INTRODUCTION TO PHYSICAL MODELING WITH MODELICA

MICHAEL TILLER, PH.D.
Technical Specialist, Ford Motor Company
Member, Modelica Association

Kluwer Academic Publishers
Boston/Dordrecht/London

Preface

In writing this book, my goal is to demonstrate how easy, useful and fun, the modeling of physical systems can be. For me, there is nothing that a computer can be used for that is more interesting than simulating the behavior of physical systems. The term "physical systems" refers to the behavior of physics-based models found across many disciplines (*e.g.*, electrical engineering, mechanical engineering, chemistry, physics). Such systems can be identified by their use of conservation principles (*e.g.*, first law of thermodynamics and conservation of mass).

In this book I will describe how the Modelica modeling language can be used to describe the behavior of physical systems. Modelica can be used for a wide range of applications from simple systems with only a few degrees of freedom all the way up to complex systems made of large networks of reusable components.

The first part of the book is focused on introducing the reader to the Modelica modeling language. The target audience would be somebody with an understanding of basic physics and calculus, an interest in modeling and no knowledge of Modelica. The intent is to cover all the basics of the language using simple examples and enable the reader to begin writing models in Modelica.

Each chapter in the first part of the book starts with an overview of the important concepts the chapter introduces. Whenever a new term is introduced it will appear *italicized* and a definition for it will be included in the glossary. The overview is then followed by a series of examples meant to gradually introduce Modelica functionality. I feel that examples are an important part of the learning process. I have tried to avoid using contrived examples. In fact, many of the examples come from real world problems I have encountered. The difficulty with examples is that they do not introduce material in a structured way, but rather in a "flowing" way. For this reason, many chapters include a "Language Fundamentals" section which attempts to formalize all of the

concepts introduced by the examples. Readers may feel free to skip over the material in the fundamentals section if they feel comfortable with the features presented in that chapter. An important note about the structure of this book is that **each example introduces new concepts**. In other words, do not assume that because you understood the first example in a chapter all the remaining examples are not worth studying.

The second part of the book demonstrates how to most effectively use the powerful features of the Modelica language. This part is intended for people who are already familiar with the basics of the Modelica language, including existing users of Modelica and beginners who have completed the first part.

This book covers nearly all of the features of the Modelica language. However, much of this material is only required in advanced applications. The "core" material required to begin doing meaningful modeling can be found in Chapters 1, 2, 3 and 7. Readers may wish to focus their attention on those chapters first and then consult the other chapters as they become more proficient.

Realize that it is not possible to introduce every nuance of the Modelica language through examples. Once you have covered the material in this book, you will require a definitive reference. The ultimate source of information about Modelica is the language specification itself. For this reason, the Modelica language specification is included on the companion CD-ROM. While not appropriate for learning the language, it is appropriate as a reference on the semantics of the language.

In summary, this book includes material that will have broad appeal and will serve both beginners and experienced users trying to get the most out of physical system modeling.

MICHAEL TILLER

Chapter 1

INTRODUCTION

1.1 WHAT IS MODELICA?

Before diving into the details of modeling using Modelica, let me provide a brief description of what Modelica is, why it was developed and what it is used for.

Since the invention of the computer, modeling and simulation have been an important part of computing. Initially, modelers were burdened with converting their models into systems of ordinary differential equations (ODEs) and then writing code to integrate those differential equations in order to run simulations. Eventually, a wide range of integrators were developed as independent software units and modelers were able to focus on the formulation of differential equations and use "off-the-shelf" integrators for simulation. This trend of allowing modelers to focus more on the behavioral description of their problems and less on the solution methods has continued ever since.

In the last three decades, numerous tools have been developed to assist modelers in performing simulations. Some of these were general purpose simulation tools such as ACSL[1], Easy5[2], SystemBuild[3] and Simulink.[4] Other tools were developed for simulations in specific engineering domains such as electrical circuits (*e.g.,* Spice[5]), multi-body systems (*e.g.,* ADAMS[6]) or chemical processes (*e.g.,* ASPEN Plus[7]). Each type of tool has its advantages.

---

[1] ACSL is a trademark of The AEgis Technologies Group, Inc.
[2] Easy5 is a trademark of The Boeing Company.
[3] SystemBuild is a trademark of Wind River Systems, Inc.
[4] Simulink is a trademark of The MathWorks, Inc.
[5] Spice is a trademark of the University of California at Berkeley.
[6] ADAMS is a trademark of Mechanical Dynamics, Inc.
[7] ASPEN Plus is a trademark of Aspen Technologies, Inc.

For example, general purpose tools do not restrict modelers to a particular domain but they may require the modeler to spend some time formulating their models for that particular tool. Likewise, tools developed for a specific engineering domain have numerical methods and graphical user interfaces which are optimal for that particular domain but they restrict the ability of the modeler to create mixed-domain models.

In 1978, Hilding Elmqvist pioneered, as part of his Ph.D. thesis, a new approach to modeling physical systems by designing and implementing the Dymola modeling language (Elmqvist, 1978). The basic idea behind the Dymola modeling language was to use general equations, objects and connections to allow model developers to look at modeling from a physical perspective instead of a mathematical one.[8] For the Dymola implementation, graph theoretical and symbolic algorithms were introduced to transform the model to an appropriate form for numerical solvers. An important milestone in the development of this approach came in 1988 with the development of the Pantelides algorithm for DAE index reduction (Pantelides, 1988). Following Dymola, numerous other tools (*e.g.*, Omola, see Mattsson et al., 1993) were developed to further explore this new approach to modeling.

A major problem with all simulation tools has been that models developed using one tool could not be used by another. In 1996, Hilding Elmqvist initiated an effort to unify the splintered landscape of modeling languages by initiating the development of the Modelica modeling language. Similar initiatives have been undertaken by various other groups (see Heinkel et al., 2000 and Fitzpatrick and Miller, 1995) but these efforts have been focused primarily on the electrical domain, while Modelica strives to be completely domain neutral.

The basic idea behind Modelica was to create a modeling language that could express the behavior of models from a wide range of engineering domains without limiting those models to a particular commercial tool. In other words, Modelica is both a modeling language and a model exchange specification. To accomplish this goal, the developers of previous object-oriented modeling languages like Allan, Dymola, NMF, ObjectMath, Omola, SIDOPS+ and Smile were brought together with experts from many engineering domains to create the specification for the Modelica language based on their wide range of experiences.[9]

Modelica can be used to solve a variety of problems that can be expressed in terms of differential-algebraic equations (DAEs) describing the behavior of continuous variables. The ability to formulate problems as DAEs rather than ODEs reduces the burden on the model developer because less effort is

[8]The physical and mathematical approaches are contrasted in Chapter 11.

[9]A detailed history of how the Modelica modeling language was developed is contained in Appendix A.

involved in formulating equations. In addition to handling continuous variables, Modelica includes features for describing the behavior of discrete variables (*e.g.*, digital signals). Often, it is convenient or even necessary to simulate both continuous and discrete behavior at the same time. Modelica allows both forms of behavior to be described within the same system model or even the same *component* model.

Modelica is a non-proprietary modeling language and the name is a trademark of the Modelica Association which is responsible for publication of the Modelica language specification. At present, Modelica is not an ISO, ANSI or IEEE standard. This means that Modelica is presently a "moving target" in much the same way as *C++* was for about a decade. In the case of *C++*, avoiding the rush to standardize did not prevent people from making use of the language and ultimately led to a much better language. Hopefully, Modelica will follow a similar path. If a need can be demonstrated for functionality not already present in the Modelica language, users can work with the Modelica Association to fill functionality gaps. The current Modelica specification can be found at the Modelica Association web site: `http://www.modelica.org`. Version 1.4 of the Modelica specification is included on the companion CD-ROM.

If you have ever been involved in large scale modeling projects you probably recognize that model development is in many ways similar to large scale software development. Just like a programming language, the purpose of a modeling language is to describe the behavior of small pieces of a larger system. A modeling language should encourage reuse of previous work and help manage the complexity of systems as they become larger. It should be possible, once a reusable set of components has been created, to work at an increasingly higher level (*i.e.*, getting away from writing equations at the component level and working more on the assembly of a complex system). Ultimately, this leads to the ability to build systems using a "top-down" approach rather than a "bottom-up" approach.

All simulation results presented in this book were generated using Dymola (Dynamic Modeling Laboratory).[10] An evaluation copy of Dymola is provided by Dynasim (Elmqvist et al., 2001) on the companion CD-ROM so that readers may gain hands-on experience with using the Modelica language. To understand how to simulate Modelica models using Dymola, please read the documentation titled "Getting Started with Dymola" which is included with the Dymola software. Dymola can also be used to generate models that can be imported into Simulink.

[10]Dymola is a trademark of Dynasim AB.

1.2 WHAT CAN MODELICA BE USED FOR?

Modelica can be used for many things, including simulation of electrical circuits (Clauss et al., 2000), automotive powertrains (Otter et al., 2000), power system stability (Larsson, 2000), vehicle dynamics (Tiller et al., 2000) and hydraulic systems (Beater, 2000). However, the best way to understand what Modelica can be used for is through an example. While most of the chapters in the book use relatively simple examples to highlight specific language features, we will start by giving a glimpse of "the big picture".

In this section we will show bits and pieces of a substantial library of Modelica models for simulating automobile performance. The library was developed for this book to demonstrate how reasonably complex systems can be modeled. While the library contains a large number of models, most of the models are quite simple. Because these models are relatively simple, they will give us only a rough estimate of how particular automobile designs will perform. The Modelica models from this section are provided on the companion CD-ROM and discussed in greater detail in Chapter 10.

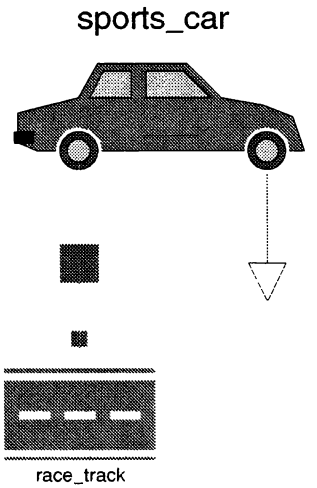

Figure 1.1. A 0-100 kilometer per hour test.

Imagine we wish to predict the acceleration performance for a particular sports car design. In order to judge the performance, we will measure the time it takes the vehicle to accelerate from zero to one hundred kilometers per hour. Figure 1.1 shows our performance test which includes a sports car and a race track.

Do not be fooled into thinking the model we are simulating is not detailed just because the picture looks simple. This is just the top-level view of the problem.

Figure 1.2 shows what we find if we look inside our sports car model. Behind the scenes, the sports car model includes models of the chassis, transmission and engine as well as a shifting strategy that decides when to change gears. Behind all of these images are behavioral models (*i.e.*, the images themselves are just used to help identify what the models represent). As we shall see, even this view of the sports car gives a deceptively simple impression.

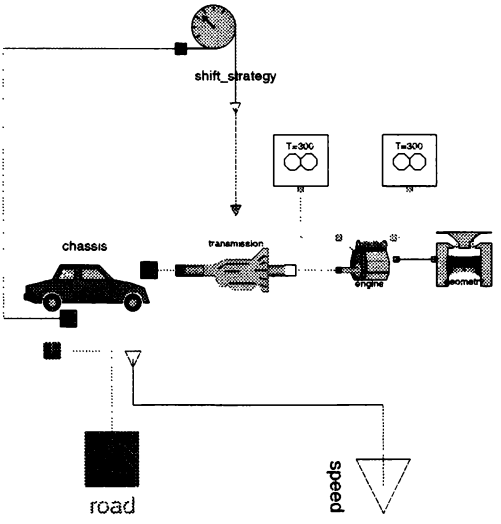

Figure 1.2. **Taking a look at what is under the hood.**

The engine model for our sports car is one of many components in Figure 1.2. If we open up the engine model we can see each of the four individual cylinders (shown in Figure 1.3). Again, the images of engine cylinders are graphics added to the models so they can be easily identified as engine cylinders. Behind each of these pictures is a detailed schematic of the components used to model an individual engine cylinder.

If we open up one of these cylinders, we find the numerous low-level component models shown in Figure 1.4. By zooming in to each of the various models shown so far, we have gone from the complete vehicle level (shown in Figure 1.1) all the way down to models of individual components such as engine valves (shown in Figure 1.4). The ability to construct such hierarchies is a central feature of Modelica. In addition, the ability to include graphical representations for the models, as we have seen in these figures, is also a feature provided by Modelica.

Each of the graphics shown in Figure 1.4 represents a component involved in the function of an individual engine cylinder. We cannot "zoom" into these

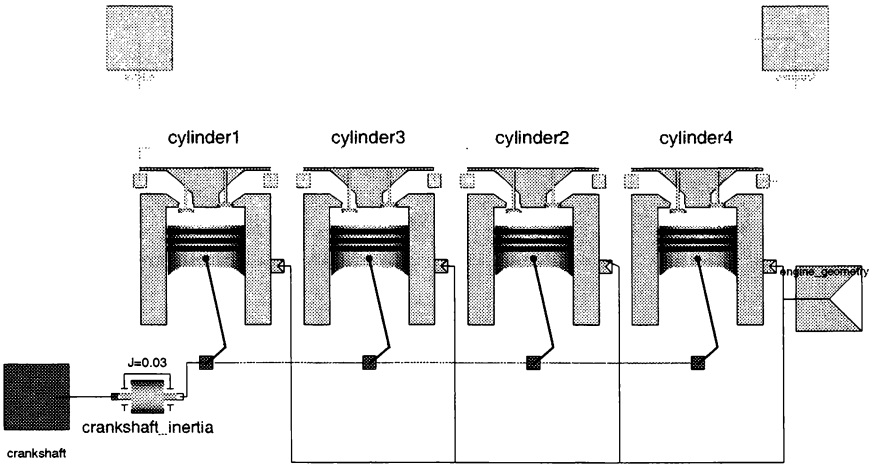

Figure 1.3. Looking inside the engine.

models because they represent the smallest pieces in the system. In a sense, they are the "atoms" of our system. It is important to understand that **these pieces are not magical primitives** that just happen to come with the software package we used to build this model. In fact, it is at this component level that we turn our attention away from all the graphics toward the real subject of this book: the Modelica modeling language. Previously, we have seen how the Modelica modeling language can be used to describe hierarchies of components. At the "atomic" level, it can also be used to describe the behavior of each of these components. The remainder of the book will provide all the necessary information to build such components and an enormous variety of other components in other engineering domains.

Building models is fun, but ultimately we want to see results from such models. When we run our simulation, we find that the sports car model presented in this section can go from zero to 100 kilometers per hour in 6.88 seconds. Figure 1.5 shows several different pieces of information recorded during the test. Notice how the transmission gear changes at different vehicle speeds. We can also see how the engine speed increases up until the transmission shifts and then it drops again. These are just a handful of signals we can extract from our simulation. Other useful pieces of information available include manifold pressure, trapped mass in the cylinder, traction force on the tires, transmission clutch pressures, *etc.* Studying such information can provide important insights during the design process.

Once we have a model that gives us good results, the next logical step is to ask ourselves "what if?". The sports car in our race model includes numerous

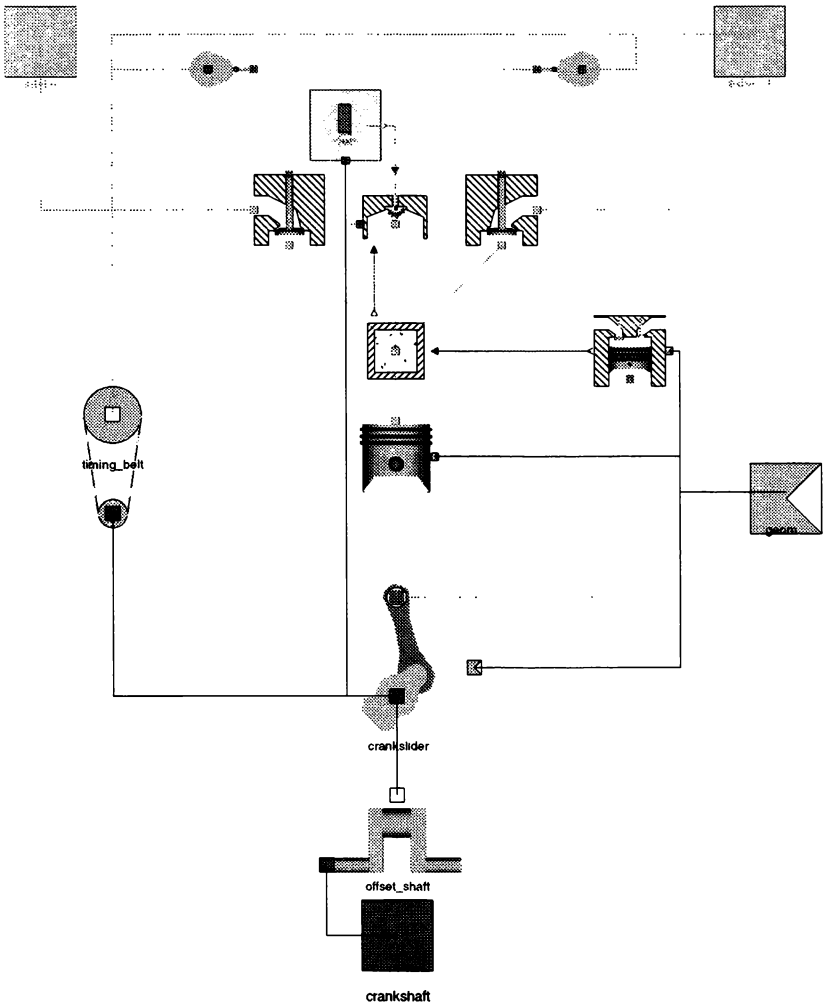

Figure 1.4. Looking inside an individual engine cylinder.

design details. For example, we can easily specify the engine geometry, valve timing, shift schedule, vehicle weight, tire radius, and so on. By changing these values, we can determine the impact each of these parameters has on overall system performance.

Remember, Modelica is a domain-neutral modeling language useful for creating models from nearly any engineering domain. The remainder of the book shows how models from many other engineering domains can be created using the Modelica modeling language.

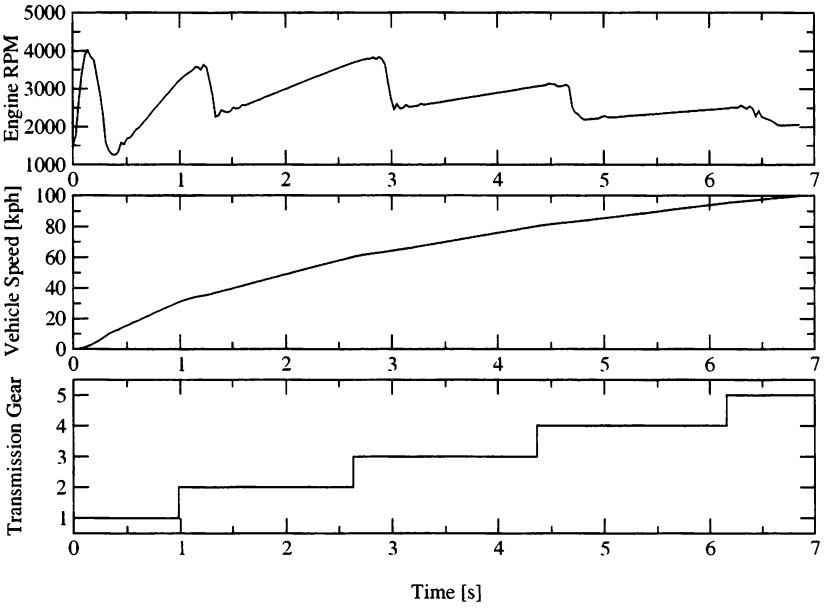

Figure 1.5. Simulation results from a sample race.

1.3 MODELING FORMALISMS

Before we start discussing how to use Modelica to develop models, let us take a moment to talk about modeling in general. There are many formalisms used for modeling continuous systems. An excellent overview of different formalisms is presented in Åström et al., 1998. Modelica supports two of the common approaches to modeling in engineering.[11] The first is called *block diagram* modeling and the other is called *acausal* modeling.[12] In this section we will discuss block diagrams and acausal formulations to better understand the differences between them.

1.3.1 Block diagrams

Using block diagrams, a system is described in terms of quantities that are known and quantities that are unknown. A block diagram consists of components, called blocks, which use the known quantities to compute the unknown quantities. A block diagram of a PI (proportional-integral) controller is shown in Figure 1.6.

[11] In addition, other formalisms like bond graphs (see Cellier, 1991) and petri nets can also be described in Modelica.

[12] Acausal modeling is sometimes referred to as *first principles* modeling.

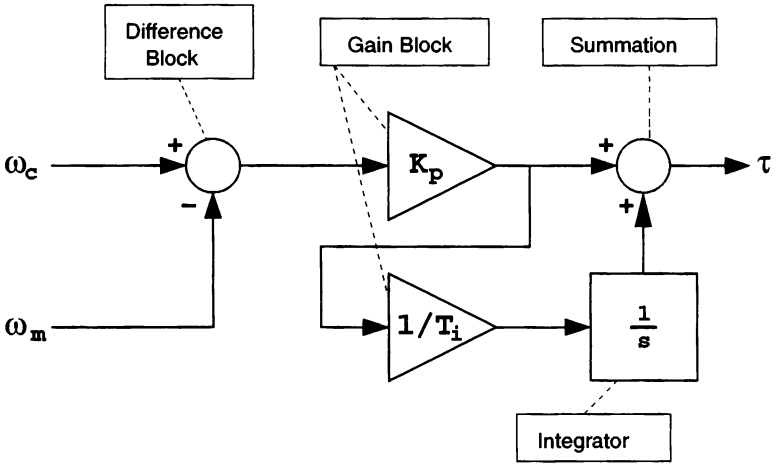

Figure 1.6. PI Controller.

On the left side of Figure 1.6 are the known quantities ω_c (the desired speed), and ω_m (the actual motor speed as read by the speed sensor). On the right side of Figure 1.6, the torque used to control the system is computed. In between are the *blocks* which describe the computations being performed. In this example, the difference block takes the desired and sensed speed as an input and computes as an output the difference (*i.e.*, the error). One gain block then multiplies the speed difference by the gain, K_p. The scaled speed difference is passed through another gain block, scaled by $\frac{1}{T_i}$ and integrated. We compute the control torque by summing the outputs from the gain blocks.

This approach to modeling is often used when designing control systems. For example, tools such as Simulink and SystemBuild use this approach. A block diagram is a natural way of expressing a control system design. However, such diagrams have their limitations as we shall demonstrate in Chapter 11.

1.3.2 Acausal modeling

Describing system or component behavior in terms of conservation laws is referred to as acausal modeling. With acausal formulations, there is no explicit specification of system inputs and outputs. Instead, the *constitutive equations* of components (*e.g.*, Ohm's law for a resistor) are combined with *conservation equations* to determine the overall system of equations to be solved. For example, when modeling electrical systems, like the circuit shown in Figure 1.7, one can use *Kirchhoff's current law* (a conservation law), which states that the sum of the currents into a particular node (in this case, *a*, *b* or *c*) must be zero. The application of conservation laws results, in general, in

systems of differential-algebraic equations (DAEs). Dymola and Saber[13] are two examples of tools that allow acausal formulations.

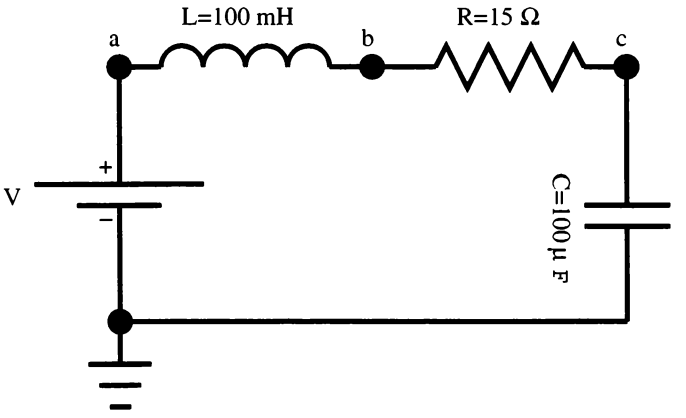

Figure 1.7. RLC circuit schematic.

In order to formulate acausal models, it is useful to identify the *through variables* and the *across variables* for the component being modeled. In general, the across variable represents the driving force in the system and the through variable represents the flow of some conserved quantity. For an electrical system, the voltage is the across variable and the current is the through variable. Note that the product of the through variable and the across variable typically has the units of power (*i.e.*, Watts in SI units). Table 1.1 includes several examples of through and across variables for different engineering domains.

Domain	*Through*	*Across*
Electrical	Current (A)	Voltage (V)
Mechanical (translational)	Force (N)	Velocity (m/s)
Mechanical (rotational)	Torque (Nm)	Angular Velocity (rad/s)
Hydraulic	Flow Rate (m^3/s)	Pressure (N/m^2)

Table 1.1. Through and across variables from various domains.

1.3.3 Further remarks on formalisms

As we shall demonstrate in Chapter 11, block diagrams are convenient for control system modeling and acausal formulations are convenient for physical

[13] Saber is a trademark of Avant! Corporation.

system modeling (*i.e.*, plant modeling). Not only does Modelica support both of these important types of modeling, but it allows both of them to be used together.

1.4 MODELICA STANDARD LIBRARY

In addition to defining the specification for the Modelica language, the Modelica Association also publishes a standard library of Modelica models. This library, called the Modelica Standard Library (or MSL), is available free of charge.[14]

The MSL was developed so that users of the Modelica language would not have to create their own basic models for the common modeling domains. Throughout this book, we start off by developing Modelica models from scratch to demonstrate the fundamentals of the language. Then, we point out similar models which already exist within the MSL. In this way, we can cover language fundamentals and models available in the MSL.

Keep in mind that the MSL is not a collection of *black box* models which are hard-wired into a tool. Instead, the Modelica representation of all the models can be viewed to help understand exactly what behavior is modeled. These models are no different than any other Modelica models. It should be noted that while the models contained in the MSL are useful, you are not required to use them.

While reading this book, be on the lookout for uses of the MSL. These can be easily recognized by looking for names that begin with "`Modelica.`". All such entities belong to the MSL. For example, the physical type `Modelica.SIunits.Voltage` is defined in the MSL. You should interpret this name to mean "`Voltage` is a type defined in the `SIunits` package nested inside the `Modelica` package". The **package** structure of Modelica libraries (including the MSL) is hierarchical and may contain numerous nested packages. Do not be surprised to see much longer names like:

```
Modelica.Electrical.Analog.Basic.Resistor
```

1.5 BASIC VOCABULARY

The Modelica language specification uses a precise vocabulary for describing the elements of the Modelica language. While being rigorous is necessary in a formal specification, it is not always good in learning material. For this reason, this book uses a simplified vocabulary. In the remaining chapters, the following terms are used:

[14] As with most things related to Modelica, the MSL can be found at `http://www.modelica.org`

model A model is a behavioral description. For example, a model of a resistor is described by Ohm's law. The model is a description of resistor behavior, not the resistor itself. In other words, it is important to separate the idea of a resistor model (*i.e.*, $V = I*R$) from the resistor instances (components with different values of resistance, R). If you are familiar with object-oriented programming, a model is analogous to a class.

component A component is an instance of a model. So, for a given model (*e.g.*, a resistor model), the actual instances (*e.g.*, the resistors) would be components.

subcomponent A subcomponent is used to refer to components which are contained within other components. For example, a resistor might be a subcomponent of another component like an electrical circuit. Furthermore, the electrical circuit could be a subcomponent of an appliance. Subcomponents are used to form hierarchical models.

system model A system model is a model which is completely self-contained. In other words, it does not have any external connections and it contains the same number of equations as unknowns.

quantity A quantity refers to those entities which have a value (*e.g.*, the resistance of a resistor). In Modelica, all values are either real, integer, string or boolean. Furthermore, a quantity might be a scalar or an array.

definition The description of all variables, parameters and equations associated with a model is called the model definition.

declaration When a component, parameter, variable or constant is instantiated (either in a system model or inside another component), that is called a declaration.

package A package refers to a collection of Modelica models, which are meant to be used together. For example, an electrical package would likely include definitions of resistor, capacitor and inductor models.

keyword A keyword is a word, such as `model`, that has a specific meaning in Modelica. As a result, keywords are reserved words and cannot be used as names in declarations (*e.g.*, of variables). In the examples, the keywords will appear in bold.

Use the explanations of these terms as a reference to help understand the more complicated explanations in this book. The glossary, which starts on page 324, includes these terms and many more used in this book.

1.6 SUMMARY

In summary, the Modelica language is a non-proprietary, domain-neutral modeling language that supports several different modeling formalisms. Modelica can be used to model both continuous and discrete behavior and an extensive multi-domain library of models known as the Modelica Standard Library is available free of charge at `http://www.modelica.org`.