least three layers as shown in Figure 9.

It appears that OMG has tried to capture both meta-models and meta-data in the same architecture, leading to many misunderstandings. A decision towards a meta-model architecture would improve and clarify the OMG standards.

6.3 Calling a Spade for a Spade

In [4], a point is made that it is important to capture all the *possible* meanings of a term when naming something. We agree wholeheartedly. We would like to add that it is essential to identify all *different* meanings of a term, when doing this work. For example, there are two different meanings of the term 'spade', see figure 11.

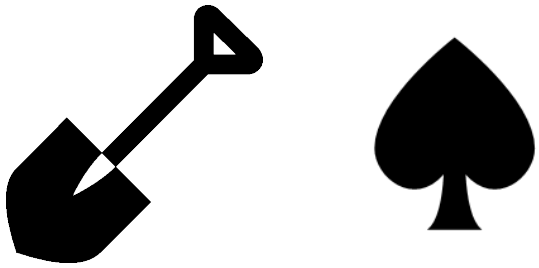

Figure 11: A spade is a spade, or is it?

Although it is important to be aware of the different semantics of some term, it is also and maybe even more important to ensure that the semantics is precise. When there are two or more different concepts denoted by the same term, then it is wise to introduce distinguishing terms, for example to distinguish the two types of spade in Figure 11. This precision is even more important when the terms are used to create an architecture that should unify different technologies, tools, and theories.

This is done in our article. We have identified three different concepts that each might be called abstraction. We have defined each of them and the we use *different* terms in order to distinguish them: instantiation which crosses layers, model-of which is on the same layer, and perspective which relates systems to reality. All three could be subsumed into the term 'abstraction'.

When separating model-of from instantiation, it appears clear that instantiation is not an abstraction relation, while model-of is an abstraction relation. Furthermore, we argued already in Section 2 that perspective is not really an abstraction as the reality is not an entity we can handle before we apply the perspective. This way it is difficult to state from what a system is an abstraction of. This way, we conclude that model-of is the only true abstraction relation in the hierarchy.

7 SUMMARY

We have presented a simple understanding of the layers using three main observations.

(1) Systems are parts of reality viewed under a perspective.
(2) Models are special systems that are abstractions of referent systems.
(3) Systems (including models) can be instantiated from descriptions by using the semantics of the language of the descriptions.

With these observations, models and their referent systems are placed on the same layer and connected with the model-of relation (abstraction), which is a horizontal relation. Descriptions are placed one layer higher than their systems and instantiation crosses the layer structure. This makes instantiation a vertical relation.

Instantiation connects descriptions with their systems. It is given by a semantics description, which is placed one layer above the descriptions. From this consideration, we need at least three layers: languages, descriptions, systems.

The elements given here provide a clean and straightforward approach to modelling - also known as the Scandinavian approach to modelling - and they are compatible with the four-layer architecture. The critical change to current practice is the understanding that UML models and most other kinds of non-physical models are actually *model descriptions*, and they do not directly relate to their referent systems. Instead, via instantiation they create a model which is a model-of the referent system (abstraction).

This approach to modelling is easily compatible with the standard approach to programming, where the code (the description) is used to produce the executions. This way, our paper also explains how programming and modelling are similar and can be used together.

It would be useful to consider calling the descriptions for description, but it is important to at least understand that they are not in a model-of relationship with the referent system themselves, even if we call them models.

REFERENCES

[1] Colin Atkinson. 1997. Meta-modeling for distributed object environments. In *In Enterprise Distributed Object Computing*. Published by IEEE Computer Society, Gold Coast, Australia, 90–101.
[2] Colin Atkinson and Thomas Kühne. 2000. Meta-level Independent Modelling. In *International Workshop on Model Engineering at 14th European Conference on Object-Oriented Programming*. Workshop Proceedings, Sophia Antipolis and Cannes, France, 1–4.
[3] Colin Atkinson and Thomas Kühne. 2002. Rearchitecting the UML infrastructure. *ACM Transactions on Computer Systems (TOCS)*, 12, 4 (October 2002), 290–321.
[4] Colin Atkinson and Thomas Kühne. 2003. Calling a Spade a Spade in the MDA Infrastructure. In *Proceedings of the Metamodeling for MDA First International Workshop*. Univ. of York, workshop proceedings, York, UK, 9–12.
[5] Colin Atkinson and Thomas Kühne. 2003. Model-Driven Development: A Metamodeling Foundation. *IEEE Software* 20, 5 (2003), 36–41. https://doi.org/10.1109/MS.2003.1231149

[6] Colin Atkinson and Thomas Kühne. 2005. Concepts for Comparing Modeling Tool Architectures. In *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3713)*, Lionel C. Briand and Clay Williams (Eds.). Springer, Montego Bay, Jamaica, 398–413. https://doi.org/10.1007/11557432_30

[7] Colin Atkinson and Thomas Kühne. 2018. *Deep Instantiation.* Springer New York, New York, NY, 1040–1041. https://doi.org/10.1007/978-1-4614-8265-9_80608

[8] Jean Bézivin and Olivier Gerbé. 2001. Towards a Precise Definition of the OMG/MDA Framework. In *16th IEEE International Conference on Automated Software Engineering (ASE 2001), 26-29 November 2001, Coronado Island, San Diego, CA, USA*. IEEE Computer Society, Coronado Island, San Diego, CA, USA, 273–280. https://doi.org/10.1109/ASE.2001.989813

[9] Tony Clark, Paul Sammut, and James S. Willans. 2015. Applied Metamodelling: A Foundation for Language Driven Development (Third Edition). *CoRR* abs/1505.00149 (2015), 1–244. arXiv:1505.00149 http://arxiv.org/abs/1505.00149

[10] OMG Editor. 2019. *OMG Meta Object Facility (MOF) Core Specification Version 2.5.1 (OMG Document formal/2019-10-01).* Technical Report. Object Management Group.

[11] Owen Eriksson, Brian Henderson-Sellers, and Pär J. Ågerfalk. 2013. Ontological and linguistic metamodelling revisited: A language use approach. *Information and Software Technology* 55, 12 (2013), 2099–2124. https://doi.org/10.1016/j.infsof.2013.07.008

[12] Jean-Marie Favre. 2003. Meta-Model and Model Co-Evolution within the 3D Software Space. In *Proceedings of ELISA 2003*. Royal Netherlands Academy of Arts and Sciences, Amsterdam, The Netherlands, 98–109.

[13] Jean-Marie Favre. 2005. Foundations of Meta-Pyramids: Languages vs. Metamodels – Episode II: Story of Thotus the Baboon. In *Language Engineering for Model-Driven Software Development (Dagstuhl Seminar Proceedings (DagSemProc), Vol. 4101)*, Jean Bezivin and Reiko Heckel (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 1–28. https://doi.org/10.4230/DagSemProc.04101.7

[14] Joachim Fischer, Birger Møller-Pedersen, Andreas Prinz, and Bernhard Thalheim. 2020. Models Versus Model Descriptions. In *Modelling to Program - Second International Workshop, M2P 2020, Lappeenranta, Finland, March 10-12, 2020, Revised Selected Papers (Communications in Computer and Information Science, Vol. 1401)*, Ajantha Dahanayake, Oscar Pastor, and Bernhard Thalheim (Eds.). Springer, Lappeenranta, Finland, 67–89. https://doi.org/10.1007/978-3-030-72696-6_3

[15] Joachim Fischer., Birger Møller-Pedersen., and Andreas Prinz. 2020. Real Models are Really on M0 - Or How to Make Programmers Use Modeling. In *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development - MODELSWARD,*. INSTICC, SciTePress, Valletta, Malta, 307–318. https://doi.org/10.5220/0008928403070318

[16] Ralf Gitzel, Ingo Ott, and Martin Schader. 2007. Ontological Extension to the MOF Metamodel as a Basis for Code Generation. *Comput. J.* 50, 1 (2007), 93–115. https://doi.org/10.1093/comjnl/bxl052

[17] Shirley Gregor. 2006. The Nature of Theory in Information Systems. *MIS Quarterly* 30, 3 (2006), 611–642. http://www.jstor.org/stable/25148742

[18] Wolfgang Hesse. 2006. More matters on (meta-)modelling: remarks on Thomas Kühne matters. *Software and Systems Modeling (SoSyM)* 5, 4 (December 2006), 387–394. http://dx.doi.org/10.1007/s10270-006-0033-9

[19] Douglas R Hofstadter. 2013. *Surfaces and essences : analogy as the fuel and fire of thinking.* Basic Books, New York.

[20] Anneke Kleppe and Jos Warmer. 2003. *MDA Explained.* Addison–Wesley, Boston, MA, USA.

[21] Thomas Kühne. 2006. Matters of (Meta-) Modeling. *Software and Systems Modeling (SoSyM)* 5, 4 (December 2006), 369–385. http://dx.doi.org/10.1007/s10270-006-0017-9

[22] Ole Lehrmann Madsen and Birger Møller-Pedersen. 2018. This is not a model : On development of a common terminology for modeling and programming. In *Proceedings of the 8th International Symposium, ISoLA 2018: Leveraging Applications of Formal Methods, Verification and Validation - Modeling, Lecture Notes in Computer Science 2018 ;Volume 11244 LNCS*. Springer, Limassol, Cyprus, 206–224. https://doi.org/10.1007/978-3-030-03418-4_13

[23] OMG Editor. 2011. *Unified Modeling Language: Infrastructure version 2.4.1 (OMG Document formal/2011-08-05).* Published by Object Management Group, http://www.omg.org.

[24] OMG Editor. 2017. *Unified Modeling Language: Infrastructure version 2.5.1 (OMG Document formal/2017-12-05).* Published by Object Management Group, http://www.omg.org.

[25] Markus Scheidgen and Joachim Fischer. 2007. *Human Comprehensible and Machine Processable Specifications of Operational Semantics.* Springer Berlin Heidelberg, Berlin, Heidelberg, 157–171. https://doi.org/10.1007/978-3-540-72901-3_12

[26] Bernhard Thalheim. 2018. *Conceptual Modeling Foundations: The Notion of a Model in Conceptual Modeling.* Springer New York, New York, NY, 559–561. https://doi.org/10.1007/978-1-4614-8265-9_80780