Models versus Model Descriptions

Joachim Fischer[1] [0000-0003-2476-3996], Birger Møller-Pedersen[2] [0000-0003-2123-3260], Andreas Prinz[3] [0000-0002-0646-2877], and Bernhard Thalheim[4] [0000-0002-7909-7786]

[1] Department of Computer Science, Humboldt University, Berlin, Germany
fischer@informatik.hu-berlin.de
[2] Department of Informatics, University of Oslo, Oslo, Norway birger@ifi.uio.no
[3] Department of ICT, University of Agder, Grimstad, Norway
andreas.prinz@uia.no
[4] Department of Computer Science, University Kiel, Germany
bernhard.thalheim@email.uni-kiel.de

Abstract. In the development of computer-based systems, modelling is often advocated in addition to programming, in that it helps in reflecting the application domain and that it makes the design and experiment activities of development more efficient. However, there is disagreement about what models are and how they can be used in software systems development. In this paper, we present the Scandinavian approach to modelling, which makes a clear distinction between models and model descriptions. This paper explains the connections between models, descriptions, systems, and executions. Combining the Scandinavian approach with the Kiel notion of model, we establish that both descriptions and executions are closely connected instruments with different roles. The paper argues that (program) executions are the models of dynamic systems, not their descriptions in terms of diagrams and text. So in a general sense programming is about writing descriptions for systems. In particular the paper clarifies when programming is also modelling.

Keywords: model · system · description · execution · semantics.

1 Introduction

The development of computer-based systems brings together different areas of experience, methods and terminology. Therefore, it is not surprising that terms have different meanings in different contexts. Surprisingly, even very basic terms such as system, model, model description, modeling and programming are affected. In the development of computer-based systems, three methodologically different computer-related disciplines meet. First, there are engineers who use computer models to design new technical systems, simulate and test them before actually building them. Second, there are IT experts who design software systems from abstract models by means of extensive transformations. Third, there are other IT experts who design and implement systems directly using specific

programming languages and techniques. Each of these three groups use programming and modelling, and all of them have slightly different understanding of what a model is. The discrepancy in the conceptual perception remains with modellers and programmers even for development of pure software systems.

As part of system development, *programmers* produce running systems. They do this by programming, that is writing programs. Often, they do not subscribe to the idea of modelling as this typically implies the creation of diagrams that do not contribute to the making of programs and quickly become obsolete.

Modellers handle different kinds of *models* (domain models, requirements and design models) in a mixture of diagrams and text. It can be argued that programmers are to a certain extent also modelling, and modellers are programming. We will look at these two activities and explain how they are similar and how they are different. In this paper, we present the Scandinavian approach to modelling as a shared understanding between modellers and programmers using an understanding of models as illustrated in Figure 1.

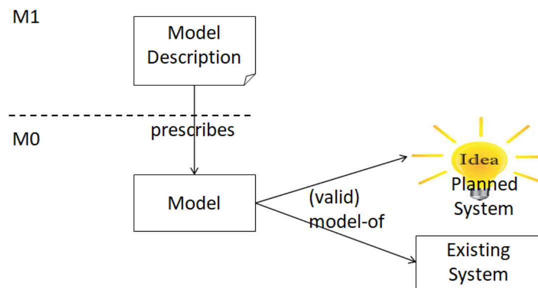

Fig. 1. Overview of the Scandinavian Approach to Modelling

The approach presented in this paper is called the Scandinavian Approach to Modelling. The Scandinavian Approach was started in the design of the language SIMULA [Dahl and Nygaard, 1965] and [Dahl and Nygaard, 1966], and developed further with the languages Delta [Holbæk-Hanssen et al., 1973], Beta [Madsen et al., 1993], and SDL-2000 [Union, 2011]. There are several articles describing aspects of the Scandinavian approach, see [Nygaard and Dahl, 1978], [Scheidgen and Fischer, 2007], [Madsen and Møller-Pedersen, 2010], as well as [Fischer et al., 2016], [Gjøsæter et al., 2016], [Prinz et al., 2016], and finally also [Madsen and Møller-Pedersen, 2018].

The paper evolves around the concept of a system, see the bottom of Figure 1. This is quite natural as systems development is about producing systems. Systems can be existing physical systems, or imagined and planned systems. Systems can be made using various kinds of *descriptions*, e.g. *programs*. These descriptions imply executions, i.e. systems. There is a dotted line in Figure 1 between the world of systems, called M0 (below the line) and the world of descriptions, called M1 (above the line).

The Scandinavian approach considers a *model* to be a (model) system, which is a model of another system called *referent* system. This referent system is an existing system in case of simulation or a planned system in case of system development. The (model) system is created by a (model) description, which is the diagram or code that describes the (model) system.

In this paper, we will focus on dynamic models. This restricted view on models helps to keep the discussion focused. Still, this is the prominent use of models in computer science and in system development. We combine the Scandinavian approach with the Kiel notion of model [Thalheim et al., 2015], which also provides a general definition of the term model.

The distinction between models and model descriptions has been made before in [Madsen and Møller-Pedersen, 2018] and in [Fischer. et al., 2020]. Here we apply the argument to the MOF architecture [Kleppe and Warmer, 2003] (see Figure 2), and we conclude that models are systems at level M0.

M3	meta-language	Example: MOF
M2	language	Example: UML
M1	model description	Example: UML diagrams for library system
M0	model execution	Example: running library system

Fig. 2. OMG four level architecture

The paper is structured as follows. In Section 2, we discuss the notion of systems as a starting point for the discussion. This is extended in Section 3 with the role that descriptions have in creating systems. Then we continue with discussing the notion of models in Section 4. We bring all these parts together in Section 5 and compare with the Kiel notion of model. We discuss the Scandinavian approach in Section 6, and summarize in Section 7.

2 Systems

Modelling and programming are used in system development, so we start by defining the concept 'system'. Interestingly, although UML claims to be a language for the modelling of systems, it does not define what a system is. In general, programming languages do not define the term system, either. However, the UML standard mentions 'running system' when talking about interactions. Thus the UML idea of systems is running systems.

System development is about making dynamic systems, i.e. systems that inherently change over time. Figure 3 shows a sample system: a room with a control system for cooling and heating. Systems are composed of parts, and the interacting parts are changing, thus bringing about the system state changes[5]. We call the system state changes the *behaviour* of the system.

[5] The state changes can be continuous or discrete. We call the systems discrete versus continuous systems, or combined systems, if both kinds of state changes appear.

Fig. 3. Systems

The parts of a system can be existing entities, like a chair in the room, and they can be planned (imagined) parts, for example the controller of the heating system. We apply the perspective of object-oriented modeling and programming, so the parts of a system are objects. The parts of a system can be systems themselves.

The system structure itself can be static or subject to dynamic changes. Of course, there is the extreme case where a system only has existing parts, for example when we create a model in order to *understand* reality. Another extreme case is a system that only contains planned parts, most often in the process of creating something new. This brings us to our definition of system, compiled mainly from [Fischer et al., 2016] and [Bossel, 2007].

Definition 1 (System). *A system is a purposeful collection of executing objects interacting with each other and with entities in the environment. It is a whole and loses its identity if essential objects are missing. A system has structure (existing and imagined objects having properties) and behaviour (executions by means of object behaviour).*

This way, a system is a set of possible executions, i.e. a set of object configurations that exist at different points in time. It has purpose, identity, structure, behaviour, and interaction with the environment.

An essential part of systems are their objects. We follow the ideas of SIMULA and UML for the definition of objects.

Definition 2 (Object, adapted from [OMG, 2017]). *An object is an individual thing with a state and relationships to other objects. The state of an object identifies the values of properties for that object. Objects may have passive behaviour in terms of methods that can be called. An object may be active, meaning that it has some autonomous behavior.*

When we want to work with a system, it is important to know what we can observe. This is given by the system *state*, which in turn is based upon the system *structure*. The structure of a system is a dynamic collection of interacting objects, each with their properties that contribute to the state of the system. With the state we can observe the system by capturing snapshots of its progression.

Systems are just some part of reality. Typically, they have *interaction* with other parts of reality, which we call *environment*. If they do, we call them open (non-autonomous) systems. If systems can exist on their own, then they are closed systems. In many cases, a system with only planned parts is an open system. In this case, it is possible to close the open system by introducing an abstract part for the environment of the system, which is an existing part. In this paper, we are mostly talking about open systems, where the system reads inputs from the environment and writes outputs to the environment.

By definition 1, not all parts of reality are systems; we have to consider a given part of reality as a system, and we abstract away unimportant parts and features of reality. There are two abstraction levels. First, we have a discourse, i.e. a way we look at reality in general, see Figure 4. The discourse starts with our understanding how reality is composed in general. For our considerations and for this paper, we consider reality to be composed of objects, following an object-oriented discourse.

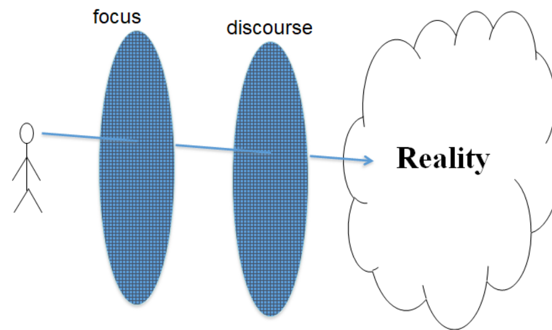

Fig. 4. Viewpoints and Discourse

In addition to the discourse, the *purpose* of the system will add another level of abstraction. This is our focus when looking at the system, and it reduces the aspects to look at. The set of five ODP-RM viewpoints [ITU-T, 1995] is an example of a high-level classification of the purpose of systems for a special class of complex system.

System identification always requires an abstraction of reality, which depends on the purpose of the system. This means that, with a completely identical system purpose, different systems can be created as object configurations. For example, the content of a boiler can be measured by the filled volume or by the water level.

All in all, we conclude that systems are part of reality (really existing or hypothetically assumed), and this means they are placed on OMG level M0.

3 System Descriptions

After we have clarified the notion of system, we will ask ourselves where the systems come from. In many cases, systems are created by making descriptions of them. In this section, we discuss these system descriptions.

3.1 Systems and Descriptions

Many systems are based on system descriptions, see Figure 5. A typical example

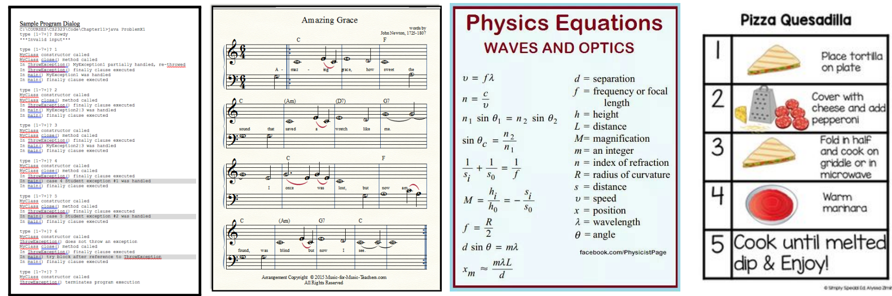

Fig. 5. Different system descriptions

is a system that is based on a program in some programming language. The program is the description, and the description leads to a running system on a real hardware. A similar example is a music sheet that describes, i.e. is a description of, a musical performance.

The system described by the description can also be an imaginary system, as for example given by physics formulae. Their meaning creates a virtual world, i.e. a virtual system. In a similar sense, a recipe (description) creates an imaginary sequence of steps for a dish that can be used to create the real dish. In all these cases, the various kinds of *system descriptions* result in running systems, see Figure 6. As discussed before, the running systems consist of executing objects.

All these examples show that the description is not a system in itself: it is not composed of objects and it does not have behaviour. The system related to a description is implied by the semantics of the language of the description and thereby the meaning of the description. It has become customary to call this connection between the description and the system for "prescribe", and then the description is rather a prescription, see again Figure 6. A similar approach is used in [Seidewitz, 2003]. This allows us to define programming.

Definition 3 (Programming). *Programming is the activity to produce descriptions (prescriptions) in order to produce systems. In a narrow sense, programming is concerned with prescriptions in the form of computer programs. In the general sense, programming means to construct a description in order*

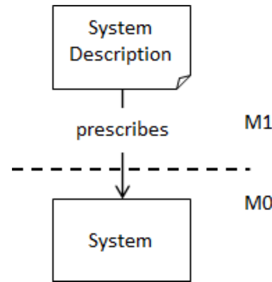

Fig. 6. Prescriptions

to create a system, as designing recipes, composing music, or inventing physics formulas.

There is a vast body of knowledge as to which steps to take in order to create good descriptions, and computer engineering explain how to do this for computer systems. In this article, we are not concerned how the description comes about, but it is essential to agree that the description is created in order to create a system.

Coming back to the OMG architecture, we notice that a description is placed on level M1, independent of the kind of description. The description then implies a number of possible executions consisting of a changing structure of objects according to the prescription. These objects, i.e. the implied system, will be on M0. The objects behave according to their prescription of behaviour as part of the description on M1.

This approach divides the world into descriptions and objects, as shown in Figure 7. Below the line, on M0, there are objects, there are states of objects,

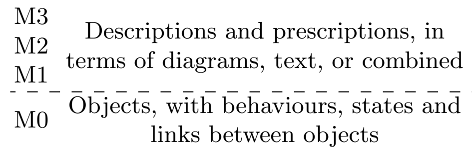

Fig. 7. Descriptions and Objects

and there are state changes resulting from the behaviour of the objects.

Above the line, on M1[6], there are *no* objects, there are *no* states, and there are *no* state changes. However, there are *prescriptions of objects* (e.g. by means of classes), there are *prescriptions of states* (e.g. descriptions of attributes of objects), and there are *prescriptions of state changes* (e.g. assignments).

[6] There are also descriptions and prescriptions on M2 and M3, but this is out of the scope of this paper.

In the following, we will use the term description for both description and prescription, except in cases where it is essential to distinguish. The main difference is between objects (on M0) and descriptions of objects (on M1-M3).

3.2 Semantics and Meaning

As already stated in [Fischer et al., 2016], it is the semantics of the language of the description defining the system belonging to the system description. Semantics is not about formality in the first place, even though system engineering is most interested in formal description languages and programming languages with a precise semantics.

In the OMG architecture, this is the move from descriptions on level M1 to objects and executions on level M0. This way, semantics is a *vertical* relation (crossing levels). The description is executed in some abstract machine. In reality, there might be several transformations (horizontal semantic steps) before the vertical step appears. In the world of computers, the most common way to provide transformation semantics is using a compiler, which is essentially replacing one description by another, more executable description.

Still, at the end we need to come to a description that can be executed, i.e. that can cross the level. We define the idea of semantics as follows.

Definition 4 (Semantics). *Semantics is the relation between a (system) description and its prescribed possible executions (the system)*[7]*. In our context, semantics is the same as the prescribe relation.*

The semantics of a specification is given by the language it is written in; it is not a property of the specification itself. The semantics as given by the language (on level M2) connects the system specification on level M1 to the implied system on level M0.

In this consideration, it is irrelevant whether the semantics is formal or not. The important point is that there is a system implied by the description. A programming language provides more formality than a music sheet, where the conductor can add some interpretation. In both cases, the description has a meaning.

Semantics provides two parts: structural semantics and dynamic semantics. *Structural semantics* details which structure the system implied by the description has, i.e. which objects and which properties it has. Each system state is then characterized by this same structure.

There will be different realizations of these system structures, depending on the underlying reality that is used to run the system. For example, there will be

[7] The DELTA language report used the neutral term 'generator' that generates a system based upon a system description, i.e. provides the vertical relation. A generator could be a machine or a human being, or a mixture. In MDA, a generator is most often understood as a tool generating a new low-level description out of the original high-level description. This would amount to a horizontal generation and is not what semantics is about here.

different ways to represent Java objects depending on the underlying machine, but on the abstraction level of the execution, these are the same. This is given by the abstraction property of system as discussed in Figure 4. In a similar sense, a musical sheet will have different realizations depending on the musical instrument it is played on.

In our object-oriented discourse, we can observe the system behaviour using *snapshots*, i.e. collections of objects with the values of their properties. Such a snapshot is a complete description of the current state of the execution of the system as defined by the execution semantics[8]. A snapshot includes information about all relevant runtime objects.

Experiments with systems, like testing and simulations, are experiments with an execution, not with descriptions. In [Exner et al., 2014], it is well documented that even prototyping is experimenting with systems in varying degrees of completeness, not experimenting with descriptions.

4 Models

After discussing systems and (system) descriptions, we are prepared to look into the definition of models. The UML standard uses the following definition [OMG, 2017]. Note the distinction between existing and planned systems (parts).

> A model is always a model of something. The thing being modeled can generically be considered as a system within some domain of discourse. The model then makes some statements of interest about that system, abstracting from all the details of the system that could possibly be described, from a certain point of view and for a certain purpose. For an existing system, the model may represent an analysis of the properties and behavior of the system. For a planned system, the model may represent a specification of how the system is to be constructed and behave.

4.1 The Model-of Relation

We start with a condensed definition from Webster, Collins, Wikipedia, UML, [Bossel, 2013], and a general understanding of model.

> A model refers to a small or large, abstract or actual, representation of a planned or existing entity or system from a particular viewpoint and with a specific purpose.

Observe that this definition of model always defines a relation between the system acting as the model, and the system being modeled, see Figure 8.

Keep in mind that a model is related to some other entity or system, and that this means that being a model is always the same as being in a relation with something we call referent system. This relation is usually called *model-of*.

[8] A debugger is a tool that can show the current state of execution in some notation.

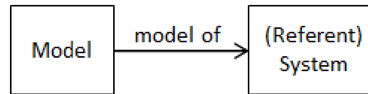

Fig. 8. Models are in a model-of relation with their referent systems.

Furthermore, a model is something that can represent the referent system, as stated in [Podnieks, 2010]: "a model is anything that is (or could be) used, for some purpose, in place of something else." Given the terms introduced so far, a model is a system itself, which means it can be executed in some sense. This leads to our definition as follows.

Definition 5 (model). *A model is a system, that is in a model-of relationship to a referent system, existing or planned. A model resembles the structure and the behavior of its referent system by a model-of relation. A model might be created using a model description (e.g. a diagram, a formula, or code).*

This way, a model is a special kind of system, which implies that a model description is a special kind of system description. The model description is used for creating the model, but it is not the model itself, see Figure 9, which combines

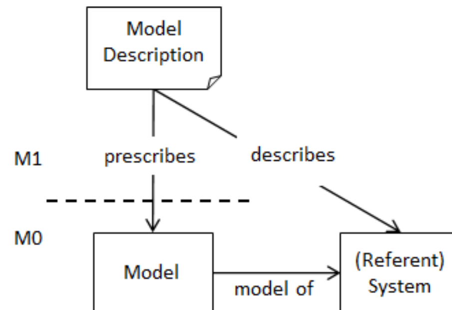

Fig. 9. Model, Description, and Referent System

Figure 6 connecting description and system (here with the special case of a model), and Figure 8 connecting model and referent system. The combination yields a combined relation between the description and the referent system. This relation is most often called a describe relation. It is a bit weaker than the prescribe relation, but it is also relating a description to a system.

A system description always leads to a system (a set of possible executions). The system does not need to be a model if there is no related referent system.

Note that both physical and mathematical models are systems, because it is their behaviour (their executions) that makes them models. Scale models are also concrete representations, but typically with an object structure that does not change over time.

Let us look at the small example in Figure 10 to see what this definition implies. There are specifications (descriptions) of books in UML and in Java

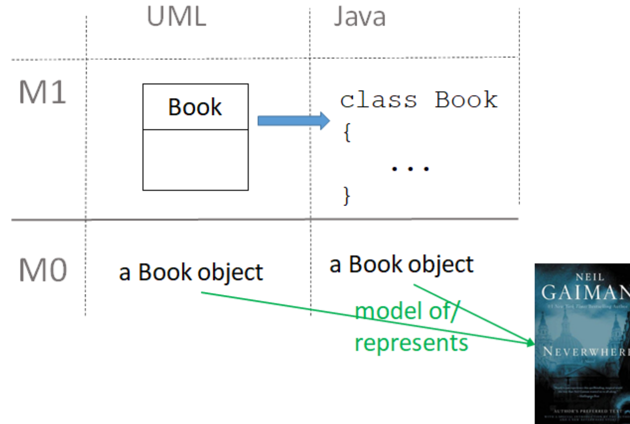

Fig. 10. Models of Books

for a library system. The Java description might be derived from the UML description (blue arrow in the figure), but this connection is not important at the moment. The UML class allows a book object to be created at runtime, in the same way as the Java class allows a book object to be created at runtime. These runtime objects contribute to form a system, i.e. a library system. These two objects are typically models of the real book object existing somewhere in a library.

Libraries are systems with changing sets of books and loans. Models of libraries with the purpose to understand libraries or to make computer-based library systems must be systems of objects representing real books and loans. The model, in this case, is an actual representation.

Now that we know what models are, we can define modelling as follows.

Definition 6 (modelling). *Modelling is the activity to create a model based on a purpose. There are two ways to create a model.*

1. *Create the model directly as a system (scale model or physical model).*
2. *Create a description that implies a system which is the model (mathematical, computer, design models).*

The Scandinavian approach fits well with the process of modeling technical or environmental systems. Authors like [Schmidt, 1987], [Pritsker, 1979] and [Hill, 2002] identify three iterative phases of this process. The first phase establishes a model problem resulting from the system purpose, and a model is derived, the validity of which has to be verified experimentally for the purpose of the investigation. The model is described (programmed) in the second phase

as an executable simulation model. Instead of experimenting with the original, we are now experimenting with the (executable) model. The third phase is dedicated to targeted experimentation. A distinction is made between experiments to prove the model validity of the simulator and experiments to solve the model problem. The phase is concluded by the intellectual transfer of the model results into the world of the original.

4.2 Correctness

The model-of relation allows us to discuss correctness of models, see Figure 11. An example for the figure are interactions and use cases of UML, which provide

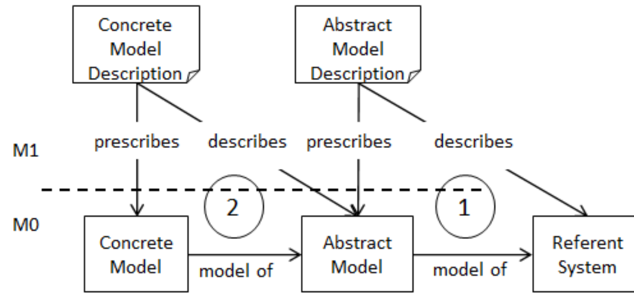

Fig. 11. Correctness

some kind of abstract system description by giving a partial formalization of ideas about the system as shown to the right of Figure 11. Typically, they need to be extended with more formality, for example class diagrams or code, as shown to the left of Figure 11.

The right part of Figure 11 named ① shows the same situation as Figure 9. In this part, the relation between abstract model and referent system decides whether the model is correct. This is not a formal exercise as the referent system does not exist in a formal way. Validation is the process of finding out whether a system has the right model-of relation to an existing or planned real system, i.e. whether their executions match. In order to prove that the model system can represent the original system for the desired model purpose, i.e. that the model-of relation can be recognized as valid, according to [Bossel, 2013], the validity must be proven with regard to four different aspects: behavioral validity[9], structural

[9] It must be shown that for the initial conditions and environmental effects of the original system that are fixed within the scope of the model, the model system produces the (qualitatively) same dynamic behavior.

validity[10], empirical validity[11], and applicability[12]. Validation by prototyping is the process of finding the system that is the desired model of a planned system.

As the match between model and referent system is already part of the definition of model, an incorrect model would not be a model at all. If the model is able to reflect the behaviour of the referent system under certain conditions, it might still be a model for a restricted purpose. Correctness is very important for the existing parts of the system. They start with a physical system and provide the model of it later. For the new parts, the referent system is imagined and the model is the first physically existing system.

In the diagram part named ②, we see again a similar diagram as in Figure 9, where the referent system is given by the abstract model. Therefore, in addition to the description for the concrete model, there is also a description for the abstract model in the role of the referent system. This means we can formally compare the semantic implications of the two descriptions and determine whether the two systems match. This approach, called verification, uses the semantics of the two descriptions to compare the syntactic structures without going into the running systems. This way, the question of correctness is lifted to the level M1.

In a system with existing parts, we can simulate the system using the models of the existing parts. This way, we can *experiment* with the models instead of with the real systems and deduce properties. This is the typical development phase, where we use a model for the existing parts and the controller is changing rapidly due to better understanding of the system. Once the controller works well in the simulated environment, we can move to the real environment. The description is still the same, but now the existing parts are exchanged with their referent systems, i.e. the existing parts themselves. This means we do now deploy the new parts onto the real referent system.

5 Relation to the Kiel Notion of Model

In this section, we relate the concepts introduced in the previous sections to the Kiel notion of models [Thalheim et al., 2015]. The Kiel notion of model is a very general notion capturing models from all areas of science. The Scandinavian approach is geared towards dynamic models in computer science. Still, as our approach is quite general as well, the two approaches can be compared. We start with an introduction to the Kiel model, then we summarize our approach, and finally we compare the two.

[10] It must be shown that the effect structure of the model (for the model purpose) corresponds to the essential effect structure of the original.

[11] It must be shown that in the area of the model purpose, the numerical results of the model system correspond to the empirical results of the Originals under the same conditions, or that they are consistent and plausible if there are no observations.

[12] It must be shown that the model and simulation options correspond to the model purpose and the requirements of the user.

5.1 The Kiel Notion of Model

The Kiel notion defines a model as follows.

Definition 7 (Kiel Model [Thalheim et al., 2015]). *A model is a well-formed, adequate, and dependable instrument that represents origins. Its criteria of well-formedness, adequacy, and dependability must be commonly accepted by its community of practice within some context and correspond to the functions that a model fulfills in utilisation scenarios and use spectra. As an instrument, a model is grounded in its community's subdiscipline and is based on elements chosen from the sub-discipline.*

In addition, [Thalheim et al., 2015] and [Thalheim and Nissen, 2015] as well as [Thalheim, 2019] give more detail to the criteria as follows.

- A model combines an intrinsic deep model and an extrinsic normal model. The deep model is based on the community's subdiscipline and has its background, e.g. paradigms, assumptions, postulates, language, thought community. Models are typically only partially developed as normal models which properly reflect the chosen collection of origins.
- An instrument is a device that requires skill for proper use in some given scenario. As such it is (i1) a means whereby something is achieved, performed, or furthered, (i2) one used by another as a means or aid, (i3) one designed for precision work, and (i4) the means whereby some act is accomplished for achieving an effect. An instrument can be used in several functions in scenarios.
- The criteria for well-formedness, adequacy, and dependability depend on the function that an instrument plays in the given scenario. Due to the function a model plays, it has a purpose and satisfies a goal. The (p) profile of a model consists of its functions, purposes and goals. A well-formed instrument is adequate for a collection of origins if (a1) it is analogous to the origins to be represented according to some analogy criterion, (a2) it is more focused (e.g. simpler, truncated, more abstract or reduced) than the origins being modelled, and (a3) it sufficient to satisfy its purpose. It is dependable if it is justified and of sufficient quality. Justification can be provided (j1) by empirical corroboration according to its objectives, supported by some argument calculus, (j2) by rational coherence and conformity explicitly stated through formulas, (j3) by falsifiability that can be given by an abductive or inductive logic, and (j4) by stability and plasticity explicitly given through formulas. The instrument is sufficient by (q1) quality characteristics for internal quality, external quality and quality in use. Sufficiency is typically combined with (q2) some assurance evaluation (tolerance, modality, confidence, and restrictions).
- The background consists of (g) an undisputable grounding from one side (paradigms, postulates, restrictions, theories, culture, foundations, conventions, authorities) and of (b) a disputable and adjustable basis from other side (assumptions, concepts, practices, language as carrier, thought community and thought style, methodology, pattern, routines, commonsense).

– A model is used in a context such as discipline, a time, an infrastructure, and an application.

Models function in scenarios for which they are build. The intrinsic deep model mainly depends on its setting: the function that a model plays in given scenarios, the context, the community of practice, and the background. Scenarios often often stereotyped and follow conventions, customs, exertions, habits. The scenario determines which instruments can be properly used, which usage pattern or styles can be applied, and which quality characteristics are necessary for the instruments used in those activities.

Therefore, we may assume that the deep model underpins any model within the same setting. As long as we only consider models within a given setting [Thalheim, 2017] we may use simpler notions of model, as given in [Wenzel, 2000] as follows.

> A model is a simplified reproduction of a planned or real existing system with its processes on the basis of a notational and concrete concept space. According to the represented purpose-governed relevant properties, it deviates from its origin only due to the tolerance frame for the purpose.

This definition already assumes the system background for simulation scenario in the context of production and logistics.

5.2 The Scandinavian Approach to Modelling

As a comparison, we have a look at the Scandinavian approach, see Figure 12. The Scandinavian approach has the following properties.

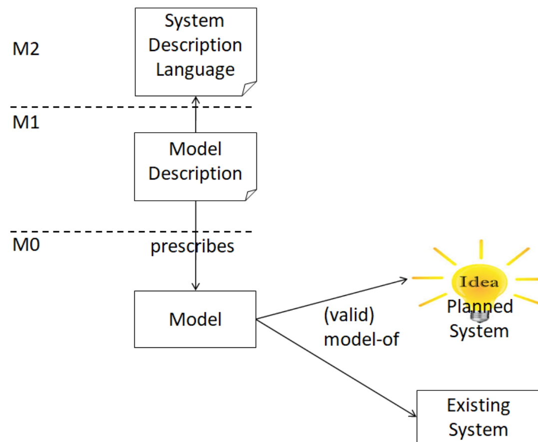

Fig. 12. Scandinavian Model Approach

- A system has a purpose, determining the properties of the system, with other properties 'abstracted away'.
- A system is abstracted from the reality in terms of a discourse.
- A model is a special kind of system.
- A model is in a model-of relationship with another system.
- A model can be created from a model description, which is a special case of a system description.
- A model description adheres to the system description language in which it is made.

5.3 Comparing the Approaches

Now we relate the elements of the two definitions, see Figure 13.

Scandinavian Approach	Kiel Notion
A system has a purpose.	profile
A system purpose provides focus on essential system properties.	(a2)
A system is abstract in terms of a discourse.	(b), (g)
A model is a system.	specific instrument
A model is in a model-of relationship with another system.	origin-relationship, (a1), (a3), (g), (b)
A model can be created from a model description.	construction scenario
A model description adheres to the language it is written in.	well-formed

Fig. 13. Comparing the two approaches

Most of the connections are obvious, but some comments are in place.

1. The Scandinavian approach distinguishes between model description and model system, which is not explicitly done in the Kiel notion where a model may consist of several tightly associated models, i.e. a model suite. A bi-model suite may consist of a model and its representation or informative model. The latter is essentially a model description. This leads to the well-formedness being related to the model in Kiel and to the model description in the Scandinavian approach.
2. The Kiel notion allows a description to be a model, which is not the case in the Scandinavian approach. The connection between description and model is further discussed in the next section.
3. The Kiel notion has much focus on purpose, usefulness, and the details of the model-of relation. This is has not been the focus of this paper, even though the Scandinavian approach has some ideas about it. Therefore, some aspects

of Kiel do not appear in the table and many are grouped into the purpose and the model-of.

4. The Kiel notion is a generalisation of modelling practices in many scientific and engineering disciplines. It can be adapted to the Scandinavian notion by using specific adequacy and dependability criteria within a system construction setting.

As for the last point in the list above, the Scandinavian approach is based, not upon many scientific and engineering disciplines, but an understanding of application domains (or reality in general) consisting of phenomena (with measurable properties and behavior that changes these properties) and concepts classifying these phenomena. This is well-known from outside computer science, and when applied to modeling and programming, the short version is that objects model phenomena in the application domain, classes model concepts, and subclasses model specialized concepts. Composite phenomena are modeled by objects having part objects, the special case being the system containing parts. Objects have their own behaviour (not just methods), reflecting that in some domains there are phenomena (e.g. processes in a process control domain) that are active and exhibit parallel behaviour. The two approaches agree that analogy is a semantic property of the systems (executions). Analogy means in both approaches that elements of a model system represent (model) the corresponding elements in the referent system (origin). It is far more general than the mapping property that is often required for the model-of relation.

The idea of system and the idea of instrument do not match completely. This article defines system, but the notion of instrument is not defined in [Thalheim et al., 2015]. Apart from being a system, an instrument (or tool) also has some property of being useful, as stated in (g). Although system and instrument may not match, the Scandinavian notion of model matches the Kiel notion of instrument in the sense that both are used for the purpose of finding out more about the system to be modeled. Even the model description as an instrument does not collide with the Scandinavian approach: here the model description plays an important role in communication about the model. The name DELTA means 'participate' in Norwegian, and the idea behind the language was that the users of the final system should be able to participate in the 'system description' (the term for model description at that time) prior to its implementation. This later lead to the idea of 'participatory design', but now other means are used, like mock-ups, prototyping. However, a description (e.g a program) still is an important instrument for developers.

6 Discussion

As explained before, the Scandinavian approach makes a clear distinction between a model and its description. This implies that a description is not a model.

We look into modelling and programming first, where we also discuss if programmers model. Then we consider code generation in the context of models and discuss why a description is not a model.

It might seem that the terms introduced and the details considered are not important in general. This is true, but still it is important to have the correct basic understanding in order to sort all the cases that might appear in programming and modelling practice.

6.1 Programming versus Modelling

The Scandinavian notion of model applies to modeling and programming in general and would be the starting point of a combined modeling and programming language as proposed by several authors ([Madsen and Møller-Pedersen, 2010], [Seidewitz, 2016], [Broy et al., 2017], [Cleaveland, 2018]).

Markus Völter has compared programming and modelling in [Völter, 2011] and [Voelter, 2018]. He uses a definition of model-driven as follows.

> Model-driven refers to a way of developing a software system S where users change a set of prescriptive models M_i representing concerns C_i at an appropriate abstraction level in order to affect the behavior of S. An automatic process constructs a full implementation of S that relies on a platform P.

In this context, his conclusion is that modelling and programming are the same, and also coincide with scripting.

> Programming and modeling, in the sense of model-driven, where models are automatically transformed into the real system, cannot be categorically distinguished. However, the two have traditionally emphasized different aspects differently, making each suitable for different use cases.

He uses a similar definition of programming as we do, but he considers modelling to be creating a system description. This is of course the same as programming. However, a model description is not only a description of a system, but the system has to be related to a referent system using the model-of relation. Then all modelling by creating descriptions is also programming, but not all programming is modelling.

6.2 Do Programmers Model?

The question could be asked if programmers model? Of course, no programmer would want to write useless descriptions, so this kind of modelling they would avoid.

However, programmers *do* model, in the sense that they create systems that relate to reality, i.e. have a model-of relationship. They identify domain concepts and represent appropriate classes in the programs in line with the purpose of the system being developed. Objects of these classes are then model-of the corresponding phenomena. This is already discussed in Figure 10.

This kind of modelling is not alien to programmers and can help programmers use modelling. Of course, a system might also contain platform- or implementation specific elements that are not models of anything.

In fact, on a more basic level, each programmer is following an idea of what the program is supposed to do and tries to write a description of a system that does the same. This is in the very core of modelling.

6.3 Code Generation and Models

Let us look into code generation, either manually or automatically, see Figure 14. What is the relation between the high-level code (UML specification) and the

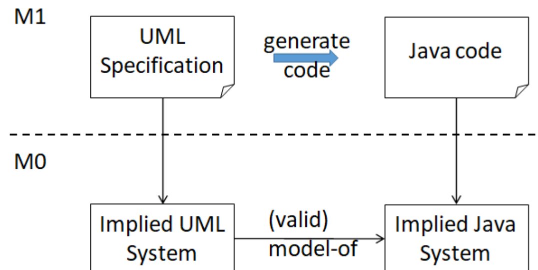

Fig. 14. Code generation and model-of

low-level code (Java)? It is often claimed that the high-level code is a model of the low-level code. The same argument as before applies, in that the code itself is not a system. The connection between the two codes is given by the semantics of each of them. They both imply a system each via their respective semantics, and these systems can be related via the model-of.

This indirect connection between the two kinds of code can be used to allow automatic code generation from the higher to the lower level. Please note that there might be different ways to create code from the higher level, and all of them can be correct as long as there is a match between the implied systems, i.e. they are semantically correct as discussed in Figure 11.

6.4 Why is a Description not a Model?

According to the arguments given before, a description is not a model, but implies a system which can be a model-of reality. But maybe it is possible to have also descriptions that are models?

Let's look at Figure 15. It is often claimed that the UML specification or the database specification is the model of the system produced later on. This is not completely wrong, but the relation between the UML specification and the referent system is indirectly composed of two relations as shown in Figure 15. There is a relation from the specification to the implied system, which is given by the semantics. The result of the semantics is then in the model-of relation to the system that is created. We have seen this combination already in the define

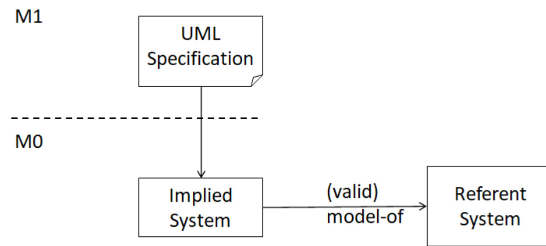

Fig. 15. Description to Model

relation, see Figure 9. So the specification (which is a description) is not the model, but the implied system is the model of the referent system. This is in contrast to the following text from [OMG, 2017] on what a model is.

> A Model is a description of a system, where 'system' is meant in the broadest sense and may include not only software and hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given level of abstraction and viewpoint) are represented in the Model.

It is important to be precise that the description is not the model itself, but implies a system which is the model. This is also true for all model descriptions of database systems: A relationship diagram is not the model of the database, but it implies a system (mathematically) that is the model. In fact, the relationship diagram is then translated to code which again is a description. Running the code provides the system that is the model.

A similar situation relates to a model of the Mini (left in Figure 16). Two alternatives, as presented by [Madsen and Møller-Pedersen, 2018], are shown in Figure 16. The same question arises: Is the description the model of the mini,

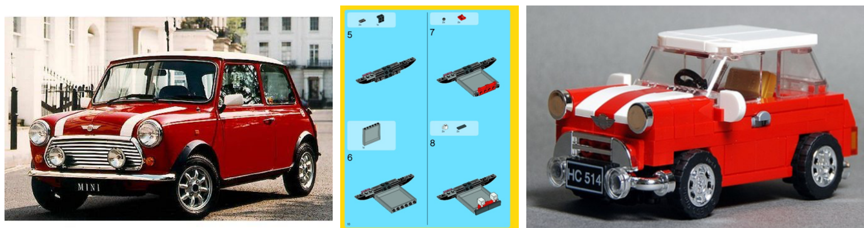

Fig. 16. Original, description, and model

or the implied result of the construction, i.e. the small Lego Mini? The answer is obvious - it is the small Mini - the system, not the description.

The connection between model system and real system is obvious when we look at experiments with the model. The Lego car allows to move forward and backward, and to turn. The situation is different with the description, which only helps to generate the Lego car.

Another example was presented in an invited talk by James Gosling, Oslo, 2017. Sea robots were developed by testing them out in a simulation of the sea with waves, currents and obstacles. The simulation, that is the program execution (system) with the sea robots as objects, is the model of how it will be for real. It is not the simulation program and the programs of the sea robots that are models of the sea and of the real sea robots. The example also shows that the programs of the sea robots became part of the real sea robots, i.e. development by help of simulation. Note that these sea robots were simply programmed in Java without using a modeling language. Still, the simulation is a model of the sea with robots, and the sea robot objects are models of the real sea robots. Our definition of model as a system that is a model of another system is independent of which kind of language is used for making the model system.

7 Summary

This paper has discussed the relationship between models, systems, and descriptions. These terms were then compared with the Kiel notion of model. In this context, an executable model is the key instrument in the communication process about models and system development. It is also key in aligning the Scandinavian approach with the general Kiel notation of model. Other aspects of the Kiel modelling concept world that relate to the purpose, usefulness and details of the model-of relation are only partially discussed and should be deepened in further investigations. As the main result the paper has clarified the differences and similarities between modelling and programming. The paper has concluded as follows.

Systems belong to the modelling level M0. The paper has argued that systems can be real or imagined, but that they exist on their own. Therefore, they are to be placed on OMG level M0.

Executing a system description leads to a system. System descriptions describe systems and lead to systems when their semantics is applied. Here it is irrelevant whether the semantics is formal or not. The description itself is not the system, but leads to it.

A model is a system being model-of a referent system. The model-of relation exists between two systems, which may or may not have a description. The description is the model, it is not involved in the model-of. Instead, the implied system of the description is the model.

Modelling is programming leading to a model. Programming is about writing descriptions for systems. When the system produced is a model of a referent system, then the programming is also modelling.

Two descriptions can describe the same system. When there are two descriptions of the same system, one high-level and one low-level, then both descriptions are related via their semantics, which may describe the same system on different levels of detail. The descriptions are not models of one another, because their syntax does not fit together, but the systems can be models.

With these clarifications it is easy to combine modelling and programming. There are far more aspects of models that are worthwhile to consider, which have been out of scope for this paper.

References

[Bossel, 2007] Bossel, H. (2007). *Systems and Models: Complexity, Dynamics, Evolution, Sustainability.* BBooks on Demand GmbH.

[Bossel, 2013] Bossel, H. (2013). *Modeling and Simulation.* Vieweg+Teubner Verlag.

[Broy et al., 2017] Broy, M., Havelund, K., and Kumar, R. (2017). Towards a unified view of modeling and programming. In *Proceedings of ISoLA 2017.*

[Cleaveland, 2018] Cleaveland, R. (2018). Programming is modeling. In *Proceedings of ISoLA 2018.*

[Dahl and Nygaard, 1965] Dahl, O.-J. and Nygaard, K. (1965). Simula—a language for programming and description of discrete event systems. Technical report, Oslo: Norwegian Computing Center.

[Dahl and Nygaard, 1966] Dahl, O.-J. and Nygaard, K. (1966). Simula: An algol-based simulation language. *Commun. ACM*, 9(9):671–678.

[Exner et al., 2014] Exner, K., Lindowa, K., Buchholz, C., and Stark, R. (2014). Validation of product-service systems – a prototyping approach. In *Proceedings of 6th CIRP Conference on Industrial Product-Service Systems.*

[Fischer et al., 2016] Fischer, J., Møller-Pedersen, B., and Prinz, A. (2016). Modelling of systems for real. In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, pages 427–434.

[Fischer. et al., 2020] Fischer., J., Møller-Pedersen., B., and Prinz., A. (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 - Volume 1: MODELSWARD*, pages 307–318. INSTICC, SciTePress.

[Gjøsæter et al., 2016] Gjøsæter, T., Prinz, A., and Nytun, J. P. (2016). MOF-VM: Instantiation revisited. In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, pages 137–144.

[Hill, 2002] Hill, D. R. C. (2002). Theory of modelling and simulation: Integrating discrete event and continuous complex dynamic systems: Second edition by b. p. zeigler, h. praehofer, t. g. kim, academic press, san diego, ca, 2000. *International Journal of Robust and Nonlinear Control*, 12(1):91–92.

[Holbæk-Hanssen et al., 1973] Holbæk-Hanssen, E., Håndlykken, P., and Nygaard, K. (1973). System description and the delta language. Technical report, Oslo: Norwegian Computing Center.

[ITU-T, 1995] ITU-T (1995). *Basic Reference Model of Open Distributed Processing.* ITU-T X.900 series and ISO/IEC 10746 series. International Organization for Standardization.

[Kleppe and Warmer, 2003] Kleppe, A. and Warmer, J. (2003). *MDA Explained*. Addison–Wesley.

[Madsen and Møller-Pedersen, 2010] Madsen, O. L. and Møller-Pedersen, B. (2010). A unified approach to modeling and programming. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, MODELS'10, pages 1–15, Berlin, Heidelberg. Springer-Verlag.

[Madsen and Møller-Pedersen, 2018] Madsen, O. L. and Møller-Pedersen, B. (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*, pages 206–224.

[Madsen et al., 1993] Madsen, O. L., Møller-Pedersen, B., and Nygaard, K. (1993). *Object-oriented Programming in the BETA Programming Language*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[Nygaard and Dahl, 1978] Nygaard, K. and Dahl, O.-J. (1978). *The Development of the SIMULA Languages*, page 439–480. Association for Computing Machinery, New York, NY, USA.

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

[Podnieks, 2010] Podnieks, K. (2010). Towards a general definition of modeling. available at https://philpapers.org/rec/PODTAG.

[Prinz et al., 2016] Prinz, A., Møller-Pedersen, B., and Fischer, J. (2016). Object-oriented operational semantics. In *Proceedings of SAM 2016, LNCS 9959*, Berlin, Heidelberg. Springer-Verlag.

[Pritsker, 1979] Pritsker, A. A. B. (1979). Compilation of definitions of simulation. *SIMULATION*, 33(2):61–63.

[Scheidgen and Fischer, 2007] Scheidgen, M. and Fischer, J. (2007). *Human Comprehensible and Machine Processable Specifications of Operational Semantics*, pages 157–171. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Schmidt, 1987] Schmidt, B. (1987). What does simulation do? simulation's place in the scientific method of investigation. *Systems Analysis Modelling Simulation*, 4(3):193–211.

[Seidewitz, 2003] Seidewitz, E. (2003). What models mean. *IEEE Software*.

[Seidewitz, 2016] Seidewitz, E. (2016). On a unified view of modeling and programming, position paper. In *Proceedings of ISoLA 2016*.

[Thalheim, 2017] Thalheim, B. (2017). General and specific model notions. In *Proc. ADBIS'17*, LNCS 10509, pages 13–27, Cham. Springer.

[Thalheim, 2019] Thalheim, B. (2019). Conceptual modeling foundations: The notion of a model in conceptual modeling. In *Encyclopedia of Database Systems*. Springer US.

[Thalheim and Nissen, 2015] Thalheim, B. and Nissen, I., editors (2015). *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston.

[Thalheim et al., 2015] Thalheim, B., Nissen, I., Allert, H., Berghammer, R., Blättler, C., Börm, S., Brückner, J.-P., Bruss, G., Burkard, T., Feja, S., Hinz, M., Höher, P., Illenseer, T., Kopp, A., Kretschmer, J., Latif, M., Lattmann, C., Leibrich, J., Mayerle, R., and Wolkenhauer, O. (2015). *Wissenschaft und Kunst der Modellierung (Science and Art of Modelling) - Kieler Zugang zur Definition, Nutzung und Zukunft*. De Gruyter, Berlin, Boston.

[Union, 2011] Union, I. T. (2011). Z.100 series, specification and description language sdl. Technical report, International Telecommunication Union.

[Voelter, 2018] Voelter, M. (2018). Fusing modeling and programming into language-oriented programming. In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification and Validation. Modeling*, pages 309–339, Cham. Springer International Publishing.

[Völter, 2011] Völter, M. (2011). From programming to modeling - and back again. *IEEE Software*, 28:20–25.

[Wenzel, 2000] Wenzel, S. (2000). Referenzmodell für die Simulation in Produktion und Logistik. *ASIM Nachrichten*, 4(3):13–17.