

Modelling Basics

Science is build up of facts, as a house is build up of stones; but an accumulation of facts is no more science than a heap of stones is a house.

Henri Poincaré
Science and Hypothesis, 1905

Before turning to the problems of modelling in mechatronics, which is our immediate domain of interest in this study, we need to set up a descriptive foundation by clarifying the essential notions, as well as to define new relevant terms, *i.e.* to give names to some (new) notions, when appropriate. A name cannot change the appearance or the properties of the named entity, but nevertheless, it has a great impact on people's attitude through associations that can be provoked – especially at the first contact with an entity. Improperly chosen or inadequate names can lead to misunderstandings and even misconceptions. As we shall see later on, names play an important role in communication, integration, standardization and other fields. We also use these names when we refer to the “building blocks of science” – notions, relations, facts, attributes and others. No “scientific house” (*i.e.* theory) can ever be built without such building blocks. For these reasons, special care should be taken when new terms are introduced or existing terms renamed. Due to the attempt to avoid invention of totally new words, the names chosen for some notions may seem strange at first glance, especially if considered out of context.

Now, let us begin with the formal meaning of the terms *model* and *modelling* and then discuss some of their most important attributes.

2.1 Models and Modelling

Let us start with the most frequently used terms and consider their motivation and interrelations.

2.1.1 Definitions

A definition is the enclosing a wilderness of idea within a wall of words.

Samuel Butler
Note-Books

The word *model* is an overloaded term. For example, the Collins Cobuild Dictionary Sinclair *et al.* (1987), specifies fifteen meanings, with three of them – instances 1, 3 and 4 – being mostly relevant for our purposes:

1. *A model of an object is a physical representation that shows what it looks like or how it works. The model is often smaller than the object it represents.*

...an architect's model of a wooden house.

...a working scale model of the whole Bay Area...

I made a model out of paper and glue.

Model is also an adjective.

I had made a model aeroplane.

...a model railway.

2. ...

3. *A model of a system or process is a theoretical description that can help you understand how the system or process works, or how it might work. (TECHNICAL or FORMAL)*

4. *If someone such as a scientist models a system or process, they make an accurate theoretical description of it in order to understand or explain how it works. (TECHNICAL or FORMAL)*

5. ...

Such overloading of the term with different meanings requires a clear initial statement of how we shall understand this term within this study. Let us have a look at some more specialized (*i.e.* not so universal) definitions.

In Stachowiak (1973), any object having the following three main distinctive features is viewed as a model: to be a representation of something, to be a simplification and to be pragmatic (in the German original they are called “Abbildungsmerkmal”, “Verkürzungsmerkmal” and “Pragmatisches Merkmal”, respectively). Actually, the first one seems to be not always required – *cf.* Section 2.1.3.2 below.

Yet another definition is found in Woolfson and Pert (1999):

The essence of the model is that it should be a simplified representation of some real object or physical situation which serves a particular, and perhaps limited, purpose.

Although these two definitions might seem different at first glance, what in the latter definition is expressed as “to serve a particular purpose” is formulated in the former definition as “to be pragmatic”.

In essence, each model is a purpose-dependent representation. According to the purpose of modelling, it might be required that different traits are represented or respectively ignored, therefore, we shall define model as follows:

Definition 2.1: *Model is a purpose-dependent, finite, simplified, but still adequate representation of whatever is modelled, allowing us to abstract from its unimportant properties and details and to concentrate only on the specific and most important traits.*

The respective implementation may use different media or principles (*cf.* Section 2.4.2.3.1 below) and is neither substantial nor pre-defined. When the model is a *representation of the object's traits and their interrelations* by means of pieces of information (or *data*), we shall speak about *informational models* or *data models*. When these pieces of information are *electronically representable values* (numbers), we shall speak – depending on the context – about *software models* or *computer models*.

To better understand the nature of models and modelling, we shall first of all examine how these notions are related, and how they depend on other factors. A somewhat humorous interpretation of what we are concerned with here is sketched in Figure 2.1.

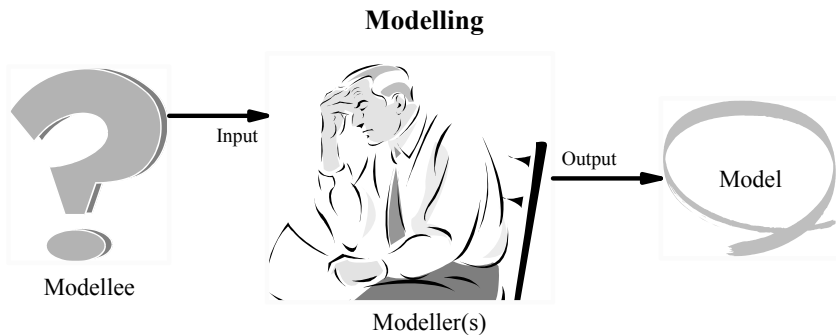


Figure 2.1. Modelling: the “holy” trinity

In a typical representation of a process, the assumption is made that somebody or something acts on something else (input) to create or achieve a result (output). Apparently, such a scenario would not represent the really important fact that processes are time-dependent, *i.e.* they “progress” with time. More precisely, it is the process of creation and development of models that is understood as *modelling*. No unambiguous generic (preferably: one-word) term exists for what occurs as input of the modelling process. Expressions like “object, product or process to-be-modelled”, for instance, would be rather long and imprecise. As we have to refer to the input of the modelling fairly often, let us consider introducing a new term instead. The latter would be used as a generic term in all cases of modelling and especially in the lifecycles of both original and derived products (*cf.* Section 2.2.3). Having looked into the existing terminology as well as having considered the possibilities of making up a new, “artificial” term that would simultaneously be intuitive, short, well-known and, at the same time, not contradictory, I’ve come to

the conclusion that the word *modellee* will be most appropriate for my purposes. This term has the same root as model and modelling and in addition makes allusion to words with similar morphology and well-known meaning like employee (person who is employed), trainee (person who is trained), adoptee (person who is adopted) and many others. So we shall use the term *modellee* for referring to *what* is or will be modelled (the object of modelling).

A conceivable approach to defining the term *model* is to enumerate whatever appears to be related to any thinkable model, and then show the relation between these enumerated “components” or attributes. A graphical representation of such an attempt is given in Figure 2.2. Since for the preparation of this picture we have abstracted from insignificant properties, concentrating only on the specific and important ones, we have eventually created a *model of a model*, or a *meta-model*.

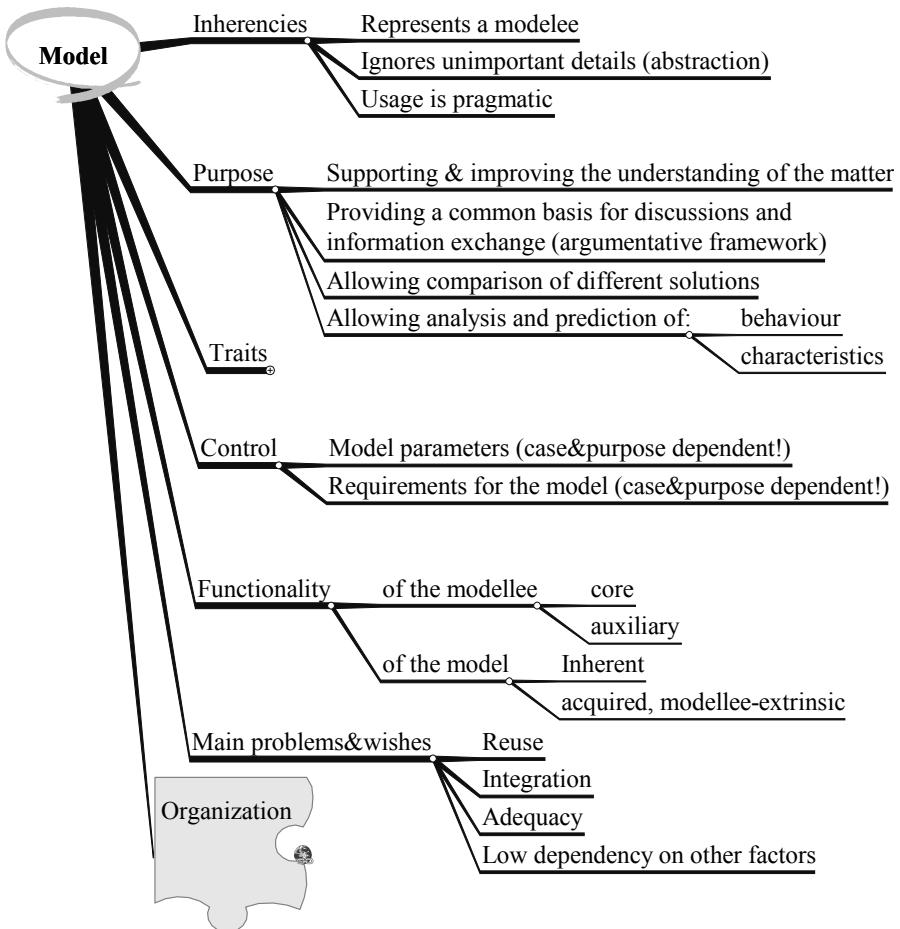


Figure 2.2. Attributes of a model and their relations

Of course, not everybody will accept the representation given in Figure 2.2 without objections. But, depending on the purpose for which this meta-model is

created, different requirements are imposed on its representation, behaviour, level of detail and so on. Inasmuch as these requirements are case-dependent, no model can be perfect *per se* (or in general, or for all cases), but any model can be perfect for a given purpose. And the purpose of the meta-model in Figure 2.2 is to give us an idea what the main attributes of every model are, and how one could start to develop a model. In order to be able to show it in more detail, the organization of a model is presented in a separate picture – Figure 2.3.

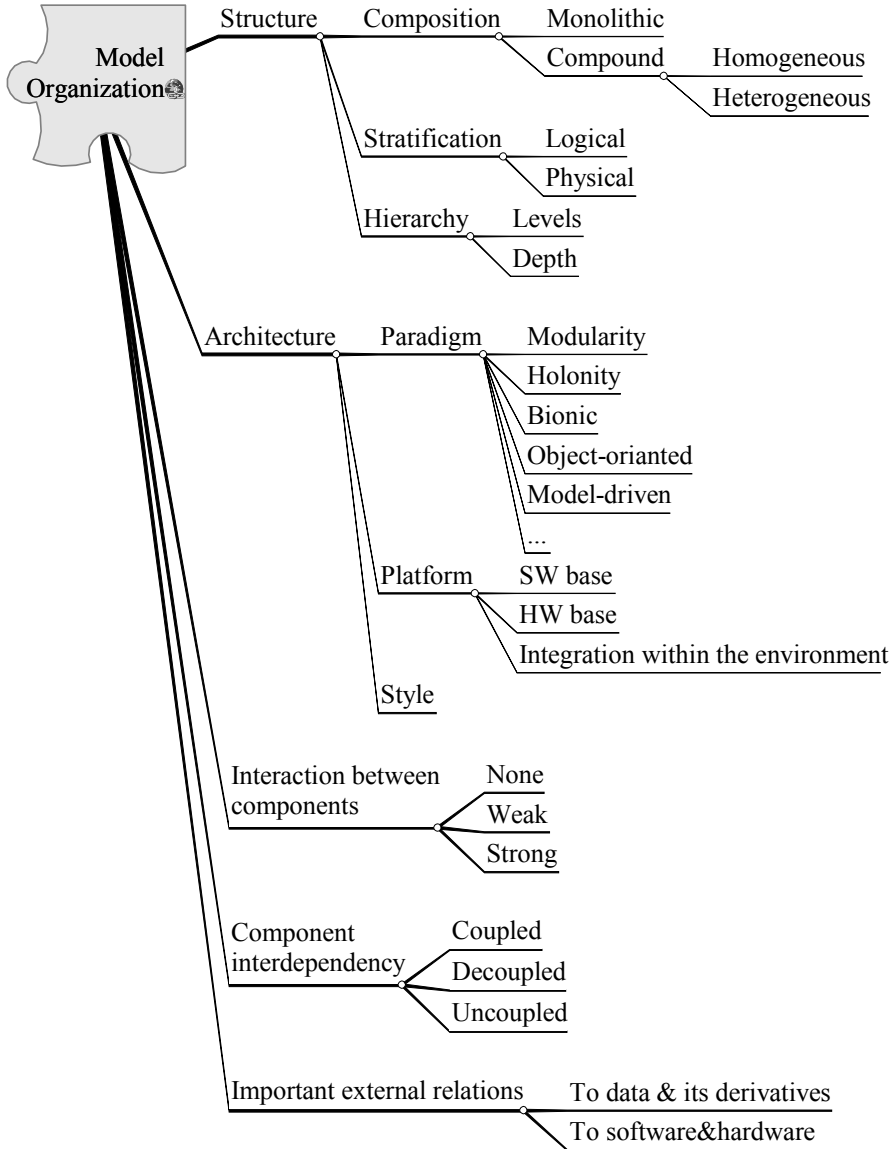


Figure 2.3. Model organization

Another more detailed representation of the most important participants of the modelling process and the relations among them is given in Figure 2.4. Such representation has some similarities with the conceptual graphs as they are defined in Sowa (2000, Appendix A), but the conceptual graphs possess greater expressive power.

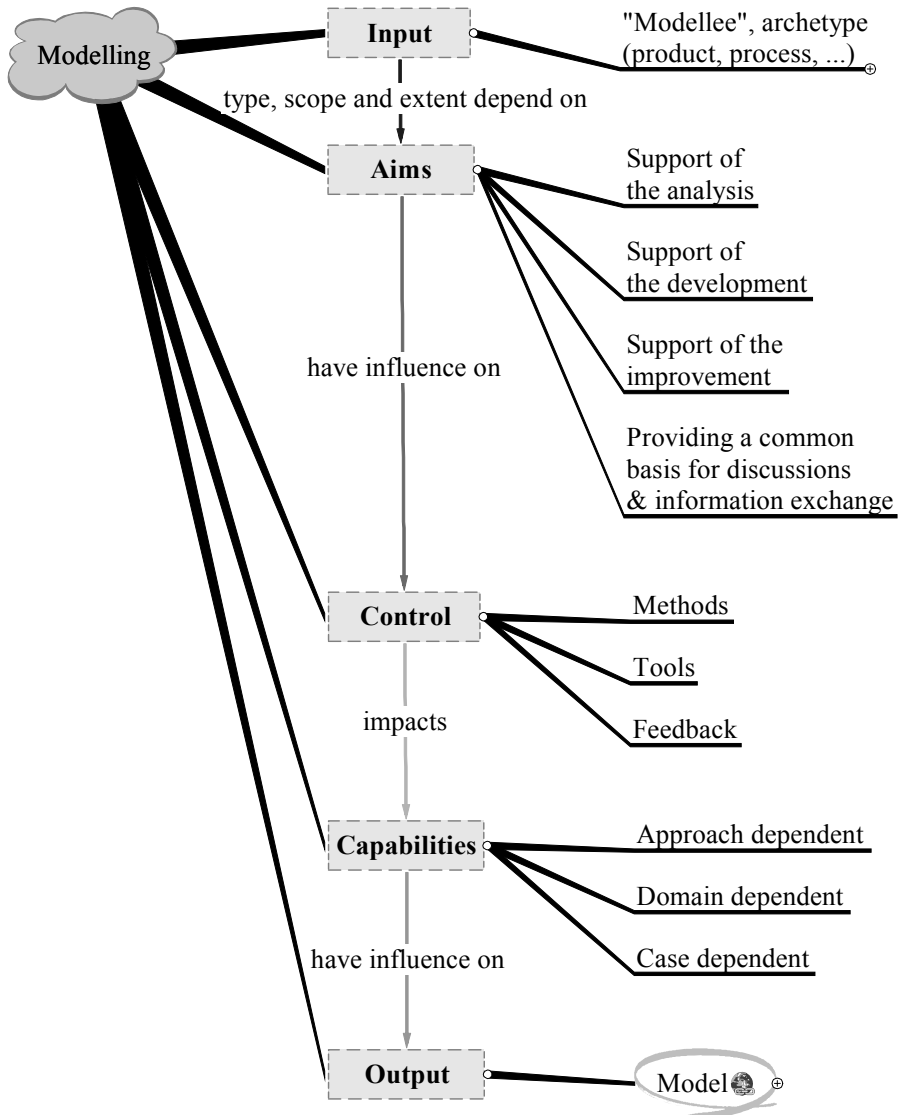


Figure 2.4. Participants in the modelling process and their interrelations

Quite in the spirit of the model of a model given in Figure 2.2, we can prepare a model of the modelling process itself (*cf.* Figure 2.6), which will enable its systematic study. As we can see there, the result of the modelling process is a

(possibly compound) model. If the content of this monograph is brought in relation to the specificities of the modelling process illustrated in Figure 2.6, we can say that it focuses on information-technology aspects and approaches for efficient, platform-independent modelling in the area of mechatronics and mechanical engineering, based on an arbitrary web-browser, a (formal) modelling language, and a 3D visualization engine.

2.1.2 Modelling Stages

Modelling is a complex iterative process and has typically several *phases* or *stages*. Consider the following quotation from Duffy and Andreassen (1995):

Phenomena models are primarily based upon observations and analysis of the “reality” of design and the use of the tools employed, and hence reflect “descriptive” models. Where appropriate, these models are then developed in more detail as information models and similarly as computational models and tools. At each stage any model can be compared or evaluated against any previous model in order to enhance our understanding and hence models.

According to the graphical representation in Figure 2.5 based on Duffy and Andreassen (1995), the nodes in the bottom row of the figure can be viewed as modelling stages, with the reality being the origin of (computer) modelling, and the development leading from a phenomenon model through an information model towards a computer model.

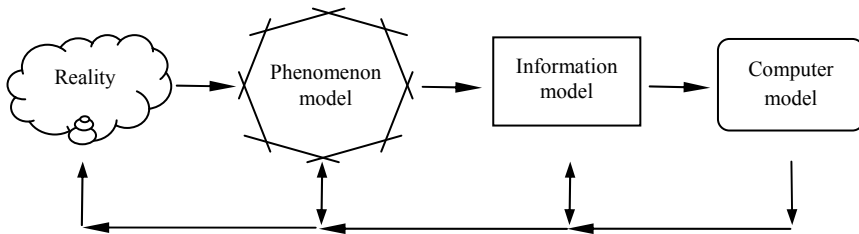


Figure 2.5. Design modelling research approach, after Duffy and Andreassen (1995)

At a careful observation, many (nested) cycles can be discovered in Figure 2.5, and it is not easy to tell where the “beginning” is meant to be: when we have to do with cycles, the cycle-start can be everywhere. Note, however that two main types of activities exist during the development of any model:

1. *Essential (modelling) activities*: improving the model and its adequacy by increasing the number of modelled properties, their accuracy and other essential qualities; and
2. *Auxiliary activities*: “fighting” with the restrictions, limits and problems of the modelling approach used, methods, tools, etc.

Thus, at each new stage some new model quality is achieved, but at the price of increased auxiliary activities. The *efficiency* of the modelling is directly proportional to essential activities and inversely proportional to auxiliary activities.

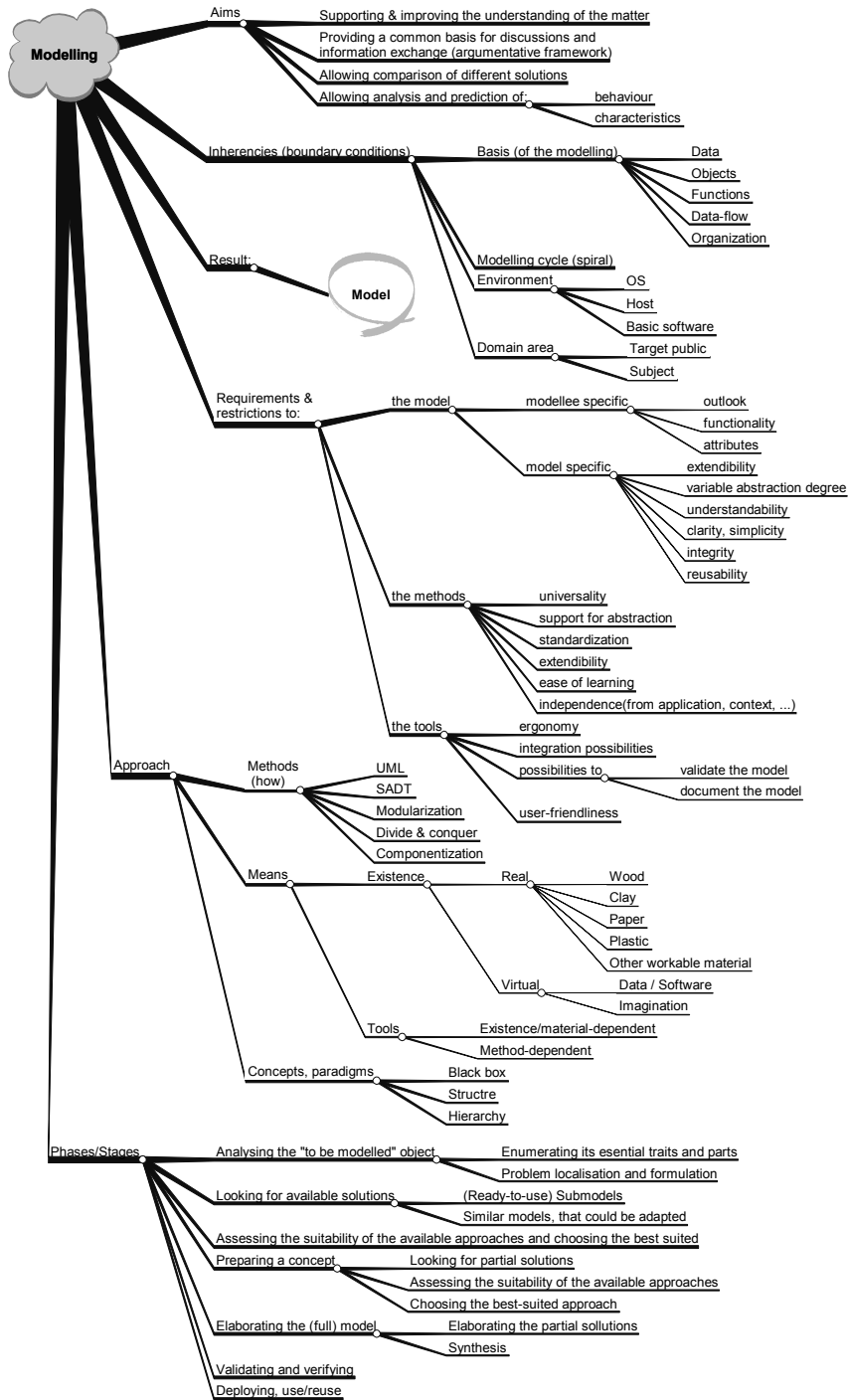


Figure 2.6. Specificities of the modelling process

The stages in Figure 2.5 are defined on the basis of “model metamorphosis” during model development. If, instead, the modeller’s activities were considered, it would certainly be possible during the model development to distinguish phases similar to those presented in Figure 2.7.

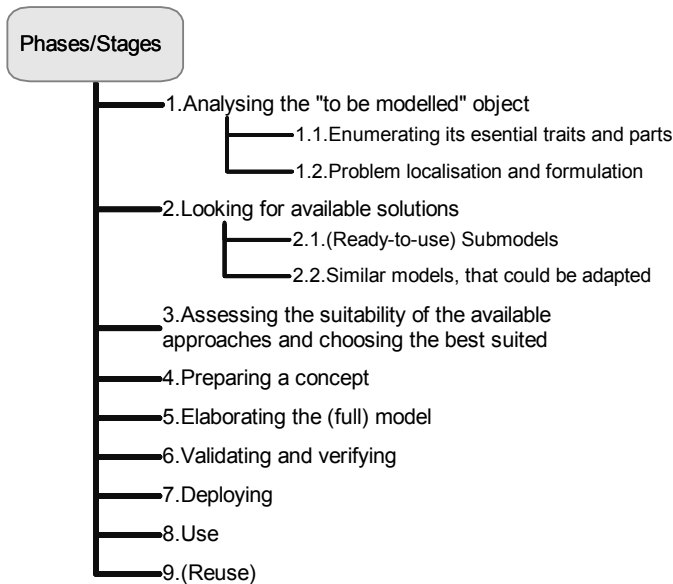


Figure 2.7. Modelling phases and activities

2.1.3 Purpose and Objectives of Modelling

*Ipsa Scientia Potestas Est
(Knowledge is power)*

Francis Bacon, Religious Meditations, Of Heresies

It is in the nature of mankind to strive for power in the most general possible sense – the ability to rule, govern or control every possible thing from the simple toy to the whole universe. For people of different professions this general “rule” sounds somewhat different – the politician strives for political power, the physician would like to have power over all diseases, the fireman wishes to be able to control each fire, the engineers pursue control over the production and so on. But the sense remains in all cases the same: to hold full control over the respective domain. Unfortunately, there are powers that we cannot (yet) control – like the four elements, the sun, the cosmic powers. In cases where uncontrollable power (often referred to as *force majeure*) is involved, the next most attractive and important option is the ability to predict the flow of the upcoming events and the near future. For instance, science and technology are not (yet) strong enough to prevent an earthquake or tsunami-waves, but the foreseeing of their oncoming, together with

the appropriate actions, can avoid almost as many casualties as its prevention and thereby avert calamity.

Let us introduce definitions of the terms *control* and *prediction*, which will be adequate for engineering purposes.

Definition 2.2: To predict means to know the causal connection between some event e_0 and its consequence c_0 . We shall call e_0 a forerunner of c_0 .

Definition 2.3: To (fully) control a given object or system means to be able to put the system in any of its known states, whichever of them is desired.

In many cases Definition 2.3 does not hold, but it is possible to avoid certain undesired states of the system. Such an ability is more important than it seems at first glance. For instance, the ability not to allow a system to reach its worst (or most dangerous) state is more important than the ability to switch this system from the worst state into any other state. We shall call the former ability *blocking* (or *weak*) *control* and define it as follows:

Definition 2.4: Blocking control is the ability to recognize a forerunner and issue a reaction r_0 to it in a way that avoids an (upcoming) undesired event or state of the system, as well as any undesired consequence.

Since, according to this definition, the controller first waits for a given forerunner to occur and then issues a reaction, we shall call this kind of control *passive control*. Of course, in many cases the controller can act on its own – *i.e.* without waiting for any forerunner – in order to change the state of the system. We say then that the controller is *proactive* or exercises *active control* over it. The actions or reactions, issued for gaining or keeping control over a certain system or object are usually called *commands*.

Both passive and active control can be possible either for all forerunners or for some of them, so we can speak about full or partial control.

Definition 2.5: When for a given system and a person controlling it Definition 2.3 holds for any forerunner, this system is fully controllable by the mentioned person.

Cases where a system is fully controllable by somebody are rather rare. Therefore, when we speak about control we typically understand a highly, but still partially controllable system.

The ability to predict, exactly as the ability to rule or control, can exist on any level of scale between the macro-cosmic and the micro-cosmic level. Somewhere in the middle there is also a level that can be viewed as the engineering level, which is in the focus of this work. But is the modelling really related to power and prediction? If yes, how are they related? Well, neither controlling nor prediction are possible without the appropriate *knowledge*. The approximate interdependence of these two abilities on *a priori* knowledge is illustrated in Figure 2.8.

So, the next question is how the knowledge about a given topic or domain is acquired and whether we could influence the acquisition speed. Let us discuss these topics with the help of Figure 2.9. Suppose that the abscissa in Figure 2.9

shows the time or, in some cases – the lifetime of a given domain or entity. Then the solid line, starting from the beginning of the coordinate system shows how our knowledge about this domain or entity changes with the time. Four different phases are denoted under the abscissa, through which the “dealing” with knowledge goes – passive and active learning, and passive and active use of knowledge. In addition, there are two areas surrounded with rectangles, whose line patterns are different. Each of these areas is annotated with text, describing the (specific) activity, which is possible only with an amount of knowledge greater than that corresponding to the knowledge curve at the lower left corner of the respective area.

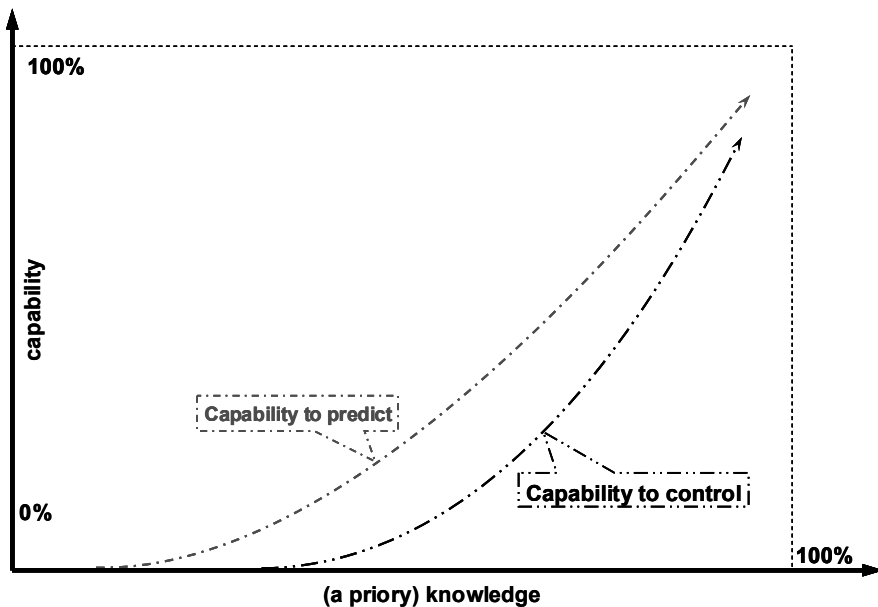


Figure 2.8. Domain-related abilities to predict and control events as a function of the acquired domain-knowledge

During the first phase (passive learning) knowledge is collected relatively slowly, mainly through *perception* – an activity that takes place during all four phases and thus exists throughout the whole lifetime; it is marked with a rectangle, bordered by a dot-pattern line. Perception can occur through any of the human's five senses, but most of it happens through *observation*.

Observation helps us learn everything that can be seen, but since most of the objects are opaque, neither their structure nor the connections among their components/elements can be studied through observation. Imagine you are presented with an unknown to you until now electric torch, having no battery yet. How do you know which position of the switch turns the light on and which turns it off if there is no inscription? One possible way to tell is to put a battery in the torch and to try which position turns the light on, but this is already another activity – an *experiment*.

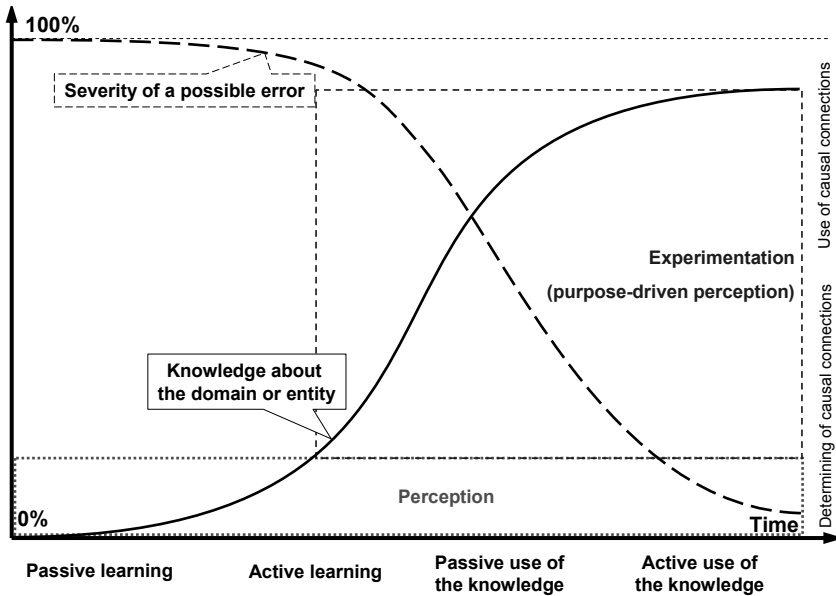


Figure 2.9. Knowledge acquisition and use

Having collected enough *basic knowledge*, an (intelligent) individual is capable of conducting some experiments, observing simultaneously the causal connections between his own *actions* and the subsequent *reactions* of the entity, increasing thus additionally his knowledge. In the case of the electric torch the basic knowledge is that it can light that it works with a battery, and that the light can be turned on and off through the switch. The action would be turning the switch and the reaction – changing the state of the light from on to off or *vice versa*. As soon as you can connect each position of the switch with a state – either lighting or non-lighting – you have learned to use the switch (and the electric torch). Now assume that a friend of yours has been with you and has watched attentively your actions all the time. Would he also have acquired the same knowledge as you did? Well, almost: he would know which position of the switch turns the light on, but he would not know, for instance, how much force is needed for turning the switch. The point is that your friend's newly acquired knowledge is gained through *passive experimenting*, while you gained the same knowledge through *active experimenting*. In general, more knowledge can be gained through active than through passive experimenting.

After sufficient experimenting, the acquired knowledge reaches another point at which the individual is able to *control* the entity to some extent (*cf.* Figure 2.8 again, with the domain being the use of an electric torch), so that some immediate goals can be reached – *e.g.*, turning the light on or off, changing batteries, *etc.* The acquired knowledge typically does not increase during controlling; instead, the *a priori* knowledge (*e.g.*, from old experiments) is confirmed and increases thus the certainty of the controller that the known commands can put the controlled entity to certain states or prevent it from getting into undesired states. Only if something unknown happens – say (if we continue our mental experiment with the above-

mentioned electric torch), the light suddenly does not go on anymore due to a burn-out of the light bulb – this can be registered as a new possible state and thus increases marginally the domain knowledge. The knowledge how to return the system from an unknown state to a known one can be either still missing or generally available – *e.g.*, pressing the reset button of any computer puts it into a well-known state. Thus, the controlling itself can lead to acquiring new knowledge only indirectly – through coming to unknown problems or situations, solvable (only) by means of experimenting. The last proposition raises a fundamental question: is it possible to gain knowledge by mental experimenting? A positive answer to this question would revolutionize the whole science by making many experiments needless. What would be possible for sure is to search the memory for patterns of similar but already solved problems, and then attempt to derive from any pattern found a solution for the current problem/task.

Since in this phase the main activities are observing and experimenting activities together with some controlling, an appropriate name for the phase is *passive use of knowledge*.

From another point of view, as soon as an individual has noticed the causal connection between two (types of) events he can predict what would happen after some known forerunners occur. Back to the example with the electric torch: after using it long enough one should know that a noticeable decrease in the light intensity means an approaching end of the battery's charge, *i.e.* it is possible to *predict* the need for a replacement in the near future, and to take care to have the battery ready at hand. The prediction itself does not increase the knowledge, but as soon as it becomes clear whether the prediction is true or false, the knowledge very often may increase³.

The most important outcome of the phases passive learning, active learning and passive use of knowledge is the determining a causal connection in the given domain. After acquiring a reasonable amount of causal connections (50–80%), it becomes more and more important to use them for acquisition of additional knowledge. We shall call such activity an *active use of knowledge* and name after it the last phase in the learning process. The reasoning and more specifically the use of techniques like induction, reduction and deduction for acquiring new knowledge on the basis of available knowledge and information are typical examples of active use of knowledge.

Let us summarize again the analysis of Figure 2.9:

- Apparently, the activity leading to the highest learning speed is the “experimenting”. Clearly, well planned experiments can additionally increase the learning speed.
- It is never possible to get 100% of the theoretically possible (or practically available) knowledge because it is impossible to learn everything through observation and experiments. Instead, the curve showing knowledge acquisition as a function of time provokes associations with the Pareto-

³ It depends on the kind of prediction, though. If you predict that the next toss up would be a head, you would not know more after the coin falls tail. But if you predict, say, that increasing the pressure in the tyres of your car twice would prolong their life, and yet the first tyre explodes during the increasing, you would know more as a results of this experiment.

principle: in many cases it is possible to acquire about 80% of the knowledge about a certain domain in about 20% of the time.

- The probability of each prediction coming true is proportional to the knowledge about the respective domain.
- It is impossible to fully control anything that is not well-known (or learned). On the other side, only tasks small enough to allow knowing everything about them, can be automated. Assume the control (of a process) can be defined as a mapping of a set of input states – problems – to a set of output states – solutions. Since the automation can be defined as delegating the control or the decision taking to an artefact (device, software, combination thereof, or whatever else), the existence of such a mapping and the possibility of its implementation are crucial. The implementation is only possible if: a) the number of probable input states is countable and exactly known, and b) an onto-mapping M of the set of problems $\{P\}$ to the set of solutions $\{S\}$ is known, and c) M is realizable as an artefact.

So, how could models help here? At least the following two reasons for using models are justified:

- a) Models can be used instead of real resources, at least during the early phases of the development, and thus make even the most intensive experimenting affordable and (financially) more effective.
- b) Models can save time when they are workable or when they allow automation of experimenting. For software models both conditions are fulfilled.

In short, the objective of modelling is to increase learning speed and the amount of acquired knowledge (reason b) and simultaneously decrease the costs of knowledge acquisition (reason a), supporting thereby indirectly the abilities to predict and to control. On this basis are built concepts like Digital Factory, Virtual Factory, or Smart Factory: if anything that has to be build in reality – from a given product up to the factory producing it – is fully modelled, studied and optimized in advance, there is a great potential for saving time, money and other resources.

Of course, there are other reasons why we need models, which are more or less directly related to the discussed abilities to rule and to predict. They will be discussed in the following section.

2.1.3.1 *Why are Models Needed*

There are so many reasons for using models that their complete enumeration and description is almost impossible. Nevertheless, let us try to consider some of the more important ones (*cf.* Figure 2.10).

Models contain or reflect only the most important, for a given purpose, traits of whatever is being modelled. As a result, they reduce the complexity of the modellee and allow the modeller to ignore unimportant traits in order to concentrate on the essentials. Therefore, models crucially support and improve the understanding of the matter. Since the models are a simplified, finite representation of something, they are easier to handle. In many cases the only way for comparison of different objects, products, solutions, *etc.* is to compare their models. For instance, we (still) cannot compare two screws atom-by-atom, particle-by-particle, and this would not make sense either. But it does make sense to compare their diameters, lengths, pitches, number of threads and a couple of other purpose-

dependent traits. Since these traits represent a kind of screw-model, it is enough to compare the models instead of the modellees.

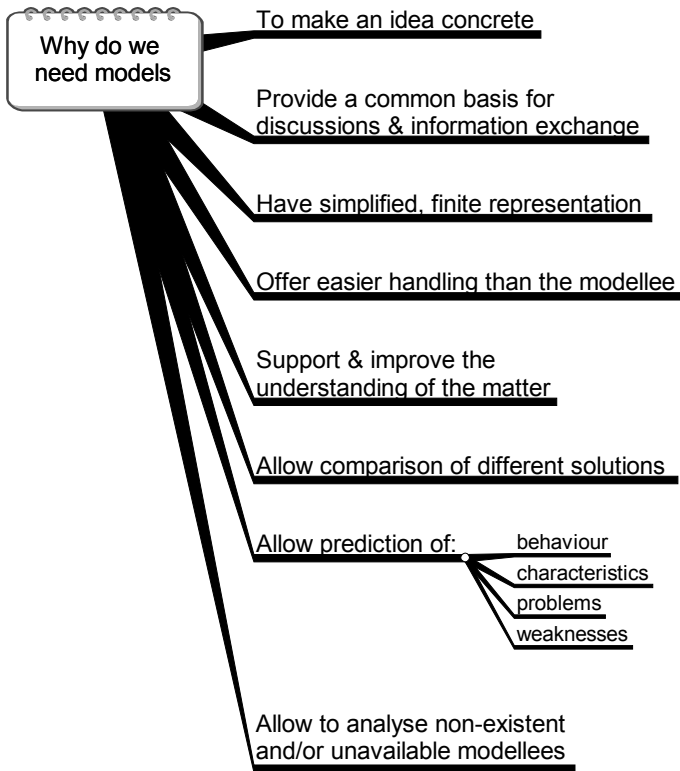


Figure 2.10. Some reasons to use models

Another interesting application of models is the prediction of properties and behaviour. This prediction is based on the comparison of relevant characteristics with those of similar but already known objects, activities, *etc.* For example, whenever we see that a bolt and a nut have the same diameter, pitch and number of threads, it is possible to predict on the basis of previous experience that the bolt will fit into the nut.

A careful look at Figure 2.11 reveals that the model is involved in two loops: gaining insight and applying it to the problem in order to solve it. Besides, the former loop is nested in the latter. The advantage of the “long way” from the real system through problem definition, target definition, model creation, experiment, analysis and so on, viewing the steps clockwise, is that the inner loop (gaining insight) allows one to collect the knowledge, necessary for the solving of the problem, much more quickly (*cf.* Figure 2.9 again!). Note that the more iterations are made in the inner loop the deeper insight would be gained, and the problem should be solved either sooner or better, or both.

Another important reason for using models is to harness them in the problem solving process. A nice example of how this could be done is given in Nyhuis and Wiendahl (2004) and reproduced in Figure 2.11.

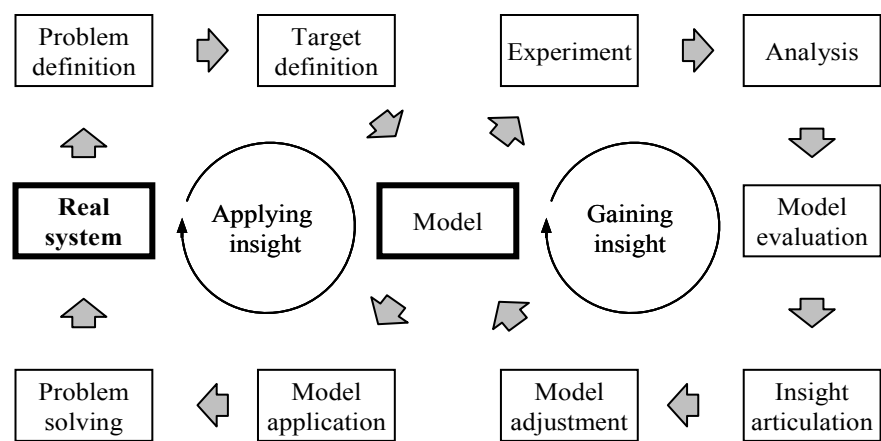


Figure 2.11. Model-based problem solving process, after Nyhuis and Wiendahl (2004)

2.1.3.2 How Models Arise

Some of the possible ways for creating a model are represented in Figure 2.12. In all cases, after a model has been created, it has to be represented in some way in order to make it useful. Without such a representation, the model creator cannot communicate the model to other people, which renders it hardly usable.

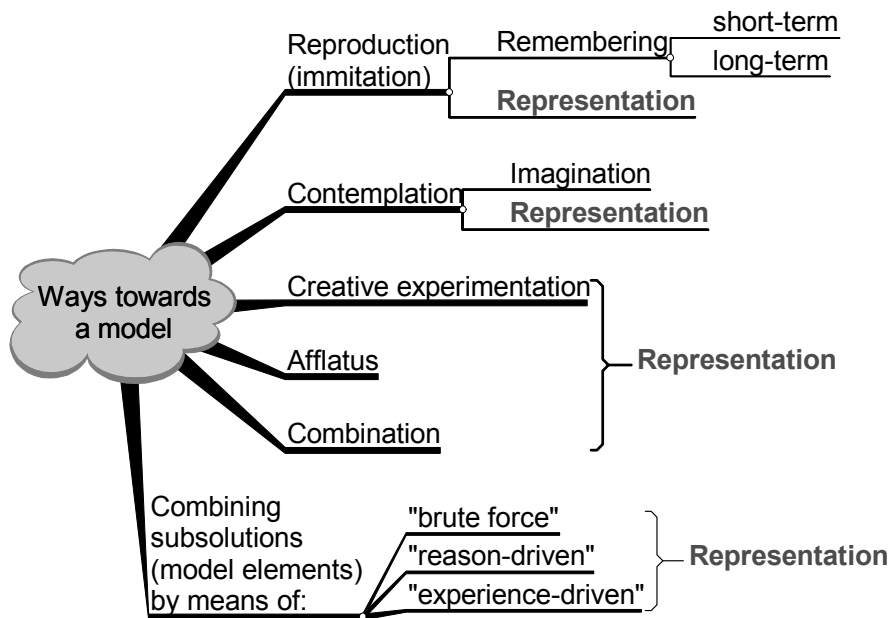


Figure 2.12. Processes leading to model inception

2.1.3.3 Models and Product Development

It seems at first glance that development is a linear or straightforward process, but this is not really the case. The point is that the real flow of this process is difficult to represent. Indeed, attempts to represent development typically concentrate on the most important traits of the process and ignore the non-essential ones. In particular, models are indispensable for the product development, and the main reasons for using models can be summarized as follows:

1. to support the decision taking
2. to shorten the development time
3. to minimize the development costs

All three reasons are more or less interdependent (at least in the direction from reason 1 towards reason 3 – i.e. easier and faster decision taking can shorten the development time, which in turn reduces the costs).

A compact explanation of reason 3 is given in Figure 2.13. The three curves reflect the nearing of the *product's achieved properties* to the *requested properties* of a product as the development advances. The right-hand solid-line curve refers to the normally developed and produced artefacts; the left-hand solid-line curve refers to a rapid prototyping and the dashed-line curve refers to a virtual prototyping – i.e., to the percentage of *modelled product's properties*.

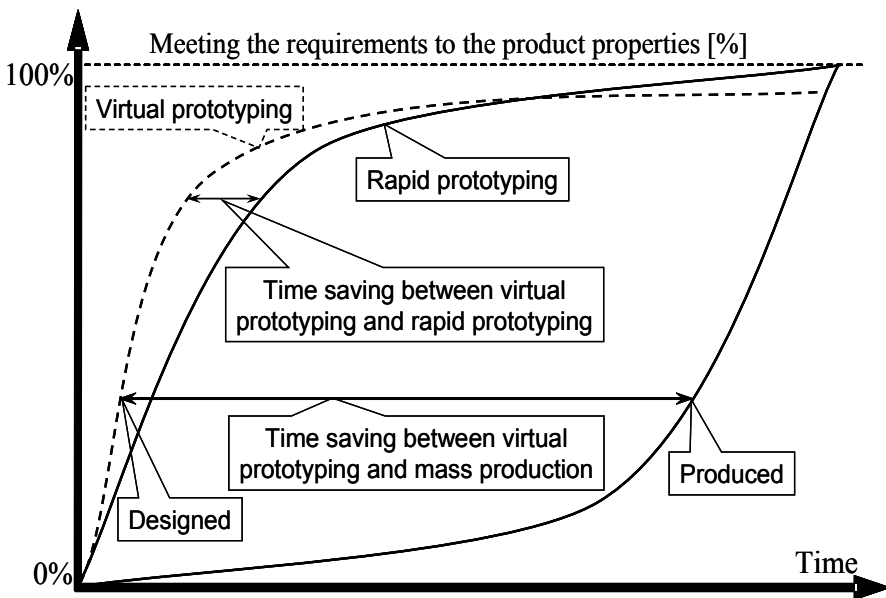


Figure 2.13. Maturity of model and (real) product during their development, after Gausemeier *et al.* (2000)

Why are these curves so different? There are several reasons for this. On the one hand, before an artefact can be produced, the respective production process has to be developed and implemented. On the other hand, for the production of the rapid prototype another technology is used, which leads quickly to a product, but is much more expensive. Due to specificities of the rapid prototyping technology in some cases the manufactured prototype does not have all properties or does not

have the same quality as the real artefact. This applies to the virtual prototypes (or virtual models) in an even stronger way: since they are immaterial, they always lack more properties than a prototype. Since the model is always simpler (by definition) than the modellee, its processing is also easier and much faster. Moreover, since development is a cyclic process, the time difference gained on each loop accumulates. Furthermore, the development of models (and especially software models) requires fewer resources than the development of the product itself or than the manufacturing of its (rapid) prototype, and can be therefore much cheaper. Consequently, it is affordable to make new iterations of the development cycle even with minimal corrections of the model, and the development progress is faster.

Use of models is indispensable also in situations when products or processes are developed for still unknown application areas – e.g. spaceflights.

In short, a careful look around confirms once again that the whole science is based on models. Without models it would be impossible to analyse, to communicate, to compare, to take decisions, to improve, to solve problems and so on. The question about the form of existence (*cf.* Figure 2.2) of a model is of a lower importance, as long as the model achieves its purpose. Accordingly, the capability to control the modelling (of everything and everywhere) can also be crucial for success.

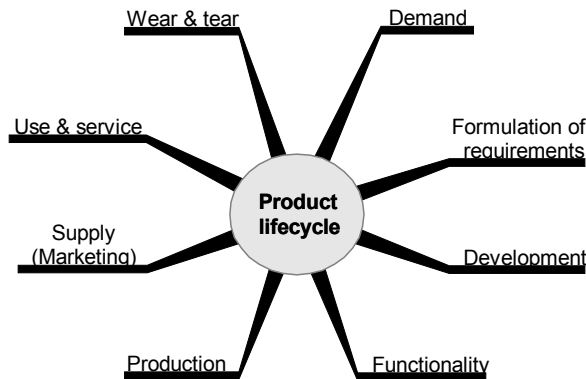


Figure 2.14. Important milestones of the product lifecycle and their sequence

2.1.3.4 Models of Product Development

Since the shortening of the (product) development cycle and the improvement of product quality can be viewed as main purposes of the modelling, it is instructive to discuss this development on the basis of a simplified model. The first question is when the development of a given product starts. Assume that there exists increasing demand for a given product. After this demand reaches a certain threshold (*i.e.* after some delay), a formulation of requirements for the product begins. As soon as the requirements are considered complete, the (actual) development can start. With the development progress, more and more of the functionality of the product is finished, which means that more and more requirements are satisfied. Again, after reaching some threshold of functionality, mass production can start, followed – with respective delays – by marketing and

use of the product. With the start of the product use starts also the wear and tear of its instances, and during use, new requirements are posed. Thus, the loop is closed, and we can speak about the lifecycle of the product with its elements or milestones. A simple model, illustrating the sequence of the mentioned milestones is sketched in Figure 2.14.

Some of these milestones are in antagonistic relations, *e.g.*, demand *vs.* supply, or unsatisfied requirements *vs.* achieved (required) functionality. The latter is more interesting from a technical point of view: as soon as the formulation of requirements is complete, the development starts, and with the fulfilment of each required function or quality the number of unsatisfied requirements decreases until all of them are satisfied. At this point the number of unsatisfied requirements is 0 and the required functionality is 100%, and the first development cycle ends here. At the latest with the beginning of the product use, though, new requirements can arise, which may cause the next loop of the cycle to start. A simplified model of this process is illustrated in Figure 2.15.

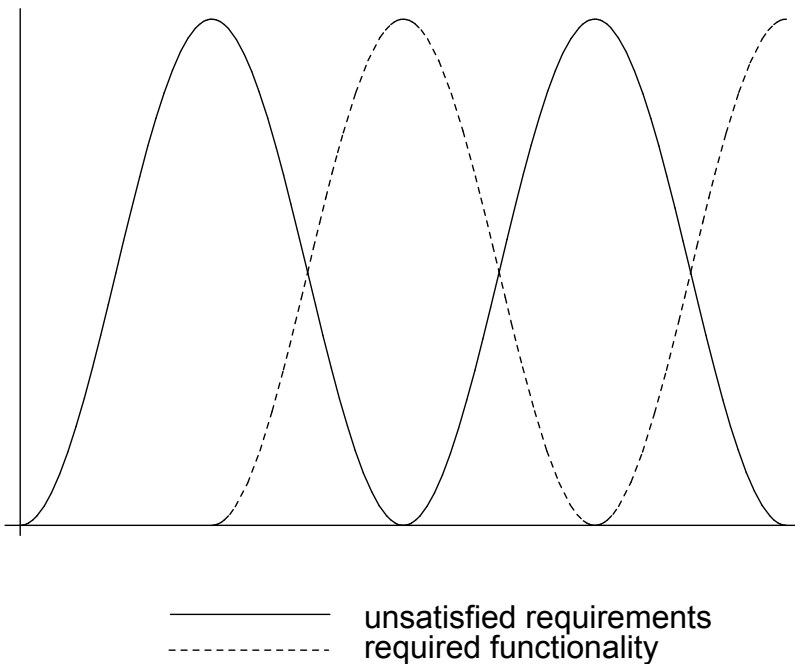


Figure 2.15. Delay between posing requirements and achieving the respective functionality

Typically the number of requirements decreases exponentially with every new loop. This fact can be reflected in the model as illustrated in Figure 2.16.

If we define *model maturity* as the difference between the unsatisfied requirements and (implemented) required functionality, the resulting graphical representation of the curve will be very similar to the upper curve in Figure 2.13.

2.1.4 Some (Unusual) Examples of Models

Let us view some examples and see how they comply with the definitions, and decide which of them are models and which are not. Paradoxically, these samples show that people are modelling all the time, even if they are not aware of it.

2.1.4.1 Text

Strangely enough, the most often used models are *text models*. Let us first consider the single words: they are finite, simple and are “linked to” or describe more or less adequately something. For instance, any verb describes a process or an action; the verb alone is hardly sufficient to achieve an accurate representation, but it represents the most important trait of the action. Most nouns cause association with specific objects or even classes of similar objects – e.g., the noun “lathe” is normally associated with the respective machine tool. When somebody says, “I am running”, we can imagine that he is moving in such a manner that his body periodically has no contact with the ground. Thus, on the one hand it seems that most words can hardly be viewed as models. On the other hand, every word is related to some concept or notion in our minds, which is in turn a simplification or idealization of something and can, therefore, be viewed as the model of this something.

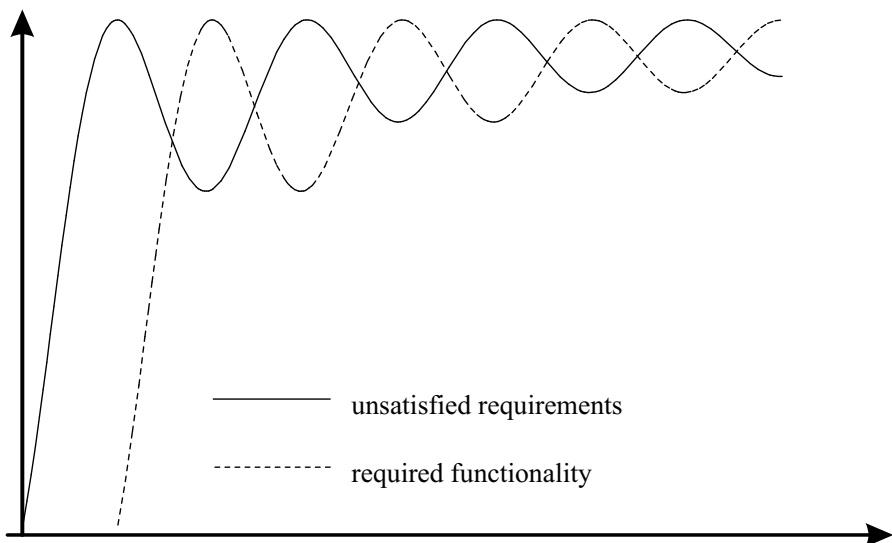


Figure 2.16. Fading in posed requirements and achieving the respective functionality

Now let us consider an arbitrary (textual) description of something. No matter how detailed the description is, it could not describe every atom, every bit, every detail – simply everything – since the description would become infinite. Thus, the author of the description tries to describe the essential things first, then some less important and so on, until there is no reason to describe further (levels of) details. Two criteria are most often applied in deciding when it would make sense to stop, namely:

- a) it is clear that the description of more details would not contribute to the purpose for which the model is created; and
- b) the effort for describing more details would be greater than the achieved benefit.

Thus, the description is a kind of representation of an object, it would become finite due to the outlined reasons, *i.e.* by applying one of the above mentioned criteria or both of them, and it would contain the most important traits of the modellee by ignoring whatever appears to be insignificant. Summing up, words can be viewed as models of notions, texts as models of ideas.

2.1.4.2 Drawings, Sketches and Maps

Similar reasoning is applicable to drawings, sketches and maps: they are finite and represent only traits that are important for a given purpose. Nevertheless, not every sketch is a model: the pen scratches, made by someone unconsciously – for instance, during a phone call – would seldom be named a model of anything.

Maps⁴ of the same landscape or area made in different scales offer us a remarkable example of models with different levels of detail. An indispensable element of almost any map is the legend, where a correspondence between real objects and their representations on the map is defined.

2.1.4.3 Pictures

Pictures in the form of photographs, drawings, scans, *etc.*, are also finite and simplified representations of something. Since they represent the outlook of something, they can be viewed as models of the respective outlook. In many cases, though, the outlook is not the most important property; in such cases pictures can hardly be called models. For instance, a photograph of a book can be a model of the appearance or of the design of a book, but not of the book itself.

It is interesting to contemplate whether a photograph of the content of a book, with still readable text, can be viewed as a model of the book. At first glance, it is a finite representation of the content of the book. At a more careful viewing, it becomes clear that the real model of the book is the text of its content, while the picture is rather a representation of the model of the book and, in a sense, it can be viewed as a meta-model of the book.

2.1.4.4 Bank Statements

A bank statement is a textual document representing the financial transactions of a customer for a period of time (usually month or year). The question is if this statement, which is final and represents important milestones of the customer's financial or bank-related behaviour, could be viewed as a model of this behaviour. The answer can be figured out if the type of information available in such statements (amount transferred, date, source, destination) is compared with the “parameters” of financial behaviour – *i.e.* frequency of transactions; volume of month's and year's turnover; minimal, maximal and average value of transactions, *etc.* It is clear that these types of information are different. Therefore, although a bank statement can be used to organize a model of someone's financial behaviour, it cannot be viewed as its model.

⁴ We mean here geographical, geological and similar types of maps.

2.1.4.5 Itemized Phone Bills

For similar reasons an itemized phone bill is not itself a model of a person's communicational behaviour, but rather could be used for building such a model.

2.1.4.6 Model of a Circle

The geometric figure circle is so well-known that having heard or read this word everybody can immediately imagine its specific shape. Therefore, if we are discussing geometric objects where the respective shape is important, the word “circle” can be viewed as a model of this class of objects, since it bears the most prominent trait of the class – its shape.

In the majority of cases, though, we would like both to use the same model for the whole class and to be able to represent different specific objects (instances) of the same class without having to model them again. To achieve this we need to factor out everything that is specific for the whole class and make some characteristics serve as parameters, allowing us to instantiate arbitrary representatives of the class. Then, in order to be able to distinguish between different objects of the same class we need only to compare the values of the respective parameters.

In the case of a (two-dimensional) circle we would need at least the radius, which can be represented by means of one number. Should we need to distinguish also between circles with equal radii, we could use in addition the coordinates of the circles' centre, which are a pair of numbers for each circle. Consider the hierarchical representation in Figure 2.17. The radius and the centre point are grouped and represented as properties. Further specifications account for model-specific names, types of properties and validity rules (denoted as constraints).

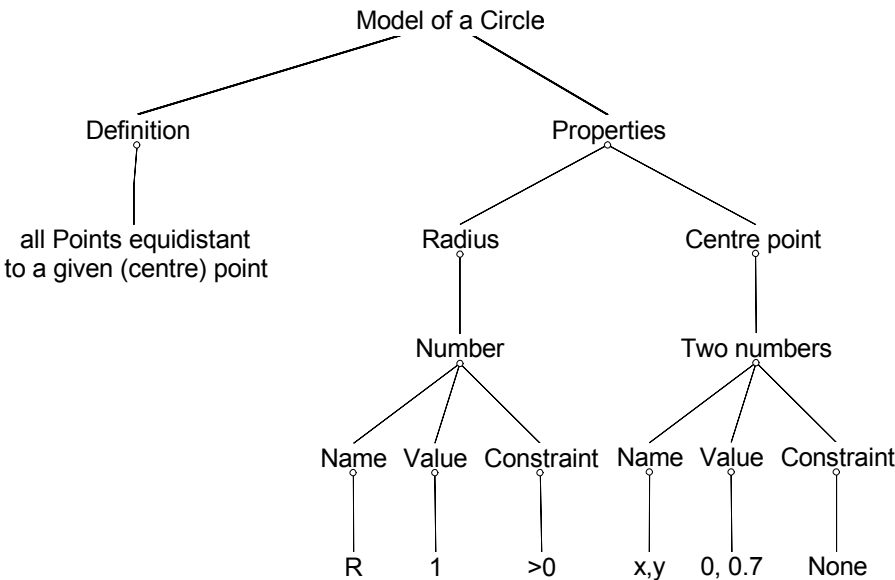


Figure 2.17. Simplified model of a circle

An interesting point for discussion could be whether the definition itself is a property or not. Later on we shall discuss other possibilities, but for the purposes of the current simplified model, let us leave it the way it is now.

2.2 Model-related Terms and Notions

The task of formalizing everything is like the construction of a medieval cathedral: it takes centuries to complete, and when it is done, someone else will have a plan for an even grander cathedral.

John F. Sowa
Knowledge Representation

Numerous terms are used in the literature in connection with models and modelling. Some of them are already well known and established, some, still contradictory or simply not popular. In order to avoid misunderstanding, we introduce in this section the terminology to be used throughout this monograph as well as some characteristics and properties of models.

2.2.1 Prototype

In mechanical engineering, we often hear the term *prototype* in addition to the terms model and modelling. The definition in Sinclair *et al.* (1987), for example, puts prototypes in a strange relation to models, namely, prototypes seem to be restricted to models only, which is certainly not the case. “*A prototype is the first model that is made of something; P. is used as a basis for later improved models*”. In yet another dictionary, we read: “*A prototype is a new type of machine or device which is not yet ready to be made in large numbers and sold.*” And as a second meaning: “*if you say that someone or something is a prototype of a type of person or thing, you mean that they are the first or most typical one of that type.*” Obviously, the latter two interpretations are more precise and do not lead to contradictions. If we replace the word “model” with the word “instance” in the first definition, all three interpretations would become compatible. With this adjustment, we can adopt the following definition for the purposes of the current study:

Definition 2.6: A prototype is the first instance that is made of something; P. is used as a basis for later improved instances.

The prototype is typically a real material thing, but in some cases it can also be virtual – in the sense of *imaginary* or not perceivable by the five human senses. The possible combinations between the type of the prototype, the type of the mature product (or end product) and their “virtuality” are sketched in Table 2.1. We can see in this table that the strangest predicted combination would be to prepare a real (in the sense of material) prototype of a virtual end product.

A similar comparison of modellee types with prepared models demonstrates that in this case all predicted combinations are possible (*cf.* Table 2.2).

Within any product's lifecycle the prototype comes clearly before the mature product. But on the basis of what is the “first instance” made then? And if we can model real objects, is it possible to model objects that still do not exist? And how should they be called?

Table 2.1. “Virtuality” of prototypes and models

#	Mature product	Prototype	Possibility	Plausibility	Example
1	real	real	yes	yes (most common case)	mock-up
2	real	virtual	yes	yes	digital mock-up
3	virtual (unperceivable)	real ⁵	hardly	?	?
4	virtual	virtual	yes	yes	software

Table 2.2. “Virtuality” of modellees and models

#	Modellee	Model	Possibility	Plausibility	Example
1	real	real	yes	yes (most common case)	mock-up
2	real	virtual	yes	yes	digital mock-up
3	virtual (unperceivable)	real ⁵	yes	yes	sketch of a magnetic field? listing of a program?
4	virtual	virtual	yes	yes	software

2.2.2 Archetype

It is possible to model everything – modellees can be existing and non-existing, real or virtual, abstract or concrete. Often the possibility to distinguish between the modelling of already existing and modelling of not yet existing modellees is crucial. The latter case is of special interest, since it is specific for every novel product or process. The models in such cases are successors of ideas, but in our view there is one additional intermediate stage between a new idea and the model or prototype of any future product: the *archetype*.

In Sinclair *et al.* (1987) the term is explained as follows: “*An archetype is something that is considered to be a perfect or typical example of a particular kind of person or thing, because it has all their most important characteristics.*”

The archetype can be viewed as a mature well-elaborated idea, which can create a clear vision of the modellee in a modeller’s head. It is a bearer of the

⁵ In this case “real” is used in the sense of “made of some material”.

inherent (or most important) traits and functions of the desired (or designed, or modelled) object, product or process.

Definition 2.7: The elaborated idea that is (or has to be) modelled will be called archetype.

Note that according to this definition, which will be adopted here, the archetype is always abstract or immaterial.

2.2.3 Interrelations Among Important Terms within the Product's Lifecycle

Now let us return to the modelling of real objects. Possible modellees (*i.e.* the objects to be modelled) originate either in nature or from organized manufacturing. An interesting question in the latter case (no matter whether the products are under development or are already mature) is which one is primary, the product or the model? This question is similar to the famous question about the primacy of the egg and the hen. Different viewpoints can obviously lead to different answers, but what remains viewpoint-independent is that there is a relation or connection between the two. To avoid contradictions and reduce the uncertainties related to these terms, let us consider Figure 2.18.

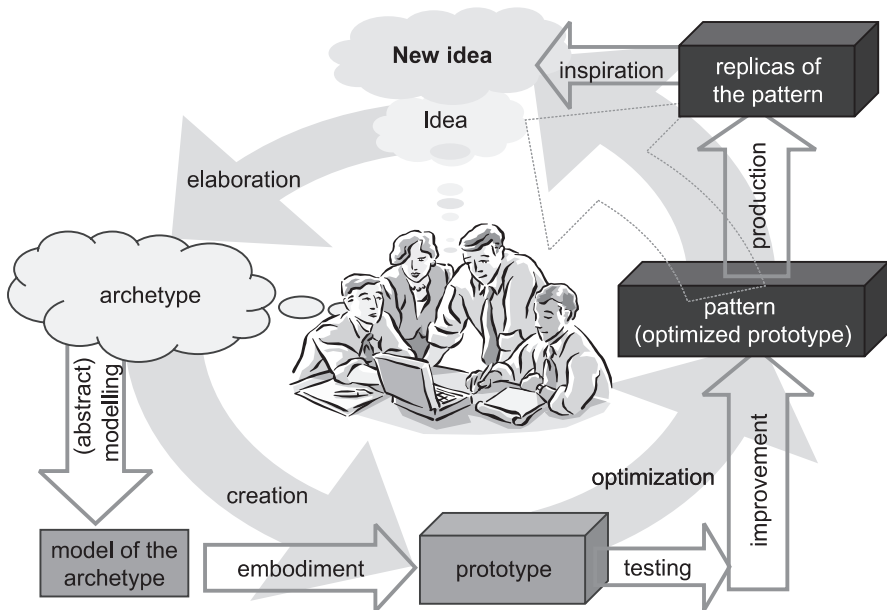


Figure 2.18. Interrelations among idea, model, archetype and prototype during the development cycle of an original product

Normally, at the beginning there is an idea. It is elaborated until there is enough information in it to prepare an archetype on its basis. On the basis of the archetype, a model is prepared. After that, through some manufacturing process, the model is embodied into a prototype. The prototype, in turn, is tested, improved and

optimized until it can be used as a *pattern* for mass production. During the production, many *replicas* of the pattern are made. Any of these replicas, their production itself, as well as the model, the prototype or the pattern could inspire new ideas or be used directly as modellee and thus initiate the lifecycle (or the development cycle) of a *derived product*, as illustrated in Figure 2.19.

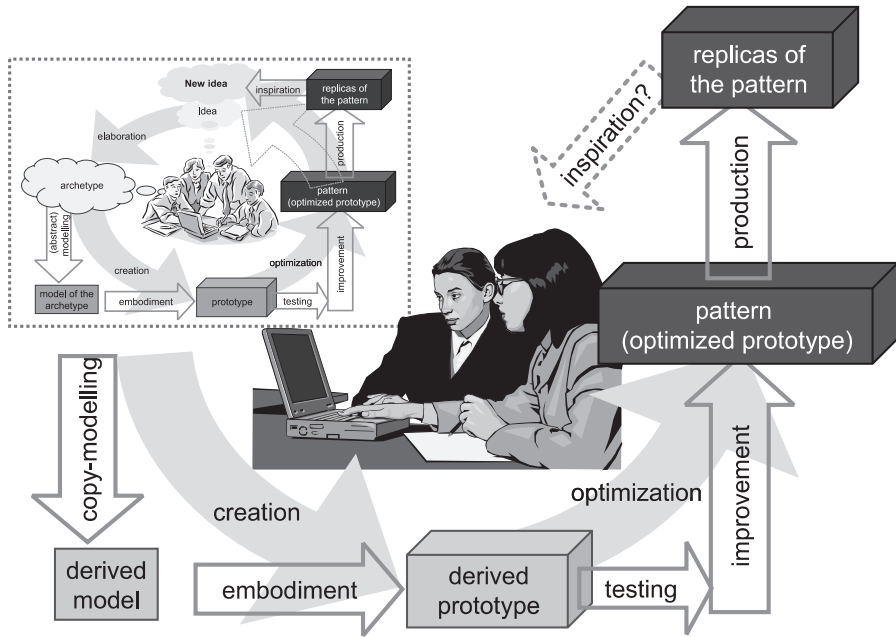


Figure 2.19. Interrelations between idea, model and prototype during the development cycle of a copied or derived product

In certain cases, derived developments are desired and intended, in other cases they are not desired but hardly avoidable, and in the worst case copy modelling or copy production could even be illegal. To summarize, copy modelling is not necessarily a breach of copyright or stealing of intellectual property – it actually takes place in each development loop and helps to improve the initial prototype. As “normal” modelling – shown in Figure 2.18 – has a lot in common with inventing, it can also be named *inventive modelling*.

Consequently, the answer to the question at the beginning of the section is that for inventive modelling the idea is prime, while for derived modelling the product (or object) is prime. This is true especially at the beginning of both processes – more precisely, for their first loop, since already for the second loop of the cycle, the case could become either mixed or transform into the “opposite” kind of modelling.

2.2.4 Process Models

Nothing endures but change

Diogenes Laertius
Lives of the Philosophers

2.2.4.1 Ambiguity of the Word “Process”

Similarly to the terms “model” and “modelling”, the term “process” is used in many different contexts and refers to activities in totally different domains. In this text the focus will be on production-related processes (including modelling itself). We shall understand *product* to be a really existing object that is usually a result of a manufacturing process. We shall understand a *product model* to be a model of an existing as well as not-yet-existing product. Although virtual objects like software models, software, *etc.*, can also be the result of “manufacturing” processes, they will be referred to with their specific names.

Definition 2.8: A process is any non-empty and time-dependent sequence of interactions of two or more objects, leading to changes in (the state of) at least one of the objects.

According to this definition processes are, for instance, the movement and changes in the orientation of an object (they can happen only as result of an interaction with the surrounding objects and are relative to them), changes in the structure or in the form of an object (usually as a result of applying mechanical, chemical or other forces). In the context of this work, we normally understand process to be a production-related one.

There are three terms in Definition 2.8 that could need more discussion: *event*, *time* and *interaction*. We shall define the term *event* as any ascertainable change in the state of an object or system of objects. The second term, *time*, has a tight relationship with the events and helps us distinguish one event from another. The third term is *interaction*, usually defined in terms of an action causing a reaction: in particular, as a pair comprising the action of one object on another one, and the respective reaction (or answer-action) of the affected object. It could be useful to distinguish here two different meanings, though. In the context of Newtonian physics, the action and reaction are two forces that (can) exist only simultaneously; in the context of process control, it sometimes makes sense to view the action and the reaction as two events that are related, but – in general – happening at different times.

2.2.4.2 Time

Time is an illusion. Lunch time doubly so.

Douglas Adams
The Hitchhiker's Guide to the Galaxy. Chapter 2

The main attribute of all processes, common to all of them, is their dependence on time. Although many great scholars have written about time, we shall try to introduce a short and pragmatic definition of this term:

Definition 2.9: For engineering purposes time can be viewed as an invisible but inherent characteristic of the universe, allowing us to correlate different events or processes, order them in a sequence and quantify the “distance” between them.

On the one hand, we try to accomplish each (production) process in the shortest possible time. On the other hand, even if it were possible to work infinitely fast, and thus reduce the accomplishment time to zero, this would be rather impractical, since without time ($\text{time}=0$) we would lose the order of events, ending thus in chaos. Therefore, when there is no need to consider relativity theory, we assume axiomatically that:

3. time depends on nothing (that we could influence);
4. time advances with constant, greater than zero speed;
5. time advances only forward;
6. there are no interruptions in time;
7. all processes are time-dependent.

When all these assumptions hold, we can describe process flows or sequences as functions of time.

2.2.4.3 Relations Between Product, Production and Process

On the one hand, every product is a specific type of object. On the other hand, products are output of some kind of production. Production itself is a process. Therefore, process modelling implies also modelling of objects, or in other words, modelling of production processes implies product modelling, too. A simplified model of a production process is represented in Figure 2.20.

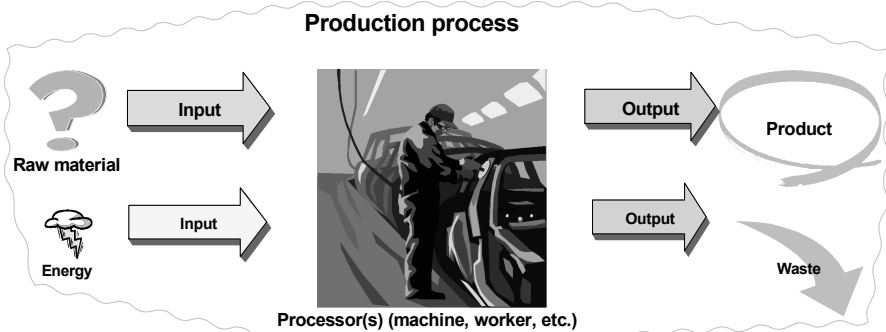


Figure 2.20. A simplified model of a production process

With regard to the relation between process and product, this simple model can be viewed from at least three different perspectives, depending on the starting point:

8. given the process, analyse what can be produced and what raw material is needed;
9. given the (required) product, find how to produce it and what raw material is needed;
10. given a raw material, analyse how it can be processed and what can be produced.

When two of the three elements are given, it is easier either to determine the third element or to make sure that no adequate third element exists.

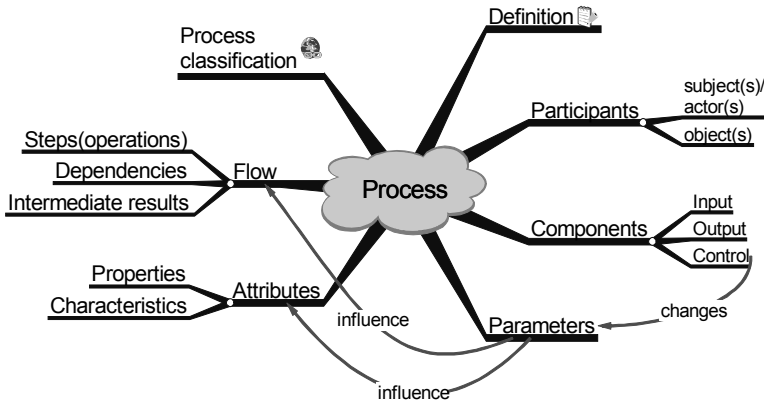


Figure 2.21. A model of a generic process

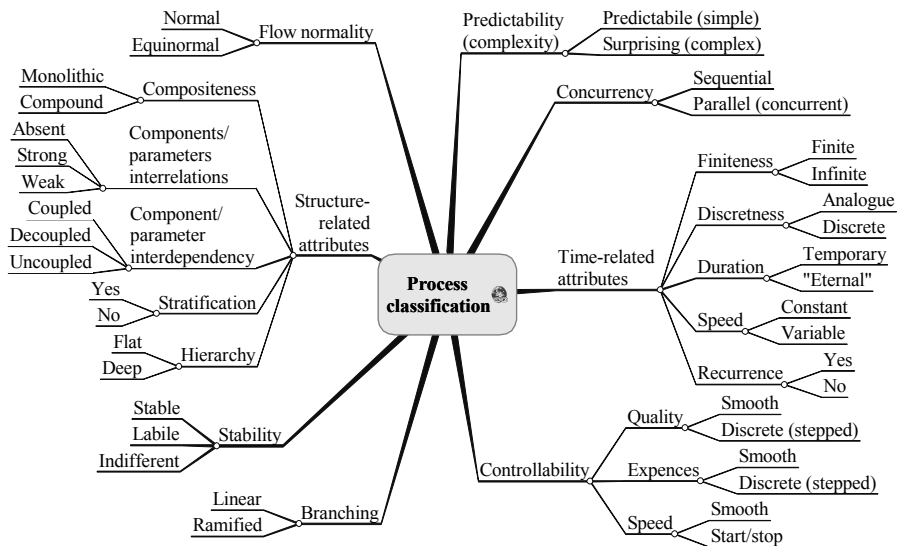


Figure 2.22. Example process classification

In contrast to Figure 2.20, which is normally “read” from left to right, the generic model in Figure 2.21 is focused in the centre, *i.e.* it is to be “read” towards

the periphery, with the reading order – clockwise or counter-clockwise – playing no significant role.

Compared to a function in the mathematical sense, which is a mapping between two sets – input and output – the process represents a similar mapping between (the elements of) an input set and an output set, but it contains a time component, *i.e.* any process needs or takes time.

Processes can be classified according to different criteria. A very systematic classification from cognitive point of view can be found in Sowa (2000). Also, Sowa gives a very intuitive and mnemonic symbolic notation for different kinds of processes and for some of their elements – start, stop, branching, concurrency, *etc.* Another possibility, better suited (in my view) to the field of engineering, is illustrated in Figure 2.22.

The development of every product and every process is influenced by a few major factors. When they are restrictive, we can speak about limitations or constraints, Dörner (1987) even speaks about barriers. An example of influencing factors is represented (in a self-explanatory way) in Figure 2.23.

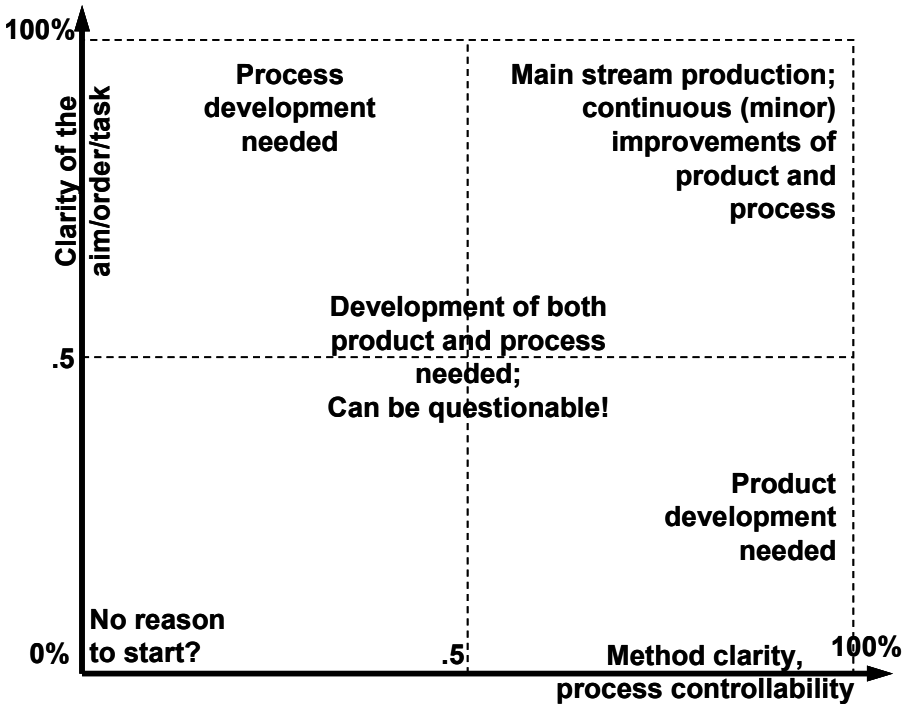


Figure 2.23. Role of some factors, influencing product and process development

In general, three main groups of questions should be asked in respect of the development of a (potential) product. They are given in Figure 2.24.

When trying to detail the answers to these questions it is usually helpful to prepare and consider a (meta-)model of the product development to consider the factors influencing it. An example is presented in Figure 2.25.

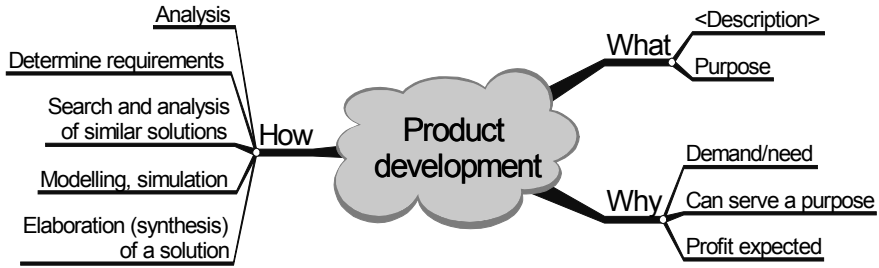


Figure 2.24. The main questions, related to product development and some related notions

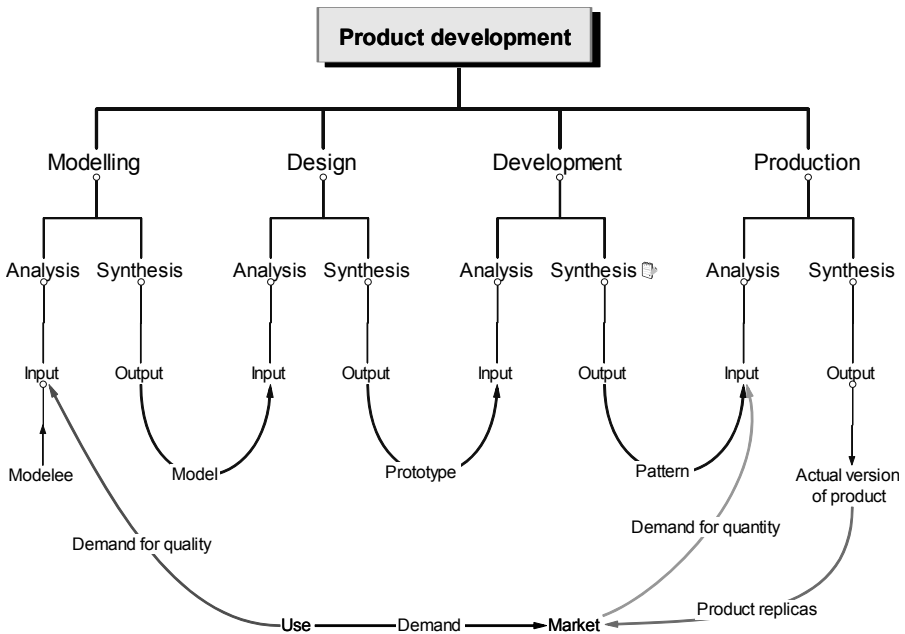


Figure 2.25. Model of the impact of some processes on product development

2.2.4.4 Simulation and Modelling of Processes

An explanation of the term *simulation* in a general sense is found in Wikipedia Wiki (2006) as “an imitation of some real device or state of affairs. Simulation attempts to represent certain features of the behaviour of a physical or abstract system by the behaviour of another system.” A more complete definition of this notion that would be closer to our needs and understanding is adopted here from VDI-Richtlinien (2000)⁶:

⁶ In original: “Unter dem Begriff Simulation versteht man ... die Nachbildung eines dynamischen Systems in einem Modell, um zu Erkenntnisse zu gelangen, die auf die Wirklichkeit übertragbar sind.” (cf. VDI-Richtlinien (2000)).

*Definition 2.10: The implementation of a dynamic system in a model suitable for experiments **and the experimenting with this model**⁷ to gain knowledge that can be transferred (back) to the reality.*

In this definition of simulation, we can distinguish the following essential points. The aim is not only to gain knowledge, but also to use it to act against weak points of the reality and towards its improvement. There are two phases in achieving this: (i) modelling of the dynamic system, and (ii) experimenting with the model. Consequently, the simulation relies upon modelling, where the modellee is a dynamic system.

2.3 Modelling: Classification

Known modelling approaches can be classified according to different criteria (*cf.* Figure 2.2), so let us consider some representative examples.

According to the medium used for the model, we can distinguish between *real* (mock-up) and *virtual* (mathematical, informational, software/computer models, *etc.*) models.

According to the application domain, we can distinguish between medical, psychological, linguistic, engineering, architectural, chemical, physical and other models.

According to the application (sub-)domain, we can distinguish (*e.g.*, within the engineering domain) between modelling in design, manufacturing, assembly, planning, marketing, service and others.

According to the basic modelling tool, there could be (CAx-) system-based, language-based (UML, Express, natural language, *etc.*), and other modelling.

According to the used (software) architecture, it is possible to have client–server architecture, distributed architecture, high level architecture (HLA) and so on.

According to the dominant method or approach, the modelling can be functional, object-oriented, feature-based, distributed, *etc.*

According to the involved concepts, the modelling can be modular, agent-based, holonic or other type.

It is also possible to classify the modelling according to the characteristics of the resulting model, which are to be guaranteed or expected.

A classification of several possible model types according to some key criteria is illustrated in Figure 2.26.

2.4 Model Traits

Ideally, each model would have or at least represent all the important traits of the modellee. In reality, the set of traits of the modellee and the set of traits of the model have a common subset, but are rarely identical (*cf.* Section 2.4.1.18 below). The traits that are specific to the model only, but not the modellee, can be called *model-specific traits*; they represent directly or indirectly the quality of the model. They depend on the modelling approach, on the methods used, on the chosen representation and many other factors.

⁷ The bold text is added from the author to make the idea clearer.

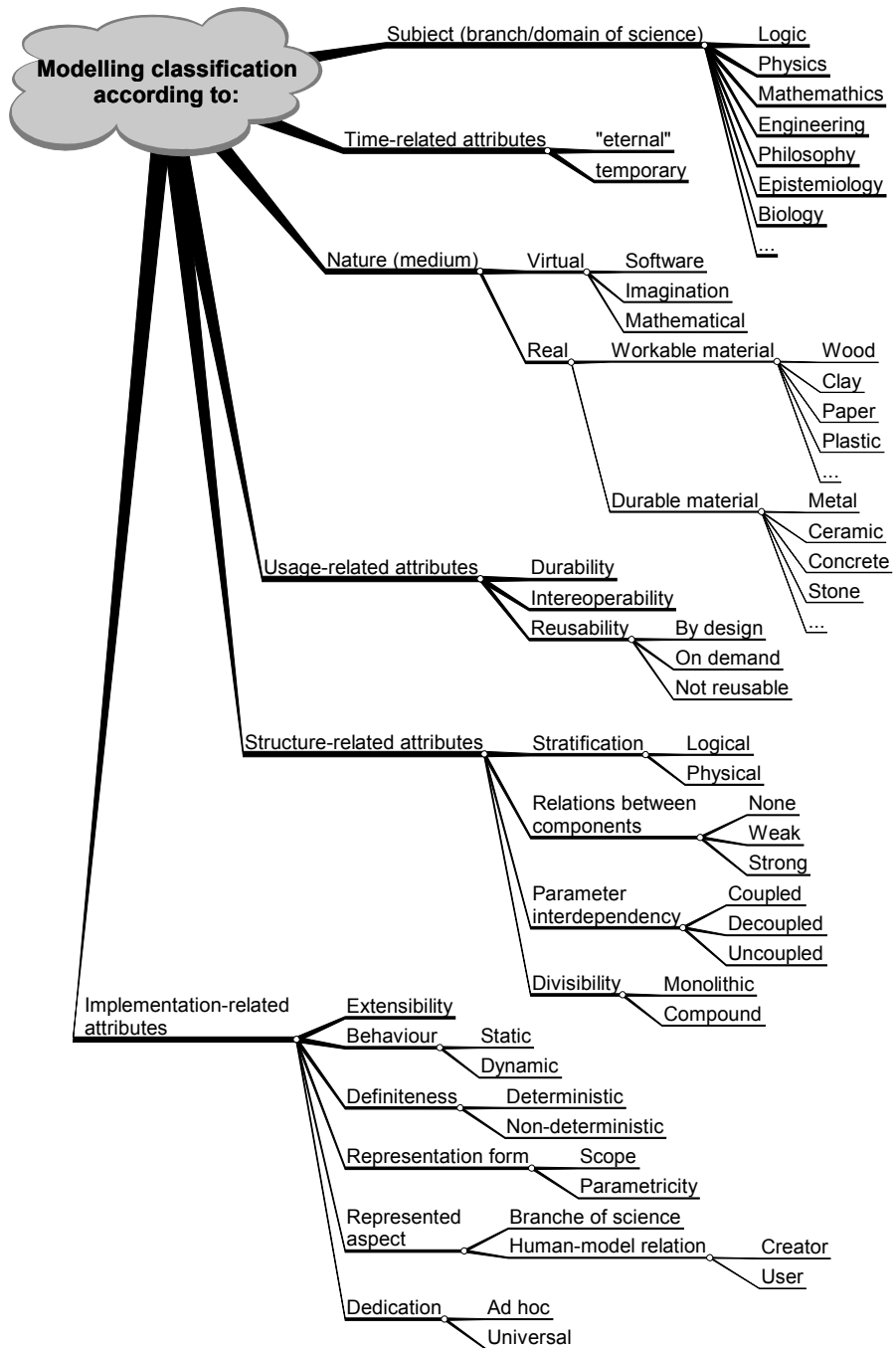


Figure 2.26. Sample modelling classification

The next section discusses some important model-specific traits that can be observed in most information models.

2.4.1 Definitions

It makes sense to define traits and their measurement in a way that will allow us to compare models of different types. This can be achieved if we use either relative values or a common comparison basis. For this reason, the formulae for calculation of the trait values have to be defined so that the range of the respective functions is between zero and one.

2.4.1.1 *Compositeness*

Theoretically, it is possible to distinguish between *atomic* (or *elementary*) models – that contain no other models – and *compound models* – that contain other (atomic or compound) models. Besides, “contain other models” refers not to the physical aspect but to the organization instead, especially in the case of software models.

It makes sense to define compositeness as having a Boolean value: *zero* for atomic modes and *one* otherwise. In practice, the software implementation of an atomic information model is not necessarily also atomic: for instance a point is – both as object in space and as (informational) notion – something single and undividable, but is usually modelled by means of at least two numbers – its coordinates in the coordinate system used. Similarly, the majority of software models are compound, too. The only exceptions are the models of some scalar attributes and properties of the modellees (like the point's coordinates above) – they are modelled by representing their value in a variable of an appropriate type (cf. Section 2.4.2.3.1).

The dependency of any trait listed below on the model's compositeness is to be explicitly mentioned in its definition.

2.4.1.2 *Divisibility*

This trait describes the possibility to split a given model in several sub-models that would be equivalent (as a group) to the initial model. Such activity can be helpful when optimizing composite (and complex) models to factor out a sub-model that is common to two or more models. In this sense the value is Boolean, it can only be true (*one*) or false (*zero*).

2.4.1.3 *Accuracy*

The *accuracy* is a trait, showing how similar to the modellee the model is, and what deviations can be expected. Extremely rarely, the accuracy of a model can be measured directly or calculated – typically, for very simple models only or for separate model traits representable with numbers. Since (the elements of) software models are represented by numbers, it is important to know how accurately the numbers can be represented in a computer. The accuracy of a software model depends on the hardware and on the representation used.

2.4.1.4 *Actuality*

The *actuality* can be viewed as a time-dependent accuracy, meaning that if a modellee is changing with time, its model should be updated or actualized in order to remain useful and fulfil its purpose. In a dynamic environment it is extremely

important to work with the most recent information in order to be able to make proper decisions. On the other side, there is a desire to make the model's lifetime as long as possible (discussed in more detail later on – *cf.* Section 3.1.1.8). These two requirements seem contradictory at first glance – *i.e.* actuality of a model despite its long lifetime. Nevertheless, it is possible to achieve both of them relatively easily by well-directed and well-localized update of the non-actual components of the model. The actuality may change in two situations. On the one hand, it can “expire” if the modellee is changed or new information about it becomes available. On the other hand, new scientific discoveries can also make a model outdated and require its actualization. It can be necessary to assess the actuality in two cases: either to determine whether a given model needs to be updated or to determine which of two (non-numbered) versions of the model is more recent.

2.4.1.5 Adequacy

Whereas the accuracy reflects the mathematical and the numerical representation of a model, the *adequacy* reflects its logical and semantic correctness. Unfortunately, there is no way to measure or calculate this trait objectively.

2.4.1.6 Aspect

Depending on the purpose of the task at hand, each modellee can be viewed from different viewpoints or aspects. If a real three-dimensional object is viewed from two different viewpoints, some of the observed things can be the same, but most of them will be different or look different. Similarly, when modelling complex modellees, it can be useful to distinguish the inherent traits of their different aspects. For instance, when we are modelling the *manufacturability* of a given aggregate, we need to know all dimensions, the shape, the material, the pursued surface quality, among others. When another aspect is modelled, *e.g.*, the *functionality*, traits of the modellee like structure, connections among the components and others become more prominent than material and surface quality. Each different aspect of a given model is usually representable as a distinct layer (*cf.* Stratification).

2.4.1.7 Autonomy

The ability of a model, object or system to react to events and changes in conditions or environment in an adequate and purposeful way will be called *autonomy*. The autonomy is a way of self-control. It can vary on a scale from zero (fully controlled or fully dependent) to one – fully autonomous. This property is rarely inherent to atomic (informational) models. It is natural to expect autonomy of a computer model when the modellee is also autonomous. All four combinations of such a pair (modellee and model) with regard to autonomy are indeed possible.

The term autonomy refers mainly to the lifetime of the respective object (compare with independence below!).

2.4.1.8 Cardinality

In set theory, the number of elements in a given set is called *cardinality*. Analogically, this term will be used here to denote the number of sub-models or components within a model. Only the direct sub-models (*i.e.* without the nested

sub-models) have to be counted⁸. We shall consider elementary models as having cardinality zero.

2.4.1.9 *Changeability*

This property is a measure inversely proportional to the effort needed to change the model. It makes sense to use absolute and relative changeability.

2.4.1.10 *Compatibility*

One entity (part, product, model, *etc.*) is said to be compatible with another entity (usually of the same type) if the former has functionality and properties corresponding to some degree with those of the latter entity. Apparently compatibility can vary on a scale from zero (fully incompatible) to one (fully compatible or equivalent), although a compatibility below 50% should be classified, in general, as “incompatibility”. Some aspects of compatibility are given in Figure 2.27. Depending on purpose and application domain, though, some of these aspects become more prominent, others become negligible, but almost always one turns out to have a dominant importance.

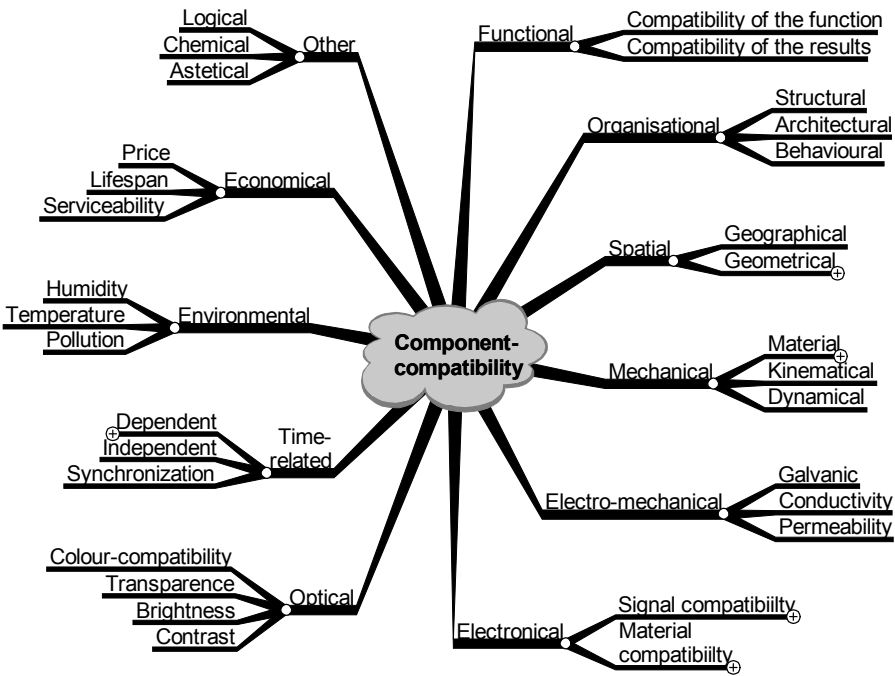


Figure 2.27. Example classification of compatibility

⁸ In contrast to set theory, though, some cases exist where the cardinality of complex models should be calculated as the sum of the cardinalities of all components, applying this rule recursively, if necessary.

2.4.1.11 Consistency

When all components (or sub-models) of a compound model are described and represented without contradictions – *i.e.* they follow the same approach, use compatible methods and organization – we shall say that the compound model is *consistent*.

2.4.1.12 Dimensions

Computer models have only one dimension – their size, measured in bytes or derivative units. Physical models (mock-ups) can have many dimensions. Nevertheless, when a model represents (or “derives”) some dimensions of the modellee, we speak about “*x*-dimensional-model” (*xD*-model), where *x* is usually a number from 2 to 4.

2.4.1.13 Durability

Durability is the ability of something to last. For materials, it depends on the material properties and on the conditions of use (environment, *etc.*). For immaterial things like concepts, ideas and similar, it depends on the durability of the respective host (see below for definition), medium or representation. Although we typically strive for high durability, it is not always good, since it could impair other traits like changeability, extensibility, flexibility and updateability (see below for their definitions).

2.4.1.14 Dynamics

The possibility for a model to (frequently) change appearance or behaviour is called *dynamics*. It is apparent that the model of any process will be dynamic. It is difficult to measure the dynamics. Although at the first glance it seems that there exist objects or models that do not change with time and thus should have *dynamics*=0 (or are static), a more careful look suggests that they actually contain two (or more) phases in their lifecycle that have different dynamics: *creation* (or *genesis*) – with *dynamics*=1 – and *post-creation* (or *use*, or *existence*) – with *dynamics*=0. Therefore, this trait is *time-span-dependent*.

2.4.1.15 Extensibility

This property describes whether it is possible to extend the respective object (model, product, *etc.*) with new functionality or other characteristics. Ideally it should be possible to infinitely extend anything (this could be denoted as *extensibility*=100%), but in the worst case *extensibility* is impossible (*extensibility*=0%).

2.4.1.16 Flexibility

In IEEE (1991) *flexibility* is defined as “*the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed*”. For the case of modelling we should redefine flexibility as the *ease with which a model or a system of models can be adapted for (use in) purposes, not intended or foreseen during the initial development*. Note that a new purpose may require new application, new environment or both.

In some cases flexibility can be achieved only by means of extensions (*cf.* the definition of extendibility above). In cases when flexibility is inherent without need to implement extensions, the term *versatility* is used as a synonym.

The flexibility can be expressed with numbers between 0 (enormous effort for any adaptation) and 1 (no effort for adaptation to an infinite number of purposes). Yet, it can be neither directly measured nor easily calculated. Nevertheless, we have to distinguish the flexibility of a system or compound object (respectively – compound model) from the sum of the flexibilities of its components. The former is usually much lower than the latter! The point is that the purpose is a determining factor, but for a system is defined top-down, while the system's embodiment happens bottom-up. Thus, using a given system with a new purpose would usually mean having a new set of functional requirements, which means that many existing modules would remain unused.

High reusability does not mean automatically high flexibility – if the object in question is reused again and again for the same purpose, it is simply durable, but not yet flexible. If a system can be used for a new purpose without adaptation, this means that either the new set of requirements is a subset of the old requirements, or the system exhibits great lateral functionality (*cf.* the respective section below).

2.4.1.17 Functionality

The *functionality* describes all capabilities of the model. It can be viewed as a set of all functions that a given entity can accomplish. Thus, it can be represented by the cardinality of this set, which is a number between 0 for no functionality and infinity for infinite, endless functionality. Actually, zero functionality would mean that the respective object is of no use, or – when the object is a model – that the model cannot fulfil its purpose. Therefore, the zero remains excluded from the range. The other end of the range – the infinity – is excluded too, since no object or model can accomplish an infinite number of functions. Thus, even the highest functionality will be a huge but countable number.

The requirements for any artefact depend on its purpose and can also be described as a set of functions, which form the *required functionality* (F_{req}). Since not every artefact fulfils its requirements, the “normal” (or *full* or *actual*) functionality is sometimes called *implemented functionality* (F_{impl}). It intersects the required functionality, as illustrated by the Venn-diagram in Figure 2.28.

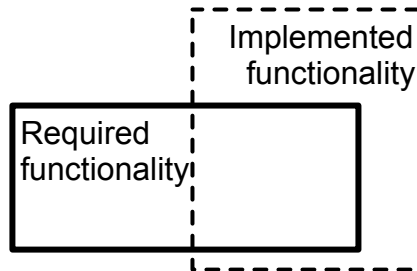


Figure 2.28. Functionality types

Functionality that is required, but not implemented, remains *due functionality* (F_{due}). It can be expressed as:

$$F_{due} = F_{req} - F_{impl} \quad (2.1)$$

Functionality that is not required, but implemented – perhaps, due to specificities of the development process or due to other considerations – can be called *lateral⁹ functionality* (F_{lat}). It can be expressed as:

$$F_{lat} = F_{impl} - F_{req} \quad (2.2)$$

Note that the subtraction in Formulae 2.1 and 2.2 operates on sets and is different from the normal subtraction.

The cardinalities of the respective subsets can be expressed as:

$$|F_{due}| = (0, |F_{req}|] \quad (2.3)$$

and

$$|F_{lat}| \in [0, |F_{impl}|] \quad (2.4)$$

Lateral functionality is very welcome when achieved as a side effect of the development (*i.e.*, without extra effort or costs), since for some new purpose it can become a required functionality and therefore increases the flexibility (*cf.* the definition in Section 2.4.1.16 above).

When the required functionality F_{req} is the same as the implemented functionality F_{impl} or the former is a subset of the latter ($F_{req} \subseteq F_{impl}$) it is said that the artefact fulfils (completely) its purpose. In such cases the cardinality of the subset of all due functions is zero ($|F_{due}| = 0$).

From the point of view of the importance of the specific functions that build the functionality, we can group them in two categories or subsets: *basic functions*, which implement the inherent functionality, and *auxiliary functions*, which implement functionality of lower importance.

When still unknown objects or artefacts are investigated, one can distinguish between *apparent functionality* and (yet) *hidden functionality*.

2.4.1.18 Coverage

Given a modellee and its model made for a certain purpose, the following considerations can take place:

11. Only the important for the respective purpose attributes and functions of the modellee have to be modelled (*cf.* Definition 2.1) and thus represented in the model.
12. Typically, some attributes and functions of the model (would) concern only the model itself and not the modellee.
13. Often there are attributes or functions that are not required, but nevertheless modelled.

If we try to represent the set of attributes and functions of a modellee $A_{modellee}$ and the set of attributes and functions of a model A_{model} as a Venn-diagram, the result is illustrated in Figure 2.29. Apparently, the greater the intersection of the two sets, the better (approximation of the modellee is) the model. We shall call the

⁹ Other possible terms for this notion are *side* or *excess* or *extra* or *unrequested functionality*.

intersection *coverage*, since it reflects to what degree the model “covers” attributes and functions of the modellee, and shall measure it as percentage.

$$Coverage = \frac{|A_{modellee} \cap A_{model}|}{|A_{modellee}|} \quad (2.5)$$

Unfortunately, coverage gives a more quantitative than qualitative impression about the model, since the importance of single attributes and functions is different. Therefore, covering a large number of unimportant attributes and functions can be worse than covering a much smaller but more important number of them.

Similarly to the functionality, the coverage can be represented by Venn-diagrams as in Figure 2.29.

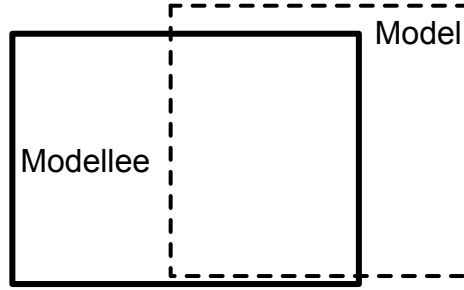


Figure 2.29. Coverage and suitability of a model

Clearly, better coverage means higher quality of the model. But with the increased cardinality of the set mentioned in 12 above, the *inefficiency* of the model also increases.

Now let us consider the coverage of compound models.

2.4.1.19 Compound Models

The traits of any model depend on the traits of its components. Unfortunately, very few model traits are representable as a sum or superposition of the respective traits of the components or by means of a simple formula.

Obviously, the set of all modelled attributes and functions can be expressed as

$$A_{model} = \bigcup_{i=1}^M A_{sub-model_i} \quad (2.6)$$

Similarly, when the modellee is also compound, its respective set can be calculated as

$$A_{modellee} = \bigcup_{i=1}^N A_{component_i} \quad (2.7)$$

Since in both cases overlapping between the sets of attributes and functions of the components and, respectively, of the sub-models can occur (*cf.* Figure 2.30), the cardinality of the top level sets will be smaller than or equal to the sum of the cardinalities of the components. The following formulae hold:

$$|A_{\text{model}}| \leq \sum_{i=1}^M |A_{\text{sub-model}_i}| \quad (2.8)$$

and

$$|A_{\text{modellee}}| \leq \sum_{i=1}^N |A_{\text{component}_i}| \quad (2.9)$$

finally,

$$\text{Coverage}_{\max} = \text{granule_count} * \text{granule_area} \quad (2.10)$$

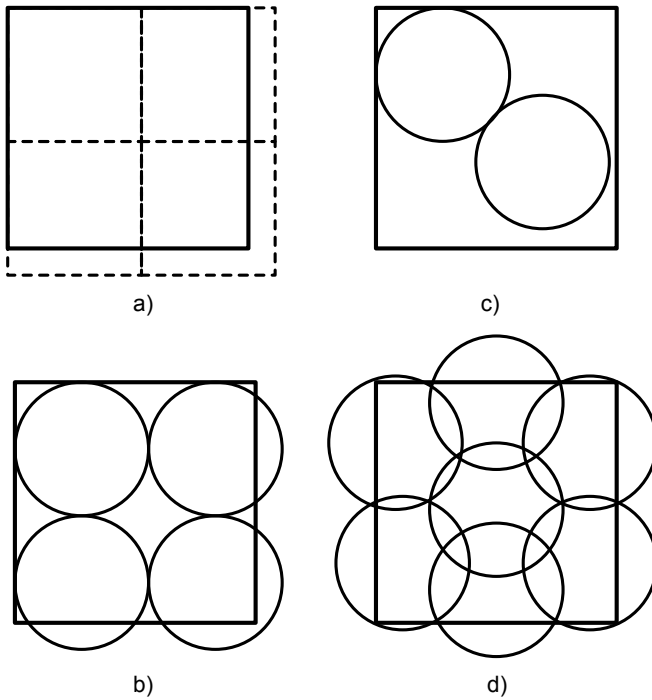


Figure 2.30. Coverage and granularity of compound models

2.4.1.20 Granularity (Only for Compound Models)

According to the Langenscheidt dictionary, the granularity is “a measure for the size of the standalone sub-operations in which a process or program can be divided for achieving parallel processing”¹⁰.

¹⁰ In the original: “**granularity** 1. Körnigkeit f; 2. Maß für die Größe selbstständiger Teiloperationen, in die ein Prozess oder Programm für die Parallelverarbeitung zerlegt werden kann”. Langenscheidt Fachverlag GmbH, München, 1999

In the context of modelling, granularity refers to the average size of all sub-models of a given model. Since some models could have different dimensions, it is important to specify to which of them the word “size” refers in the previous sentence, *i.e.* which of them is taken for determining granularity. For instance, if the quality of the modelling is assessed, an appropriate measure for the “size” could be the coverage of each sub-model. If the efficiency of the memory usage is assessed, a better candidate for the dimension to be used could be the size of each sub-model in bytes.

Since the granularity can be very useful for comparison or assessment of compound models, it will be discussed again later on.

2.4.1.21 Homogeneity

This property shows whether all sub-models have the same (type) of origin and are thus homogeneous and directly compatible with one another, or have different (types) of origin and are heterogeneous. Sub-models of the latter type typically require special effort for their integration.

2.4.1.22 Independence

This is a measure of the strength of the relations to or of the dependencies on other elements of the surrounding system or environment. It could be related to or combined with model properties like existence, functionality and others.

Unlike autonomy (*cf.* the respective section above), independence is more related to the genesis of an object than to its lifetime.

2.4.1.23 Intelligence

This property is discussed in Section 2.4.2.3.1.

2.4.1.24 Interchangeability

If two entities (real or virtual) are fully compatible with each other (*i.e.* equivalent) and each can be used instead of the other without discernable loss of functionality, quality or anything else, we say that they are interchangeable. When a single entity is said to be interchangeable, it is meant that the design of the entity provides such a possibility and that spare parts of the same type are deliverable. Interchangeability is usually viewed as a binary (*i.e.* true or false) property (*cf.* also *compatibility* above).

2.4.1.25 Openness and Modifiability

The term *openness* refers to the possibilities of changing or extending any given model, and is *implementation-dependent*. The less functional a given model is, the higher is the probability that new desires concerning its functionality will arise, so that the model will have to be extended. The more complex a given model is, the higher is the probability that errors will occur or (for mechatronic systems) failures will happen during the exploitation, so that the model will have to be corrected/changed/repared at the end user's place. For pure software models this is seldom a problem, but for complex mechatronic systems the distance to the place of use could cause problems (or at least additional costs).

Increasing the openness of a given model has strong influence on many of its other traits. In most cases it is positive – extendibility, flexibility, integrability, *etc.* In one aspect, though, the change is negative: the increased openness of a model

makes it easier for the competition to imitate. For this reason many producers of software and software models sell their products as turnkey products. A necessary condition for achieving model openness is a clear definition of its interfaces. Depending on the model type, the interfaces can be mechanical, electrical, software, or any combination of these.

2.4.1.26 Paradigm

Webster's dictionary gives the following definitions for paradigm:

Example, pattern; especially: an outstandingly clear or typical example or archetype.

...

A philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated.

Another explanation is found in Wikipedia: "*From the late 1800s the word paradigm refers to a thought pattern in any scientific disciplines or other epistemological context.*"

One of the most popular paradigms in modelling is the *object-oriented modelling* (or *object-oriented paradigm*), related also to *object-oriented analysis*, *object-oriented design* and *object-oriented programming*. The main objection to all OO techniques is that the attribute "object-oriented" is somewhat misleading. Actually, the focus of these techniques is the grouping of similar objects into classes in order to factor out the common knowledge (data, procedures, *etc.*) about them and to increase the efficiency through reuse. Therefore, an attribute like *class-oriented* would be more self-explanatory.

2.4.1.27 Platform

By *platform* we shall understand the set of hardware, operating system and possibly additional software, providing an environment for a software product or software model to "live in".

2.4.1.28 Portability

Portability is usually defined as the easiness of making a model usable on a different platform, *cf.* for instance Howe (2006). For the purposes of computer aided engineering I would define it as the (average) easiness of making a model usable on any possible platform. It is inversely proportional to the effort necessary to adapt the model for use on a new platform. This effort is proportional to the number of platforms and to the complexity of the model to be adapted. So it is not trivial to compare the portability of differently complex models.

One of the ways to achieve (better) portability is to develop the models upon a layer (or basis) that is already portable.

2.4.1.29 Effort for Porting to new Platform

Very often it is necessary to make already existing model available and functioning on a new platform. The process is called *porting* or *migration* and the additional work to achieve this is the *effort for porting* (*cf.* Figure 3.5 in the next chapter).

2.4.1.30 Platform Independence

We shall understand by platform independence, the ability of a software application or software model to run on different platforms without (or with minimum) changes for adaptation.

According to Frankel (2001, p.25), the notions of main importance to achieve platform-independence have evolved from the 1960s until now starting from processors in the 1960s, through 3GLs (third generation languages) in the 1970s and 1980s, through middleware in the 1990s and MDA (model driven architecture) since the new millennium.

2.4.1.31 Quality

It is difficult to measure quality since it depends on the purpose of the model. The same model can be perfect for one purpose but totally unsuitable for other. For this reason, no mathematical definition will be given, but let us consider one guideline of The Association of German Engineers (*cf.* VDI-Richtlinien (1993)):

The quality of the model is decisive for the quality of the analysis results. Only if the model realistically describes the system, it is possible for the subsequent model analysis to produce results that can be transferred to reality.

2.4.1.32 Reliability

According to Howe (2006) reliability (of a system) is “*An attribute of any system that consistently produces the same results, preferably meeting or exceeding its specifications. The term may be qualified, e.g. software reliability, reliable communication.*”.

$$Reliability = F \left(reliability_{Host\ system}, \prod_{i=1}^N reliability_{component\ i} \right) \quad (2.11)$$

2.4.1.33 Reparability

During its use any model can get broken, malfunction or cease to be useful. This could happen due to internal problems (specific mainly to material models) or due to a critical change in the environment. The latter can impact on all kinds of models, including software models – a typical example was the problem of the 2000th year¹¹. We shall call *reparability* the possibility to restore the functionality of the respective model or the conditions allowing us to use it in accordance with the initially foreseen purpose.

2.4.1.34 Reusability

Before defining the term “reuse” as “second or multiple use of something”, let us see what we mean by “(first) use”. Some reasons for ending the “first use” of a product are listed below.

¹¹ This problem caused calendar-related modules in some improperly designed software programs and electronic devices to function improperly due to overflow of the year-counter.

- There is no more need to use it for the initially foreseen or intended purpose.
- It gets broken or there is a malfunction.
- There is no qualified user anymore.
- Its use is not legal anymore.

In case b) we can say that a *purposeful* or *natural reuse* can be pursued. In case a) one could try to reuse the product for alternative purposes. Since the aim is to achieve economic advantages or even profit, a suitable term here can be *economically based reuse*.

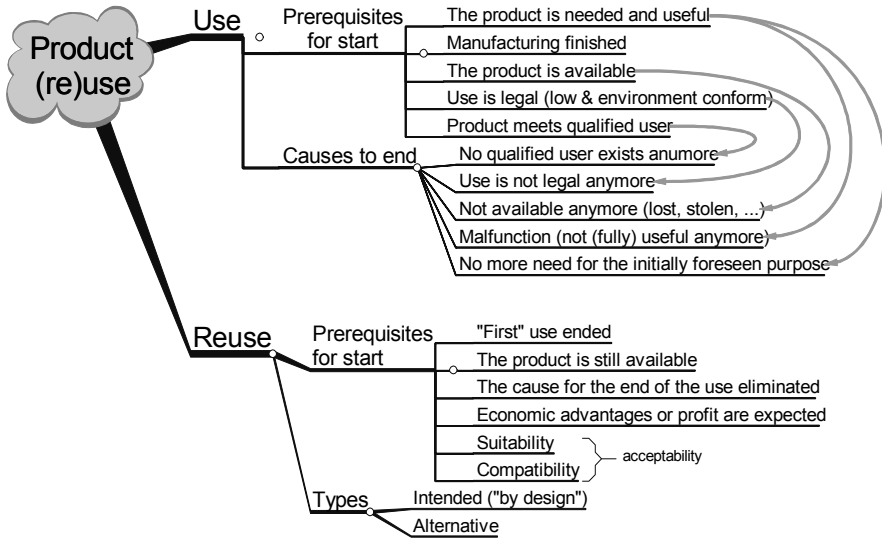


Figure 2.31. Conditions and prerequisites for (re)use of a product

Depending on the flexibility of the product and – with a composite product – on the flexibility of the components, several possibilities have to be considered. Of course, in the best case every product is fully reusable and in the worst case – absolutely not reusable. In the majority of cases some subset of the components of any composite product can be reused.

$$Reusability = f \left(\sum_{i=1}^N (R_{Component_i}), \sum_{i=1}^N \left(\sum_{j=i+1}^N (R_{Link_{i,j}}) \right) \right) \quad (2.12)$$

2.4.1.35 Robustness

In general, this term is used to denote the capability of a product to keep its integrity, usability and functionality despite negative and possibly unforeseen influences of the environment. In the context of computer science, the meaning is extended to cover not only hardware but also the software, including behaving incorrectly or – possibly intentionally – even illegally.

2.4.1.36 Scalability

When a greater need for a given function can be satisfied by employing more instances of the respective model (or object), working in parallel, we shall speak about *scalability*. This trait belongs more to the result of a process than to the respective “processor” and does not always exist. For instance, we can use more cars for transporting more things, but more cars cannot help us travel faster – we need another quality. In other words, we could define the scalability as the possibility to trade quantity for quality.

2.4.1.37 Size

The size of a software model is the volume of memory it needs to be saved on disk – sometimes referred to as *static size* – or in the operating memory – also known as *dynamic size*. It is measured in bytes or their derivatives. The size of a software model is also an indirect measure of its complexity (*cf.* Section 3.1.2 in the next chapter).

2.4.1.38 Time Dependency

Time dependency explains whether a given model changes with time or not. It makes sense to define this trait as having a binary (or Boolean) value, since atomic models are either time-dependent or not. A compound model becomes time-dependent if any of its elements is time-dependent. All models of processes are time-dependent, too.

2.4.1.39 Universality

This property shows for how many different purposes a given model is well suited. More purposes mean higher universality and *vice versa*. Although it can be tempting to develop models with universality as high as possible, it is not always rational to do so. And achieving a full universality – *i.e.* developing a model that is suitable for all thinkable purposes – is impossible.

2.4.1.40 Updateability

The more valuable the trait actuality of a given model is, the more important becomes the possibility to update this model regularly. The actuality of a non-updateable model (*e.g.*, a wood mock-up) can only decrease with time, while the actuality of an updateable model can be improved regularly and on demand.

2.4.2 Organization of Models

In order to understand models, their capabilities and interaction it is important to analyse how they are organized. The two most often used terms in this respect are *structure* and *architecture* of models. Some of their more important properties, as well as their interrelations, are visualized in Figure 2.32.

Static models (*e.g.*, a physical mock-up of an object) do not have organization – they have only structure and sometimes also architecture. Monolithic objects have neither structure nor architecture, but they still can have static functionality and relations with the environment. Software models are usually dynamic models, having always structure, functionality, architecture, external relations (interfaces), and are unique in the sense that they have a programmable behaviour. For these

reasons we shall say that they have organization too. Many authors use the terms organization and architecture as synonyms, but for the above-mentioned reasons I consider the architecture a component of the organization.

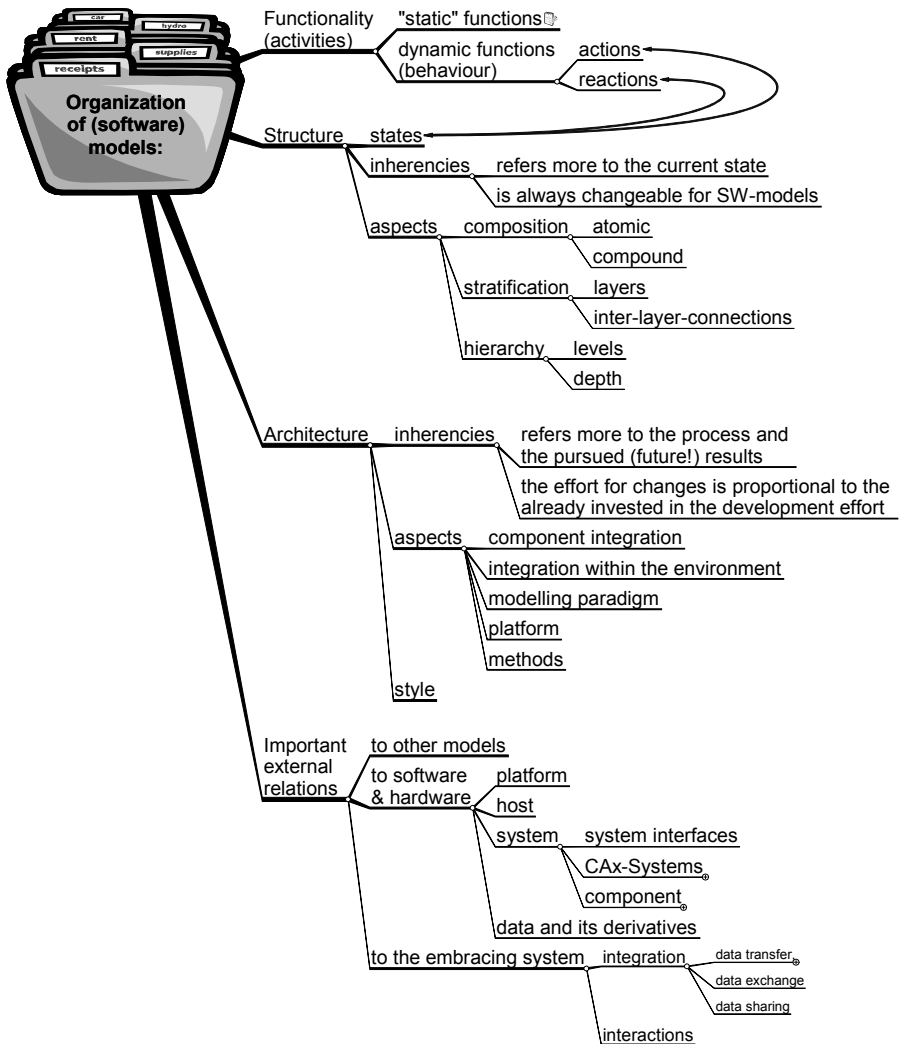


Figure 2.32. Organization of software models

2.4.2.1 Structure

The structure of a model describes its components and the (static) relations among them. Only compound models have structure.

2.4.2.1.1 Inherencies

The structure of a model refers mainly to its current (or already achieved) state in the model's lifecycle. It can be (and usually is) changed between the iterations of

the development process. The structure of a software model is always changeable, but this is not true for existing non-software models, *e.g.*, for mock-ups.

2.4.2.1.2 *Structure-related Aspects of Models*

The structure can be viewed from several viewpoints or aspects.

2.4.2.1.2.1 *Composition*

This attribute describes how the models are built. The simplest models are the so-called *atomic* or *elementary models*. They are *monolithic* or *not dividable*, having thus no structure but being used in building all other (non-atomic) models. Atomic models are often used for modelling properties of a modellee that are representable as scalar values.

Assuming that several models are available, they can be *grouped* according to different criteria. A group can be useful for easier reference to all objects simultaneously or for defining and performing operations on all elements of the group at the same time. Depending on the purpose, different criteria can be chosen. Grouped models can come from the same library or developer. They may be independent from one another or even be hosted on different platforms.

Models comprising other models are said to be *compound models*. For instance, the model of a circle in Figure 2.17 has (among others) the properties radius and centre point. The former property can be viewed as a model of a scalar, the latter as a model of a point; therefore the model of the circle itself is a compound model. Any compound model can be viewed as a group of models (*e.g.*, the radius and the centre point can be viewed as a group of models, belonging to the same compound model), but not every group is a compound model – *e.g.*, the group of all models, created by the same modeller are not necessarily parts of a (larger) compound model.

Often we have to model several objects (or modellees), which do not belong to one another but are interacting within a given process or are somehow related, and we need to model this relation. In such cases, we speak about *system of models*. If the involved models are physically or geographically distributed, but still interact with one another, they can be referred to as a *distributed system of models* or a *system of distributed models*.

2.4.2.1.2.2 *Stratification*

Another structure-related aspect of models is their stratification. It is inherent to some compound models and reveals the existence of several *layers* in their structure. The criteria for stratification are very interesting from a scientific point of view, *cf.* Figure 2.33.

Since every model is similar to the modellee, it is natural for a model to be layered if the modellee is layered too. Nevertheless, all combinations between the two are possible, as illustrated in Table 2.3.

In some cases the modellee can be viewed from different viewpoints and thus exhibit different *aspects*. From an organizational point of view the aspects can be represented either as different layers in the same model or as different standalone models. An important characteristic of the different layers is that they are always connected in a certain way. In the example for case #2 in Table 2.3, for instance, the values of the three colour components are connected with one another so that they form together the modelled colour.

In general, there are properties that are represented in a single layer, in all layers, or in a subset of layers. We shall call the modellee properties that are present in (almost) all layers *core properties*, those that are present in at least two layers – *essential properties*, and the rest – *aspect-specific* or *auxiliary* properties. It is obvious that the core and the essential properties can be used to build *inter-layer-connections*. They play an important role in integration of separately modelled aspects.

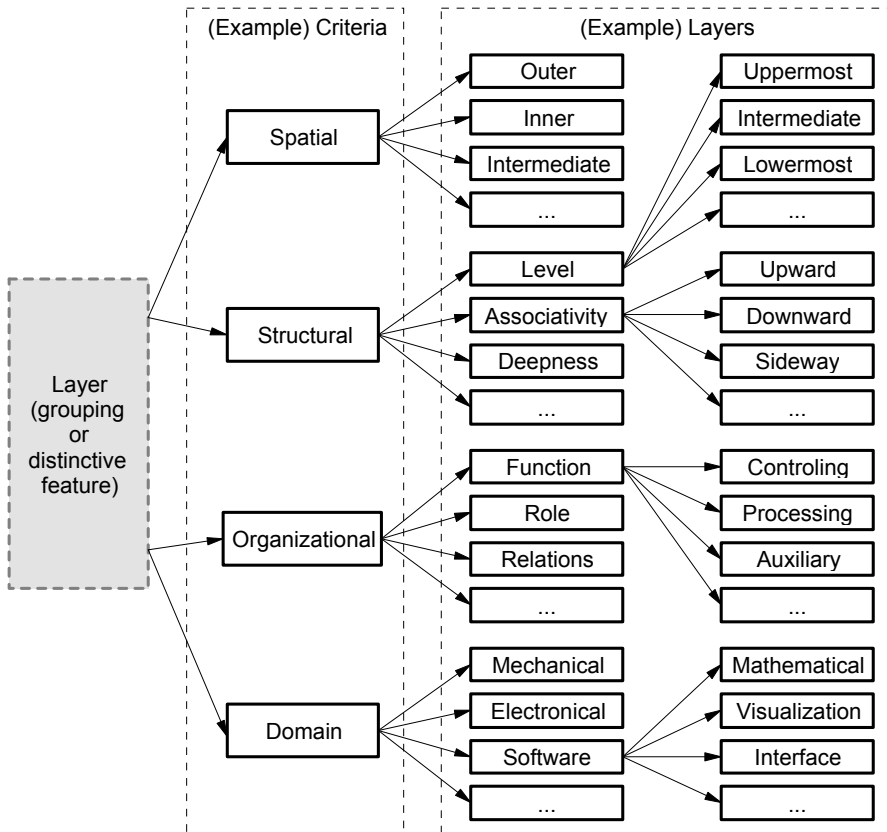


Figure 2.33. Criteria for defining layers

2.4.2.1.2.3 Hierarchy

The hierarchy is an attribute of the structure, which describes the systematic character in the order of the components and their relations. It can be said that a hierarchy is vertically divided in several *levels*, whereas each level can contain one or more components. In many cases we have to do with nested structures of models, in which hierarchical structures are clearly recognizable. A simple hierarchy is given in Figure 2.17.

Table 2.3. Stratification of modellee and model: possible combinations

#	Stratification of the:		Possibility	Plausibility	Example
	modellee	model			
1	no	no	yes	yes	sculpture of a man
2	no	yes	yes	yes	all colour models (RGB ¹² ,CMYK ¹³)
3	yes	no	yes	yes	clay model of a car
4	yes	yes	yes	yes	car and a model car

2.4.2.1.2.3.1 Levels of Hierarchical Structure

Levels are the biggest components of each hierarchical structure. Splitting of sophisticated systems, models or objects in several levels, where each level consists of approximately the same number of objects to deal with, is an often used method for hierarchical structuring. It simplifies the manipulation of both the separate levels and the objects within each level.

The same holds for sophisticated models, but sometimes the number of levels in the model is not equal the number of levels in the modellee – it can be both greater or smaller. One example of a “Multi-dimensional Meta-modelling Architecture” is given in Jeckle (1999, p.11). He describes five level of a modelling hierarchy:

- M^{-1} : Instance
- M^0 : “Reality”
- M^1 : Modelling language
- M^2 : Meta-language (“Grammar”)
- M^3 : Meta-meta-language (“Meta-grammar”)

Again there is discussed the use of the XML Metadata Interchange Format (XMI format) within the four layer Meta-model Architecture Jeckle (1999, p.14).

Depending on the purpose and the point of view, the same structure can sometimes be interpreted either as layered or as hierarchical (levelled). For instance, depending on distinctive characteristics, the group of models clothes(skin(muscles(skeleton))) can be viewed either as nested, hierarchical structure or as models belonging to the different layers beauty, protection, movement and support.

2.4.2.1.2.3.2 Depth of Hierarchical Organization

This characteristic can give us an impression of the complexity and the possible minimal and maximal number of elements in a hierarchy. It can be used either to show the relative position of an element or to denote the depth of the whole hierarchy. The model in Figure 2.17, for instance, exposes a hierarchy of depth 6.

¹² RGB: method for representing any colour as a mix of the primary colours red, green, blue.

¹³ CMYK: A colour model that describes each colour in terms of the quantity of each secondary colour (cyan, magenta, yellow), and “key” (black) it contains. The CMYK system is used for printing Howe (2006).

The minimal possible depth is one, which implies that the organization is *flat* or there is no hierarchy. There is no restriction on the maximum depth, but the complexity of organization rises exponentially with each new level in the hierarchy.

2.4.2.2 Architecture

This term is so ancient that it is usually viewed as already known and is rarely defined. One of the few definitions that can be found is given in Wikipedia Wiki (2006):

Architecture (in Greek αρχή = first and τέχνη = craftsmanship) is the art and science of designing buildings and structures.

The literal translation makes an allusion to the fact that many creators have imitated already existing artefacts, since this is easier than creating a totally new artefact. Often the first artefact has already established a pattern or even a norm that is simply followed by the others.

2.4.2.2.1 Inherencies

We can think of three properties that are inherent to any architecture. It defines how the subcomponents one level below the top level are to be put together. This depends on the pursued results. In fact, changes are always possible, although more and more difficult with time.

2.4.2.2.2 Architectural Aspects of Models

Several aspects of model architecture are clearly distinguishable: the model structure, the integration within the environment, the modelling paradigm, the platform and the methods to be used.

To summarize, the architecture can be defined as the combination of concepts, approaches, methods and techniques that are used in the initial building phase of any model, system, or other compound and complex object, and therefore plays a crucial role in determining its most important traits and its entire future – development, use, maintenance, re-use, *etc.*

2.4.2.3 Important External Relations

2.4.2.3.1 Relations (of Software Models) to Data and its Derivatives

On the lowest level of all kinds of information and knowledge structures is the data. Each of the levels above can be viewed as a data derivative. Since these derivatives play an enormous role in modelling, we shall explain briefly some of them.

2.4.2.3.1.1 Data and its Derivatives

2.4.2.3.1.1.1 Data

We shall understand data to be *strings of (ordered) symbols*. These symbols are represented in computers by numbers, and in turn, the numbers are represented by binary digits or bits. Apart from bits, the numbers are the smallest “building block” in the representation of data and data derivatives – including algorithms – in a computer. An example of such a string of symbols is “3.1415”.

2.4.2.3.1.1.2 Meta-data

The meta-data is a connection or relation between groups or pieces of data. Given the strings of data “5.003” and “5.02”, we can connect them into a relation, for instance by the sign “smaller than”:

$$5.003 < 5.02$$

Thus, the “<” symbol has special meaning and is meta-data in this case.

2.4.2.3.1.1.3 Information

The information emerges from interpreting data. Interpreting means that each piece of data and meta-data is connected or put into a relation with already known facts as well as with all other pieces of data. Thus, the example above would be interpreted as putting two numbers in a relation, saying that the second one is greater than the first one. To achieve this, the interpreter (human or machine) should have some *a priori knowledge* – i.e. to be able to read the numbers and to understand the meaning of the sign “<”. This knowledge is often called *context* or *background knowledge*.

Since the context always plays a crucial role by gaining information from given data, another possible definition for information is *data in certain context*. Thus, when hearing the (incomplete) expression “to be or not to be...” people, who are experts in different domains, can interpret it differently. A fan of Shakespeare could see an allusion to the famous phrase of Hamlet; a philosopher could think about The Question of Douglas Adams' book *Life, the Universe and Everything*, and a mathematician could write it down on a piece of paper as $2b \mid \neg(2b)$ and say “this is always true!”.

Another possibility to define the term information is as a combination of meta-data and (groups of) data that are to be connected/related, for instance:

$$\pi \approx 3.1415$$

Such a combination of data and meta-data is usually named an *attribute-value pair*. In representing more complex information it is possible to nest attribute-value pairs by using a given pair as the value of another one. A sample graphical representation of nested attribute-value pairs can be seen in Figure 2.34.

When the value of an attribute-value pair contains just data (i.e., there is no nesting), this pair can be named *basic* or *substantial* attribute-value pair. Independently of their representation, basic attribute-value pairs can be viewed as the smallest units of information.

2.4.2.3.1.1.4 Meta-information

Meta-information is information about other information. For instance, the node “Invariable properties” in Figure 2.34 is meta-information and denotes that all nodes below it are invariable for the whole class of objects of the circle type.

2.4.2.3.1.1.5 Knowledge

We shall call *knowledge* the ability to gain new information from already existing information or data. For instance, knowing that the circumference of all circles is equal to $2\pi R$, and that the radius of a given circle C is $R=1$, we can conclude that the circumference of C is equal to 2π .

Another possible way to define knowledge is to view it as the union of the meta-information and the pieces of information that have to be connected.

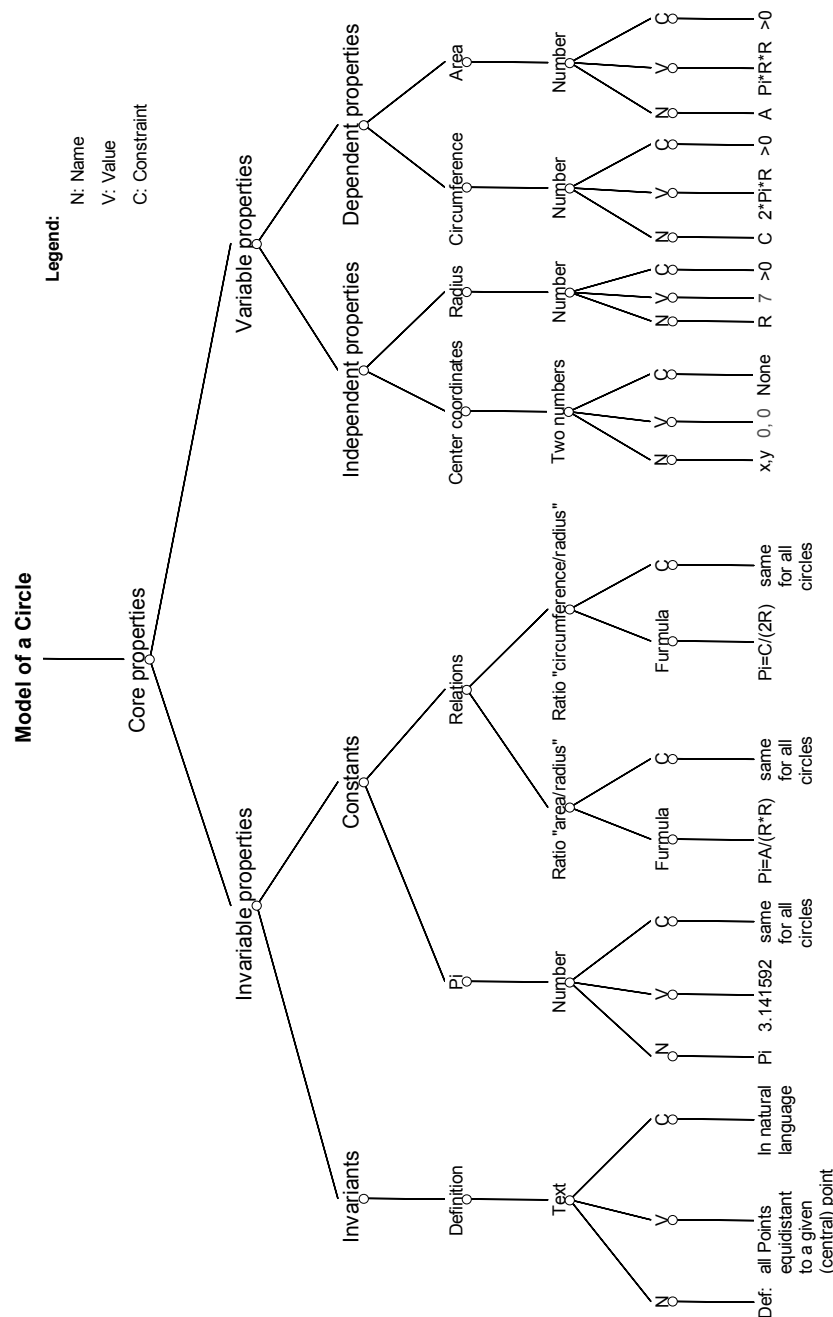


Figure 2.34. Representation of the data, information and knowledge levels for a simple model

2.4.2.3.1.1.6 Intelligence of a Software

Intelligence is a quality that is inherent mainly to human beings and is usually related to thinking and especially to the *ability for reasoning* on the basis of *a priori knowledge*.

We perceive and rate intelligence mainly based on behaviour. But what is, actually, behaviour? If we define the set of all activities of a given person or object as behaviour, it is possible to distinguish between two main behaviour types:

- c) *passive behaviour*: reactions to external influences (coming from the environment or from other objects);
- d) *proactive behaviour*: actions, initiated for some internal reasons and having – intended or not – impact on other objects (*i.e.*, causing their reactions);

Of course, a mix of both above-mentioned types is also possible and can be referred to as either *mixed behaviour* or simply *behaviour*.

Strictly speaking, it is almost impossible to meet pure proactivity as defined above. In practice, when we say about a human that he is proactive we usually mean that he shows initiative. In turn, taking the initiative from an individual can be viewed as attempt on his part to predict or anticipate the next action towards himself and prepare for it or even initiate the appropriate reaction in advance.

Since any reaction can cause another reaction, often one simple activity initiates a chain of interlinked actions and reactions – *i.e.*, interactions. From the point of view of an external observer, the life of any object can be viewed as a chain of interactions with its environment. Note that according to the definition above, activities having negligible or no influence on other objects (*e.g.*, breathing, digesting or even thinking itself), are not considered proactive. Both thinking and reasoning are themselves activities, internal for every individual. Therefore we can rate them and estimate how intelligent they are only after the result of the reasoning is communicated to us by some activity – at least by speaking.

Now let us turn to the software. On the one hand, it is obvious that any software possesses some (kind of) behaviour. On the other hand, the possessing/exposing behaviour alone is insufficient for being intelligent. In regard to humans we would say that somebody's behaviour is intelligent after we compare it with either another person's or with our own (supposed or real) behaviour in a similar situation. Thus, we can state that a) there is no absolute intelligence and b) *intelligence is relative* and can be discovered only in comparison with something. In regard to software, usually similar reasoning is applied: when we say that a program is intelligent, we mean that in a given situation it either behaves better than most programs with similar purpose, or attempts to behave like a human being who experiences a similar situation. The most well-known test in this area is the test proposed by Alan Turing and named after him Turing (1950). This test is based on a chain of questions and answers (interactions). It should help one to decide whether a given machine or program can think and, consequently, can be considered intelligent. Up to now, no computer/program has passed this test; why should we then discuss intelligence of a software systems? The point is that on the one hand, the Turing test gives only a binary answer whether a given computer (system) is intelligent; on the other hand, each travel begins with the first step and we have to do it, if we want to reach the destination. Therefore, we need some other measure of intelligence in the meantime until intelligent machines become available. For engineering purposes we do not have to start with fully intelligent machines.

Anything more than “not intelligent at all” can lead to improvements and savings in the respective area.

We shall view a product or process model as being intelligent if it possesses at least one of the following capabilities:

- e) to behave/react as the modellee or simulate its reaction when the user simulates acting on it;
- f) to guess what (re)action is desired/needed in any given moment and either propose alternatives or perform it immediately (*cf.* the description of initiative by human's behaviour above);
- g) to find, determine or request any missing data or information alone;
- h) to recognize invalid data or information and correct it alone.

The more of these capabilities a given model possesses, the more intelligent it is considered. On the other hand, of two models, the one that can complete more tasks with less effort of the controlling/requesting user or program is considered more intelligent. The effort could be measured either as number of necessary instructions or as the time spent to give them.

Apparently, neither models drawn on paper, nor models made of clay or other workable materials can be given behaviour and, therefore, they cannot become intelligent. The only model type to which intelligence can be granted is the software model.

In general, proactive behaviour would require more intelligence from a model than passive behaviour, since the initiative implies own desires resulting from thinking or reasoning. Computer science, though, is not expected to achieve such advances in the next decade. Therefore, we concentrate on passive behaviour, since it is simpler to implement. But as mentioned above, passive behaviour alone is not sufficient to achieve intelligence. Turing (1950) says:

Intelligent behaviour presumably consists in a departure from the completely disciplined behaviour involved in computation, but a rather slight one, which does not give rise to random behaviour, or to pointless repetitive loops.

Two types of passive behaviour are distinguished: reactions to commands (leading to a desired result that is known in advance) and reactions to other events (all actions without commands). It is clear that reactions to commands (“disciplined behaviour”), which is immanent for every software product, cannot represent intelligence.

2.4.2.3.1.1.7 *Wisdom*

Wisdom is empirical information (experience), complementing the available knowledge and intelligence and making possible or increasing the probability to take proper decisions in yet unknown or not explicitly foreseen situations.

2.4.2.3.1.2 *Models for Representation of Data and its Derivatives*

We have defined data as strings of symbols, and each symbol is represented in the computer as an integer number. There are no (more) problems with representing symbols as numbers after the adoption of the Unicode standard, since the number of symbols used is finite. But how should the numbers themselves be represented when their number is infinite and no computer has infinite memory? Each pupil knows that the number of different fractions is infinite even in small ranges like

this between 0 and 1. Then, how is the finite number of symbols, plus the infinite quantity of numbers, represented in the finite computer memory? The answer is simple: by analysing the different application areas, and using a model of the respective range of numbers, which is appropriate for the given purpose. These models are called *data types* and have the inherencies illustrated in Figure 2.35.

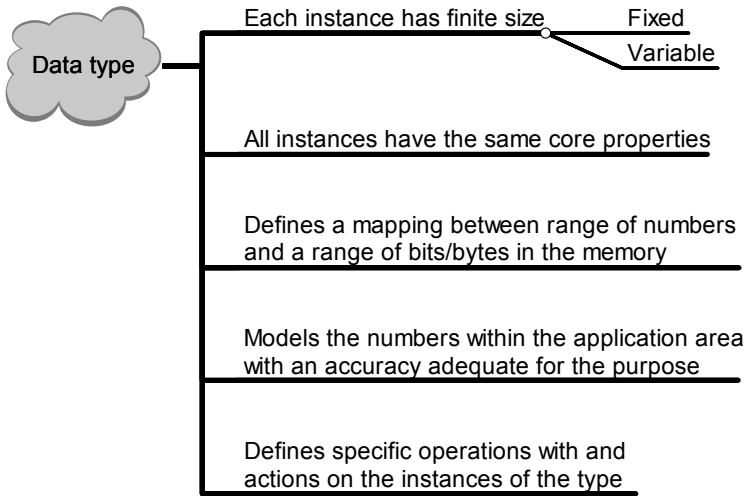


Figure 2.35. Inherencies of a data type

The simplest data type is dedicated to representation of Boolean values (true and false). Since there are only two possibilities, a single bit¹⁴ would be sufficient, but due to technical considerations, usually a whole byte¹⁵ is used. The representation of a finite set of consecutive numbers is commonly achieved by means of the data type *integer* or its derivatives. This data type has a typical size of 2, 4 or 8 bytes, depending on the implementation. The number N of consecutive numbers in the representable range R is directly dependent on the size S of the data type (in bytes) in the specific case, and is calculated with the following formula:

$$N = 2^{S*8} \quad (2.13)$$

Thus, with an integer data type variable of size one byte we could theoretically represent 256 consecutive numbers. If the numbers to be represented have a sign (*i.e.* the range is zero-symmetric), one bit has to be reserved for the sign representation. In this case, since the zero is “sign-neutral”, *i.e.* can have both signs, the integer data type should either be able to represent +0 and −0, which is redundant. So one number less can be represented, or the implementation should perform some checks for representing +0 and −0 as the same state of bits in order

¹⁴ A *bit* (abbreviated b) is the smallest information unit used in computing and information theory. It can be either one or zero and thus can represent any two mutually exclusive values or states like true or false, “on” or “off”, *etc.*

¹⁵ One byte (abbreviated B) has eight bits. Often used with prefixes like kilo, mega, giga, *etc.*

to avoid the reduction of the amount of representable numbers by one. The size of this data type is typically restricted to 8 (sometimes also to 10 or more) bytes, and thus the largest representable symmetrical range R is

$$R \in [-2^{8*8}, 2^{8*8}] \text{ or } R \in [-2^{64}, 2^{64}] \quad (2.14)$$

or

$$R \in [-18446744073709551616, 18446744073709551616] \quad (2.15)$$

Although fairly large, this range could be insufficient for some applications. So, to represent either larger ranges or rational numbers with a finite number of bits, another data type is used: *floating-point number*. Assuming that we reserve one bit for the sign S , p bits for the significand¹⁶, representing the most-significant digits of the number, and e bits for the exponent, the mathematical model of the range of numbers R representable by floating-point data type with $size=1+p+e$ bits will be as follows (all numbers in decimal base!):

$$R \in [-1^1 * 2^p * 2^{2^e}, -1^0 * 2^p * 2^{2^e}] \quad (2.16)$$

Here the significand is represented by 2^p , the factor 2^e is the exponent and the base is 2. The symbol \in in the last expression should be interpreted with a “restriction”: since p and e are integer numbers, not all numbers within the given interval are representable (e.g., no number between 2^2 and 2^3 is representable and exponentiation is used in the representation of both the significand and the exponent), therefore R is only defined through the representable numbers. Thus, only a small subset of the real numbers can be represented in a (digital!) computer exactly, and the rest are represented as the nearest rational numbers. A more detailed description of the floating-point representation and its problems would go beyond the scope of this work, but can be found, e.g., in Goldberg (1991).

2.4.2.3.1.3 Data-derivatives in Software Models

The software models are built-up from data, data-derivatives and (possibly) code. Therefore, many of the model traits depend on the traits of the underlying data and its derivatives, as well as on the chosen representation. For that reason, it should be kept in mind that most of the properties of software models are represented through real numbers, and when these numbers are approximated in their computer representation, the respective models could be badly influenced. Software models can use additional data for specialization (concretization) and communication. Software models can use bound or built-in code (as a special kind of data) for implementing intelligence.

2.4.2.3.2 Relations to other Terms and Software and Hardware Components

The information technology (IT) has huge influence on all computerized production methods. Since the rapid IT developments during the last decade have introduced numerous novelties and respective new terminology, let us consider some definitions and assumptions that would facilitate the further discussion.

¹⁶ According to Goldberg (1991), “This term was introduced by Forsythe and Moler [1967], and has generally replaced the older term *mantissa*.”

2.4.2.3.2.1 Platform

The term *platform* denotes the hardware and software used as a basis for either development or use of a given software model. When a model is used on a platform that differs from the platform it has been developed on, we speak about *cross-platform development*.

2.4.2.3.2.2 Host

Since the platform for the use of one particular software model may differ from the platform for its development, the possibility to distinguish between these two platforms is crucial. The platform where a given model can be used (or where the model can “live”) is referred to as *host*. The more platforms that can be used as hosts of a given model, the more portable or *platform-independent* this model is. The lower the number of components or layers required for the host, the higher the autonomy¹⁷ of the respective model.

2.4.2.3.2.3 System

The term system is overloaded with different meanings related to different areas of the science. A definition of this term that resembles our understanding closely enough is given in Wikipedia (*cf.* Wiki (2006)): “*A system is an assemblage of inter-related elements comprising a unified whole. From the Latin and Greek, the term “system” meant to combine, to set up, to place together*”. For our engineering purposes this definition has to be slightly modified in order to reflect the specifics of the majority of systems – both models and modellees – in the field of engineering.

Definition 2.11: A system is an assembly of inter-related components (subsystems, modules or elements) built together in a unified whole to serve a certain purpose.

In this sense, any compound model is also a system.

The purpose of a system together with the art of the components and their connections and relations determine most of the system's properties. A simplified model of a system, based on its most important properties, is presented in Figure 2.36. A taxonomy of some more important system-related attributes, terms and activities is presented in Figure 2.37.

2.4.2.3.2.4 System Interfaces

A system is typically connected to the outside world through *interfaces* – the set of all discernable input and output “channels” of the system that ensure cross-boundary communication with the outside world (*cf.* Figure 2.36). Apart from systems, all subsystems, modules and other components have interfaces, too. A system of hardware components has *hardware interfaces*, while a system of software components has, respectively, *software interfaces*.

Since the software “lives” in hardware (*cf.* Section 2.4.2.3.2.2 above), the properties of all software components – including interfaces – are strongly influenced by the properties of the underlying hardware. Software models either “live” in software systems or form themselves *systems of software models*.

¹⁷ *Cf.* the definition in Section 2.4.1.7.

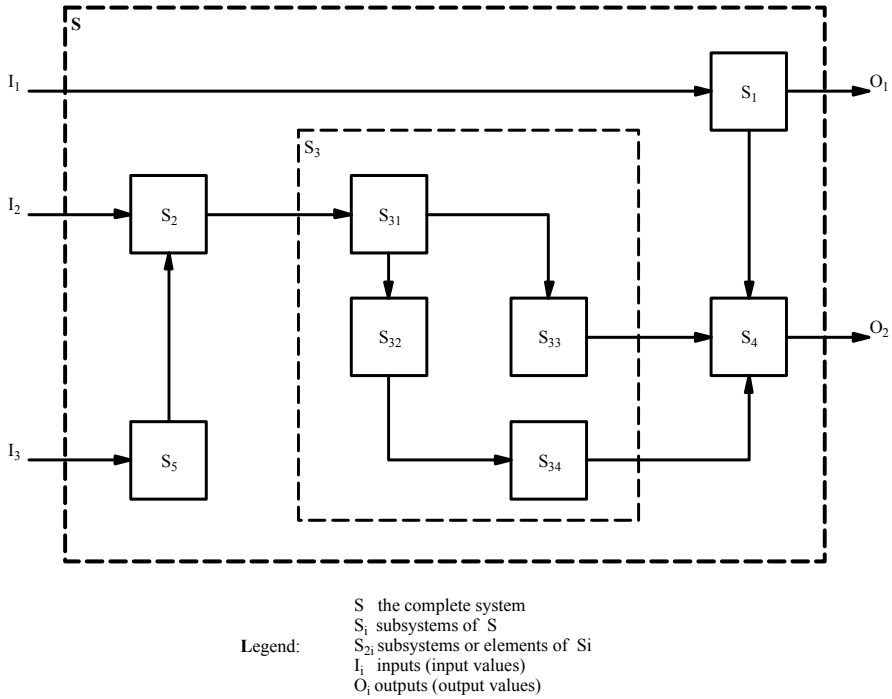


Figure 2.36. A simplified model of a system, advanced after Pahl and Beitz (1993)

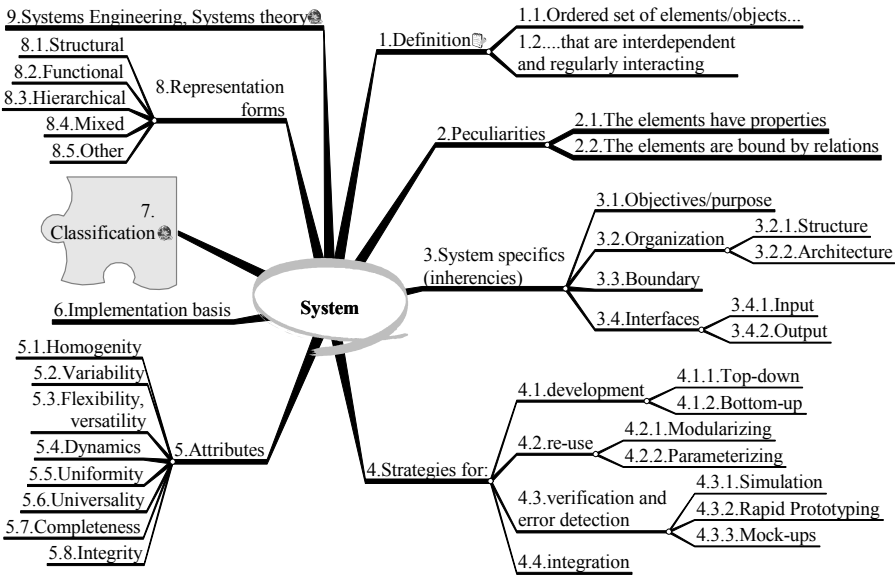


Figure 2.37. A taxonomy of some system-related attributes, terms and activities

In the domain of software programs or models, according to OMG¹⁸ or Booch *et al.* (1999), “every interface should represent a seam in the system, separating specification from implementation”. Let us call it *OMG-interface* to distinguish it from its typical meaning in CAX-context.

A system that possessing interfaces is sometimes called an *open system*. Theoretically it is also possible to have the opposite type of system – a *closed system*, but since such systems cannot communicate with the outside world, they are not of interest for our study (except if we are inside such a system), and are not discussed further.

The systems can be classified according to different criteria. An example classification is given in Figure 2.39.

2.4.2.3.2.5 Systems Engineering

Systems engineering (also known as *systems design engineering*) is a relatively new (originating around the time of World War II) branch of the science with focus on the definition, realization and characterization of complex, but at the same time successful systems. Some of the well-known subfields of systems engineering are safety engineering, reliability engineering, interface design, cognitive systems engineering, communication protocols, security engineering.

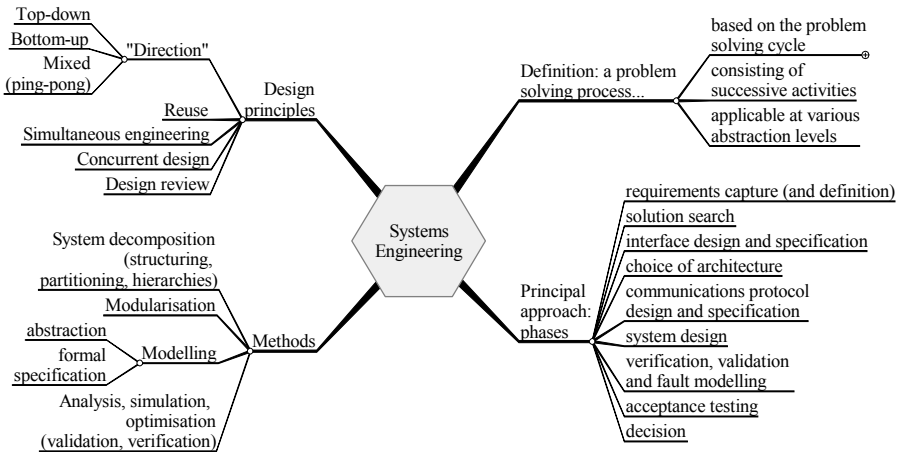


Figure 2.38. Inherencies of the systems engineering

2.4.2.3.2.6 Software Model vs. Computer Model

The terms software models and computer models are often used. Before we make use of them, we shall clarify what is similar and what is different between them.

A *software model* is an implementation of an information model (*cf.* Figure 2.5). It is a kind of representation (or mapping) of the information model by means of data structures and algorithms. Typically some formal languages are used to code the algorithms and the respective data structures into *programs*.

¹⁸ Object Management Group, Inc. *Cf.* <http://www.omg.org/>

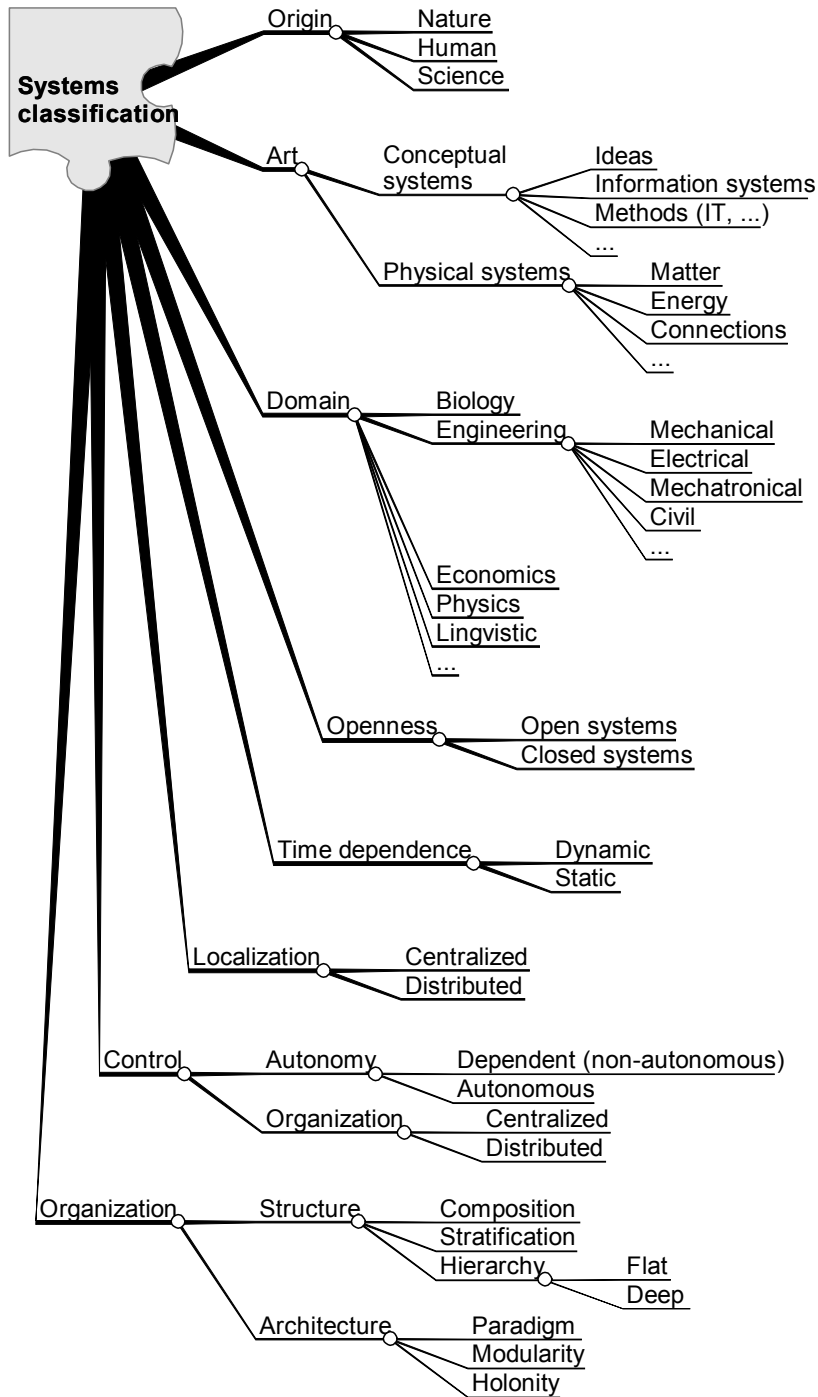


Figure 2.39. Sample classification of systems

Therefore, any software model can be viewed as a set of programs and data. Software models can be kept or transported on a medium, but the medium alone is not enough to allow the model to “come to life”. This can be achieved only when the model is loaded in some hardware, and the control is transferred to it. Thus, we can say that the model “lives” in its host – typically, a computer.

Without software no computer can directly be used to represent anything. When we say that something is “modelled with computer” we mean: “modelled by means of software running on a computer”. Thus, a *computer model* is nothing else than a software model that is loaded into a computer and activated there. In other words, both terms can be viewed as synonyms referring to a system consisting of hardware, programs and data. The main difference is that each of the terms stresses a different aspect of this kind of modelling, *e.g.* the use of software or the use of hardware, with the respective specificities.

2.4.2.3.2.7 *Computer Aided Systems*

A Computer Aided system or in short *CA-system* (also spelled without dash) is a complex software system, dedicated to solving tasks in a specific subject area. The subject area is typically a phase of the product lifecycle (design, planning, manufacturing, marketing, *etc.*) or an activity existing in many phases (*e.g.*, quality control, product-data management) and its name is usually reflected in both the long and the short forms (*e.g.*, Computer Aided Manufacturing system or CAM system). The alternative short form *CAX-system* is often used as a collective name for all possible short forms, where “x” is a placeholder, matching the name of any phase or activity. The “computer aided” is not really an obligatory part of the name, since it is implied for numerous activities. Thus, nobody would speak about word-processing without some kind of computer, but the respective “computer aided” system – the word processor or word-processing system – meets the definition and should be considered as belonging to the group of CAX-systems, too.

When several CAX-systems are used to automate related activities and are developed from the same producer they are often referred to as *software packages*, *software packets* or *suites*.

2.4.2.3.2.7.1 *CAX-model*

The usage of CAX-systems has become so common during recent decades that many people tend to forget: most CAX-systems create as one of their outputs a model (CAD-model, DMU-model, FEM-model, *etc.*). This model is often either the most important or the only result produced.

Some of these CAX-models are product models, some of them are object models (*i.e.*, something that is not going to be produced, but is used as part of other models) and some are models of processes. Therefore, the term CAX-model is used in the text as a generic term, referring to models of any of the types mentioned.

2.4.2.3.2.7.2 *CAX-system Centred Approach*

For each product several different product models, related to different phases or aspects of its lifecycle, are created and used. Typically, the model related to a given phase or aspect is prepared by a dedicated (CAX) system and can be modified and further developed only by *identical* (*i.e.* having the same type/dedication *and* from the same producer) or *compatible* (*i.e.* capable to read the models in their initial format) system. Even more important is that the models,

created from a CAX-system, can be used mainly *within* the system-creator, sometimes – *within* another (foreign) system and almost never – *alone*. In other words, any model “lives” only within a given CAX-system, and it in turn “lives” only within the respective hardware. For these reasons I shall use the name *system centred approach* (in short, *SCA*). The system where a specific model lives is called its *host* or *host system*.

In contrast to the object-oriented approach, where everything is centred around the concept of an *object*, and where the objects’ *methods* (algorithmic description of operations on objects) are typically defined only on the objects of the same class (*i.e.* type), CAX-systems operate on *diverse types of objects* but can perform only a given *group* (or *class*) of operations¹⁹. Thus, a general term for referring to all types of CAX-systems together could be “operation-oriented system”. Since in the computerized support of production we more often speak about classes of operations having to be performed than about classes of objects having to be processed, we have one more reason to say that conventional computer aided production is *system centred*.

2.4.2.3.2.8 Data Integration

The process of data integration includes collecting all data “pieces”, putting them in the same (or in compatible) format and possibly performing other actions to ensure that they are usable together.

2.4.2.3.2.8.1 Data Transfer, Data Exchange

The term data transfer/exchange refers to all actions that have to be performed in order to make a model created or existing on a given platform, work on another platform. These actions include the physical transfer of the model and possibly conversions (translation) on different levels. One could speak of unidirectional and bi-directional exchange, as well as exchange among multiple platforms. Typically, data exchange uses a file as entity, is unidirectional (*i.e.* it is transfer and not real exchange!), occurs offline and not so often as the data processing itself.

2.4.2.3.2.8.1.1 Models of the Data Exchange and their Qualities

As any other process, the process of data transfer or exchange can also be modelled. The resulting models can be divided into two main groups, aiming either at process development and realization or at its analysis and possibly – requests for improvement. A short classification of data exchange is given in Figure 2.40; more different models and a discussion of their qualities can be found in Avgoustinov (1997).

2.4.2.3.2.8.1.2 “Interface Pressure”

Suppose that a “force” called *demand for data exchange* exists, and that the *descriptive potential*, defined in Avgoustinov (1997) as the cardinality of the set of elements of the source language, symbolizes the “area” on which the force is applied. Then we could use the metaphor “interface pressure”, which (exactly as normal pressure) is proportional to the force and inversely proportional to the area.

¹⁹ This does not mean that the object-oriented approach is not used in CAX-systems; it is simply applied on a different (lower) level.

Similarly, it is possible to say that the target system has “*interface resistance*” that is also proportional to the demand for exchange, and inversely proportional to that part of the descriptive potential which is utilized in the models to be transferred.

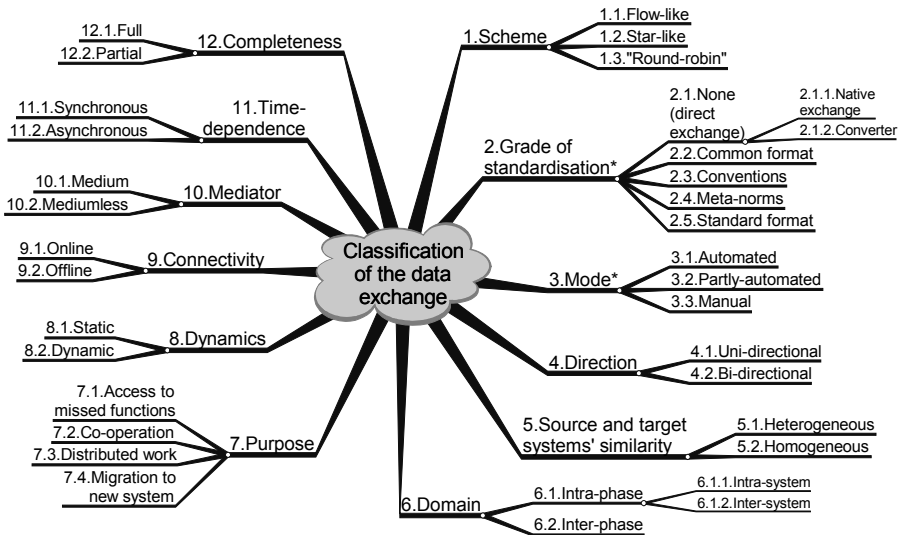


Figure 2.40. An example classification of the data exchange. Reworked and extended, after Avgoustinov (1997)²⁰

2.4.2.3.2.8.2 Data Sharing

Data sharing is a general term used to denote the process of making data available to more than one user or system. In contrast to data exchange there is no typical entity. Only needed data is accessed and the sharing occurs on demand, online, multidirectional and even multiple times per processing. Since typically only very small parts of the source model are needed, accessing only them and only on demand makes the process much more efficient than data exchange.

2.4.2.3.2.9 Component

We shall refer to any at least logically separable part of a model, product, system or any other compound object as a *component*. When a component is not compound, we can call it also an *element*. In the domain of software programs or models, according to OMG as in Booch *et al.* (1999), a component is “*A physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.*”

²⁰ Slightly different notation is used in the cited publication for the two terms, denoted “*” in Figure 2.40, but the actualized notation (as given in the figure) seem more adequate to me.

2.4.2.3.2.9.1 Integration of Components

On the one hand, components (both software and physical) are designed to be integrated into systems and also to be interchangeable. On the other hand, they cannot be combined arbitrarily. The process of achieving effective interaction (including communication) among components, as well as interaction of the user with these components independently of their location, host, state, *etc.*, is called in this study *component integration*.

2.4.2.3.2.9.2 Component-based CAX-system

Any CAX-system that is built up from software components is called *component-based CAX-system*. According to Kilb and Arnold (1998) “... using a system based on the CAX object bus, there is no longer a need for file based data transfer between different integrated systems. Only the necessary representation information of the data models has to be transmitted as CAX objects through the CAX object bus.” The problem here is that the “necessary representation information” has not only to be transmitted but – depending on the case – also converted to the respective format!

2.4.2.3.3 Relation to Suitability, Relevance, Adequacy, Reusability

The components and their organization and granularity have a strong influence on many traits of the system that contains them. Below are enumerated some relations of the components (or componentization) to some of these traits or to other notions of interest; in all formulas $f(\dots)$ symbolizes a function of something, whereas F denotes some kind of functionality (*e.g.*, a set of functions), explained by an appropriate subscript; after some formulas are given their ranges.

$$\text{Suitability}=f(\text{form, granularity}) \quad (2.17)$$

If all functions composing the functionality are equally important and their implementation is either full or null, suitability can be expressed as the ratio between the cardinalities of the set of (fully) covered functions and the set of needed functions.

$$\text{Suitability}=|F_{\text{covered}}|/|F_{\text{needed}}| \quad [0, 1] \quad (2.18)$$

Since usually some functions are more important than others, the above formula should be extended with weight factors for each functions. To keep results within a normalized range (*i.e.*, within $[0,1]$) the sum of all weight factors should be equal to 1 (or 100%).

$$\text{Suitability}=f((\text{coverage} - \text{needs})/\text{needs}) \quad [0, 1] \quad (2.19)$$

The gaps between the granules (*i.e.*, the uncovered areas) decrease the suitability and together with the excess functionality, form the inefficiency of the system.

$$\text{Adequacy}=f(\text{overlaps, shortage, excess}) \quad [0, 1] \quad (2.20)$$

It can be reasonable to use the inadequacy instead:

$$\text{Inadequacy}=f((\text{overlaps} + \text{shortage} + \text{excess}) / \text{needs}) \quad (2.21)$$

The reserve (or spare) functionality can be expressed as follows:

$$F_{\text{Reserve}} = F_{\text{total}} - F_{\text{needed}} \quad (2.22)$$

Analogously, the redundant (or excess) functionality can be expressed as a function of the doubled (or overlaped) in different components functionality:

$$F_{\text{redundant}} = f(F_{\text{overlaped}}, F_{\text{needed}}) \quad (2.23)$$

2.5 Model Representation

After its elaboration, each idea has to be represented somehow. The representation allows us to communicate the model to others, to save it for later use and even to discover things or relations among them that were invisible or not obvious before. When the modellee is not an idea, but something really existing, the situation is not very different: the main difference is that the modeller first “gets the idea” and then prepares a representation, reflecting the most important and relevant-for-the-purpose properties of the modellee.

2.5.1 Reasons for Discussing the Representation

Apart from the fact that for software models the representation has great influence on the efficiency, compactness and other characteristics of the model, there are other reasons for discussing the models' representation:

- very often models are mixed up with one of their representations;
- improper representation could lead to confusion or loss of information;
- some models represent not a single entity/modellee, but a whole class of similar entities (modellees); we shall call such models *parameterized*;
- the model's representation is non-abstract (real) in contrast to any software model itself.

Very often models have more than one representation – *e.g.*, if the same model were drawn twice, but in two different colours, we would get two representations of the same model. But we should not confuse different models of the same modellee with different representations of the same model: if we represent some modellee once as a text and once as a drawing, these would be two different models of the same modellee.

On the other hand, some models can have different views or aspects – for instance, a three-dimensional (3D) model can be viewed from different viewpoints; although each viewpoint can be represented on paper as a 2D drawing or snapshot, they remain different 2D representations of the same 3D model.

2.5.2 Classification of the Representation Types

The representation of a model can depend on many different things – medium, method, changeability, dynamics and others. The representation can even change during the model's use. Since each representation type has its advantages and disadvantages, the modeller can choose the type most appropriate for the purpose. Therefore, we say that each representation is purpose-dependent. Sometimes –

especially with respect to multi-purpose models – the modeller can choose to provide several representations, so that the user of the model is able to choose the most appropriate for the moment or for the respective task representation.

We shall distinguish between two different types of model representation: internal and external.

The *internal representation* concerns how a given model is represented within a software system or within a computer, and is implementation dependent. For instance, the internal representation of the simplified model of a circle from Figure 2.17 can be as short as three numbers, connected with the knowledge that they represent the radius and the coordinates of the centre point, respectively.

The *external* (or *observable*) representation of a software model is usually a dynamic representation, depending on the values of the model data at the moment. Figure 2.17 itself is an external representation of the (parameterized) circle. It should be noted that the external representation is usually based on the internal one.

Within the internal representation we distinguish between *model data* (or parameters) and *model invariance* or *model knowledge*.

The model data is different among the instances of the modelled class of entities and is used to create distinguishable representatives of the class. It is almost always included in the model saved on a medium to guarantee its persistence.

Model invariance can be of two subtypes: *programs* and *metadata*. A program can be viewed as data, describing one or more algorithms. It can describe operations on real data or on placeholders. At runtime the placeholders are replaced with the actual values of the parameters.

The metadata describes the relation among the data elements at the lowest level (the parameters) and is usually implemented by means of data structures.

On the next level the relations among the metadata can be described by means of meta-metadata – cf. Figure 2.17 once more. Since we can always describe the relations among elements of one level by means of *meta^x-data* on the next higher level, we can speak about *metadata of different degree* (cf. Section 2.4.2.1.2.3.1 Levels of Hierarchical Structure above).

2.6 Integration of Models

The majority of devices, machines and other products are actually complex systems built up from separately produced components. These components can be of pure mechanical, electrical, or electronic nature, or they can also be intermixed. When users observe and use them as a whole – *i.e.* the product – there is no need to speak about integration from the user's point of view. From the manufacturer's point of view, however, all components have to be assembled or built together; this process can be viewed as integration. Therefore, when a compound product is modelled, depending on the purpose of the model, it could be natural and useful first to model each component alone and after that to integrate all these models in a compound model of the product. But what is integration? According to Lutters (2001), there are countless definitions of integration. One more reason to define again what should be understood under integration in the present work is that none of the known definitions is perfect, including this in Lutters (2001): “*the*

facilitation of mutual cooperation and interaction between distinct functions in the manufacturing environment”. Weak points in the last definition are the lack of an aim in the definition and the word “facilitation”; a better attempt would be to use “accomplishing” instead. Our approximation for a more general definition could be the following:

Definition 2.12: The integration of two or more (manufacturing) components is the process of making them work on one and the same task or contribute to achieving one and the same outcome.

How exactly integration will be achieved – whether the components will be “physically joined” or “obey the same control”, or just the results of their work will be joined – is a question of secondary importance. In other words, the integration is not an end in itself: either the result, achieved after the integration of two models, is better than the sum of the results of the two non-integrated models, or such a result cannot be achieved at all without integration.

In the simplest case integration of two models would be to achieve their simultaneous use within (or from within) the same environment (hardware, operating system, application software, *etc.*). For more sophisticated dynamic systems, though, making the communication between the models involved and the interaction (of the user) with them effective, independently of model location, host, state, *etc.*, is also indispensable.

2.6.1 Integration Classification

The integration can be classified according to different criteria. Perhaps the most important criterion is the type of components that have to be integrated – real or virtual, material or abstract, *etc.* – since this can affect most of the other criteria. It is apparent that the method for integration of the components of a real car will be different from the methods for integration of the models of the same car components.

The technique used for achieving the integration can serve also as a characteristic of classification of integration techniques. An example is given in Figure 2.41.

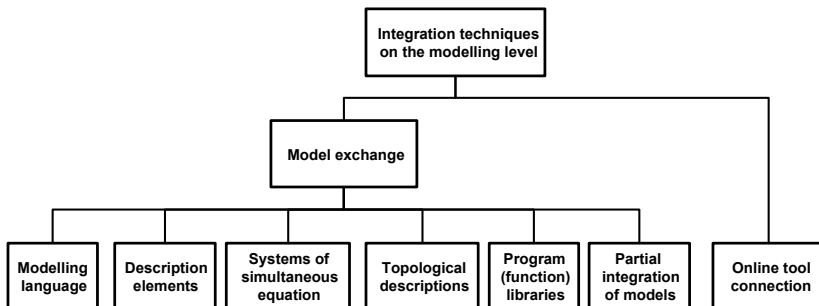


Figure 2.41. Some integration techniques, after Gausemeier and Lückel (2000)

A sample classification according to several additional criteria (inherent integration traits) is given in Figure 2.42.

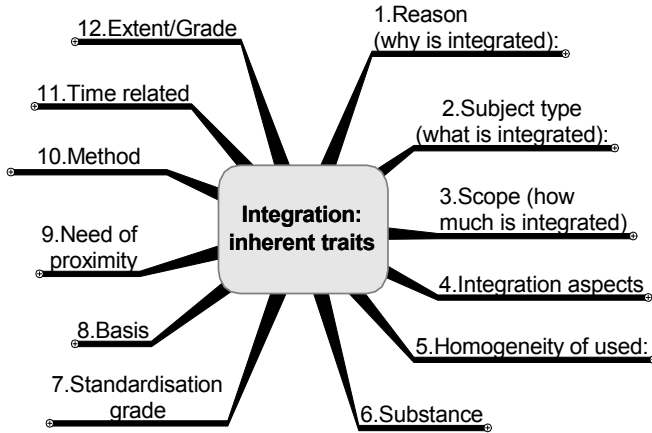


Figure 2.42. Example for a possible classification of the integration

2.6.2 Models, (Software) Applications and their Integration

Computer programs (a.k.a. software programs) that aim to solve user-specific (and not system-specific!) problems are often called *software applications*. They reside on top of the operating system and are very often used to:

14. control some local-computer-related process like printing, visualization, communication, *etc.*;
15. control some process that is not related to the host computer, but to business or some other part of real life – production, transportation, commerce, *etc.*;
16. model and simulate different real processes;
17. support various processes otherwise.

Exactly like many other things that have structure or are part of a structure, applications may need to be integrated. Even if we consider only case 16 above, models of sub-processes have to be integrated to achieve a simulation of the full process that is modelled. In the case of modelling, simulation or control of complex processes it can be necessary to integrate many applications of different type, different origin, and different sites within a given enterprise and even located in different enterprises. In similar cases an often used term is *enterprise application integration*. One of the popular definitions is given in Wikipedia Wiki (2006):

Enterprise application integration (EAI) is the use of software and architectural principles to bring together (integrate) a set of enterprise computer applications. It is an area of computer systems architecture that gained wide recognition from about 2004 onwards. EAI is related to middleware technologies such as message-oriented middleware MOM, and data representation technologies such as XML. Newer EAI technologies involve using web services as part of service-oriented architecture as a means of integration.

Achieving integration – or even better, integrability – is often more important than achieving good coverage or good functionality. Again in Wikipedia Wiki (2006), the following is said about the role of the integration:

Without integration, enterprise computing often takes the form of islands of automation, where the value of individual systems is not maximised because they are working in partial or full isolation. However, if integration is carried out without following a structured EAI approach, many point-to-point connections grow up across an organization. Dependencies are added on an ad hoc basis, resulting in a tangled non-maintainable mess, commonly referred to as spaghetti.

...

EAI is not just about sharing data between applications. EAI focuses on sharing both business data and business process.

Many different integration approaches have been developed and tested, achieving great or small success, but none of them has been generally accepted. One of the objectives of this book is to clarify the role of the modelling approach in achieving satisfactory model integration and therewith also better integration of the respective modellees. More details are discussed in the following chapters.

Modelling in Mechanical Engineering and Mechatronics
Towards Autonomous Intelligent Software Models

Avgoustinov, N.

2007, XXIV, 226 p. 97 illus., Hardcover

ISBN: 978-1-84628-908-8