CHAPTER

Getting Started with SysML

3

This chapter provides an introduction to SysML and guidance on how to begin modeling in SysML. The chapter provides a brief overview of SysML, and then introduces a simplified version of the language we refer to as SysML-Lite, along with a simplified example, and tool tips on how to capture the model in a typical modeling tool. This chapter also introduces a simplified model-based systems engineering (MBSE) method that is consistent with the systems engineering process described in Chapter 1, Section 1.2. The chapter finishes by describing some of the challenges involved in learning SysML and MBSE.

3.1 SysML PURPOSE AND KEY FEATURES

SysML[1] is a general-purpose graphical modeling language that supports the analysis, specification, design, verification, and validation of complex systems. These systems may include hardware, software, data, personnel, procedures, facilities, and other elements of man-made and natural systems. The language is intended to help specify and architect systems and specify their components that can then be designed using other domain-specific languages such as UML for software design and VHDL and three-dimensional geometric modeling for hardware design. SysML is intended to facilitate the application of an MBSE approach to create a cohesive and consistent model of the system that yields the benefits described in Chapter 2, Section 2.1.2.

SysML can represent the following aspects of systems, components, and other entities:

- Structural composition, interconnection, and classification
- Function-based, message-based, and state-based behavior
- Constraints on the physical and performance properties
- Allocations between behavior, structure, and constraints
- Requirements and their relationship to other requirements, design elements, and test cases

3.2 SysML DIAGRAM OVERVIEW

SysML includes nine diagrams as shown in the diagram taxonomy in Figure 3.1. Each diagram type is summarized here, along with its relationship to UML diagrams:

- *Package diagram* represents the organization of a model in terms of packages that contain model elements (same as UML package diagram)

[1]OMG Systems Modeling Language (OMG SysML™) is the official name of the language, but it is referred to as SysML for short. Additional information on SysML can be found at the official OMG SysML website at http://www.omgsysml.org.

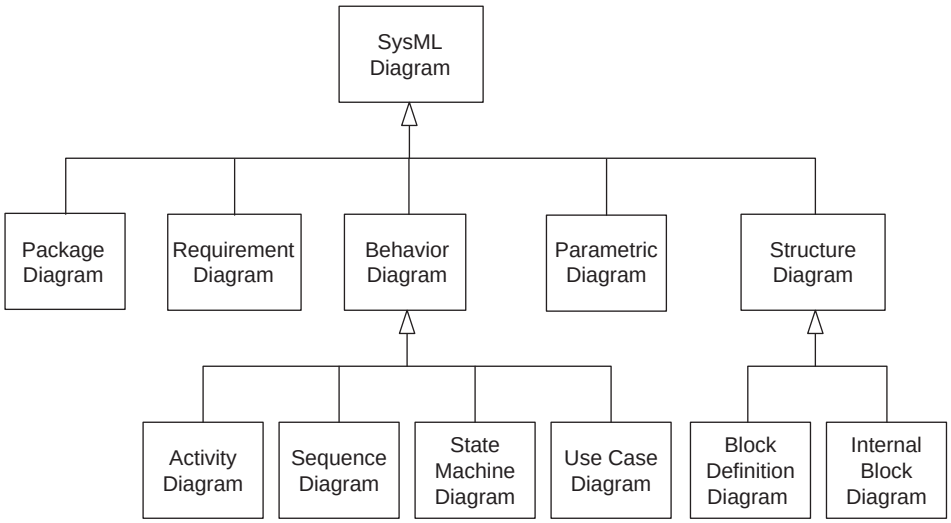

FIGURE 3.1

SysML diagram taxonomy.

- *Requirement diagram* represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability (not in UML)
- *Activity diagram* represents behavior in terms of the order in which actions execute based on the availability of their inputs, outputs, and control, and how the actions transform the inputs to outputs (modification of UML activity diagram)
- *Sequence diagram* represents behavior in terms of a sequence of messages exchanged between systems, or between parts of systems (same as UML sequence diagram)
- *State machine diagram* represents behavior of an entity in terms of its transitions between states triggered by events (same as UML state machine diagram)
- *Use case diagram* represents functionality in terms of how a system is used by external entities (i.e., actors) to accomplish a set of goals (same as UML use case diagram)
- *Block definition diagram* represents structural elements called blocks, and their composition and classification (modification of UML class diagram)
- *Internal block diagram* represents interconnection and interfaces between the parts of a block (modification of UML composite structure diagram)
- *Parametric diagram* represents constraints on property values, such as $F = m * a$, used to support engineering analysis (not in UML)

A diagram graphically represents a particular aspect of the system model as described in Chapter 2, Section 2.1.2. The kinds of model elements and associated symbols (e.g., diagram elements) that can appear on a diagram are constrained by its diagram kind. For example, an activity diagram can include diagram elements that represent actions, control flow, and input/output flow (i.e., object flow), but not diagram elements for connectors and ports. As a result, a diagram represents a subset of the underlying model repository, as described in Chapter 2, Section 2.1.2. Tabular representations, such as allocation

tables, are also supported in SysML as a complement to diagram representations to represent model information.

3.3 INTRODUCING SysML-LITE

SysML-Lite is introduced here as a simplified version of the language to help people get started modeling with SysML, but is not part of the SysML standard. It includes six of the nine SysML diagrams, and a small subset of the available language features for each diagram kind. SysML-Lite provides a significant modeling capability. This section provides a brief introduction to SysML-Lite, along with a simple example to highlight the features of SysML-Lite. This section also includes tool tips to assist a new modeler in the use of a typical modeling tool.

3.3.1 SysML-Lite Diagrams and Language Features

The six (6) kinds of diagrams that are part of SysML-Lite are highlighted in Figure 3.2. Each diagram contains a diagram header that identifies the diagram kind, and other information about the diagram which is explained in Chapter 5, Section 5.3.2. In particular, SysML-Lite includes the:

- package diagram to capture the model organization
- requirement diagram to capture text-based requirements
- activity diagram to represent the behavior of the system and its components
- block definition diagram to represent the system hierarchy
- internal block diagram to represent the system interconnection
- parametric diagram to capture the relationship among system properties to support engineering analysis

This set of diagrams provides a model user with a substantial capability for modeling systems that covers many of the classical systems engineering diagrams and more.

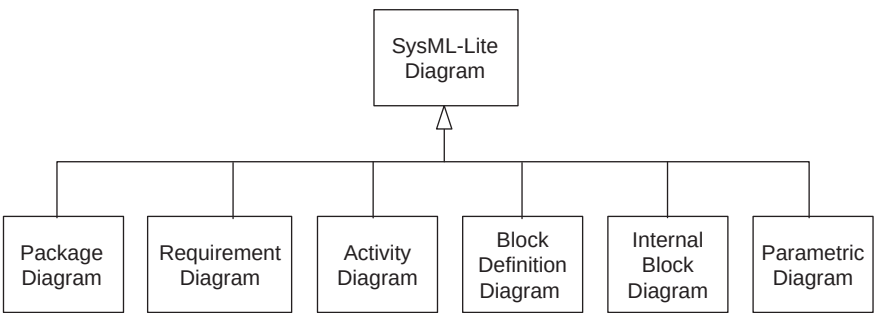

FIGURE 3.2

SysML-Lite includes six of the nine SysML diagrams and a subset of the language features. It is intended to introduce a new modeler to SysML, while providing a substantial modeling capability.

SysML-Lite includes a small subset of the language features for each of the six SysML diagrams. Some of the features of SysML-Lite are represented in the diagrams in Figure 3.3. The precise subset of SysML language features can be adapted to the need. The figure also shows thick lines with arrowheads that are not part of the language, but highlight some of the important cross diagram relationships. These relationships are generally consistent with classical systems engineering methods, such as functional decomposition and allocation.

The package diagram, labeled pkg, is used to organize the **model elements** contained in the model. In this diagram, the *System Model* appears in the diagram header and contains packages for *Requirements*, *Behavior*, *Structure*, and *Parametrics*. Each of these packages, in turn, contains model elements that are represented on the requirements diagram, activity diagram, block definition diagram, internal block diagram, and parametric diagram, respectively. Note that model elements for both the block definition diagram and internal block diagram are contained in the *Structure* package.

The requirement diagram is labeled req and represents a simple hierarchy of text-based requirements that are typically part of a specification document. The top level requirement named *R1* contains two requirements *R1.1* and *R1.2.* The corresponding requirement statement for *R1.1* is captured as a text property of the requirement and corresponds to the text that would be found for this requirement in the specification document.

The activity diagrams are labeled act. The activity diagram named *A0* represents the interaction between *System 1* and *System 2*. The initial node represented by the filled dark circle and final node represented by the bulls-eye indicate the start and finish of the activity, respectively. The activity specifies a simple sequence of actions starting with the execution of action *:A1*, and followed by the execution of action *:A2*. The output of *:A1* and the input of *:A2* are represented by rectangles on the action boundary called pins. In addition, the activity partitions labeled *:System 1* and *:System 2* are responsible for performing the actions that are enclosed by the partitions. The action called *:A1* satisfies the requirement *R1.2* which is represented by the *satisfy* relationship.

The action called *:A1* in the activity diagram *A0* is decomposed in the activity diagram called *A1* into actions *:A1.1* and *:A1.2*. These actions are performed by *:Component 1* and *:Component 2,* respectively. The output of the activity *A1* represented by the rectangle on its boundary corresponds to the output pin of action *:A1* in activity *A0.* As indicated in the activity diagrams for *A0* and *A1*, the outputs and inputs are consistent from one level of decomposition to the next.

The block definition diagram is labeled bdd and is often used to describe the hierarchy of a system similar to a parts tree (e.g., equipment tree). A block is used to define a system or component at any level of the system hierarchy. The block definition diagram in the figure shows the block *System Context* composed of *System 1* and *System 2*. *System 1* is further decomposed into *Component 1* and *Component 2*. The *System 1* and *Component 1* blocks each contain a value property that can correspond to a physical or performance characteristic, such as its weight or response time.

The internal block diagram is labeled ibd and shows how the parts of *System 1* are interconnected. The enclosing diagram frame represents *System 1*. The small squares on *System 1* and its parts are called ports and represent their interfaces. *System 1* is also represented by the activity partition in the activity *A0*, and the components are similarly represented by activity partitions in the activity *A1*.

The parametric diagram is labeled par and is used to describe relationships among properties that correspond to an engineering analysis, such as performance, reliability, or mass properties analysis. In this example, the parametric diagram includes a single constraint called *Constraint 1* that corresponds

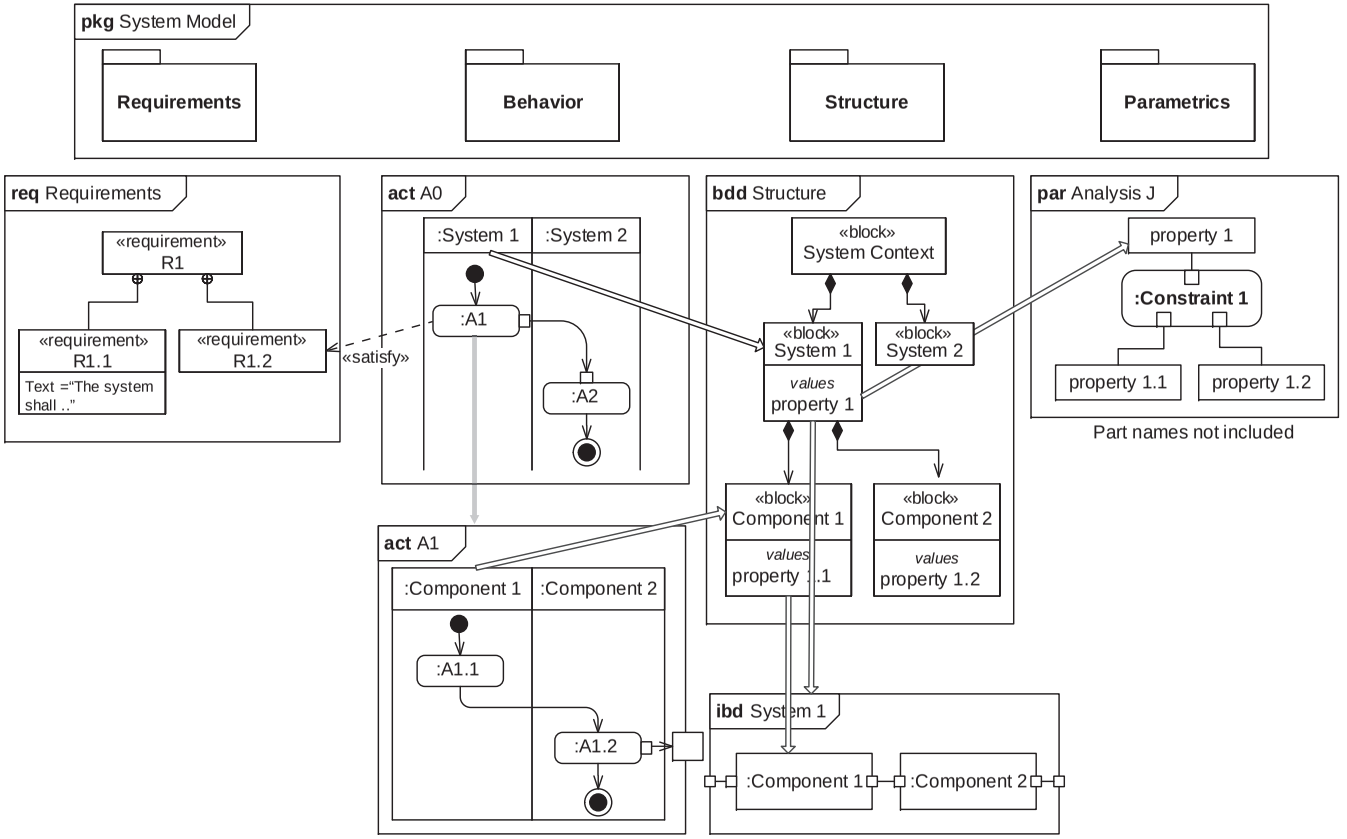

FIGURE 3.3

Simplified diagrams highlighting some of the language features for each kind of diagram in SysML-Lite.

to an equation or set of equations. The small squares flush with the inside of the constraint represent the parameters of the equation. The properties of the system and component blocks can then be bound to the parameters to establish an equality relationship. In this way, a particular analysis can be aligned with the properties of the system design. Often, a single constraint is used to represent a particular analysis, and the parameters represent the inputs and outputs of the analysis.

In the above diagrams, only a small subset of the SysML language features are illustrated to indicate some of the key constructs used to model systems. The following simplified model of an air compressor illustrates how SysML-Lite diagrams and language features can be applied.

Note that some of the names include a colon (:). This is described in Chapter 4, Section 4.3.12, and is further described in Chapter 7, Section 7.3.1.

3.3.2 SysML-Lite Air Compressor Example

The following is an example of using SysML-Lite to model an air compressor that is used to power an air tool. This model is highly simplified for the purposes of illustration, and includes the same type of diagrams that were shown in Figure 3.3.

Figure 3.4 shows the package diagram for the *Air Compressor Model* and includes packages for *Requirements*, *Behavior*, *Structure*, and *Parametrics*. The model organization follows a similar pattern as described in the section on SysML-Lite above and shown in Figure 3.3.

The *Requirements* package contains a set of requirements that would generally be found in a system specification for the air compressor. The requirements are captured in the requirements diagram in Figure 3.5. The top level requirement called *Air Compressor Specification* contains a functional requirement to compress air, performance requirements that specify the maximum pressure and maximum flow rate, a requirement to specify storage capacity, power requirements to specify the source power needed to compress the air, and reliability and portability requirements. The text for the *Storage Capacity* requirement appears in the diagram, whereas the text for the other requirements are not displayed to reduce the clutter.

The *Behavior* package contains an activity diagram, shown in Figure 3.6, called *Operate Air Tool* that specifies how the *Air Compressor* interacts with the external systems, including the *Air Tool,* the

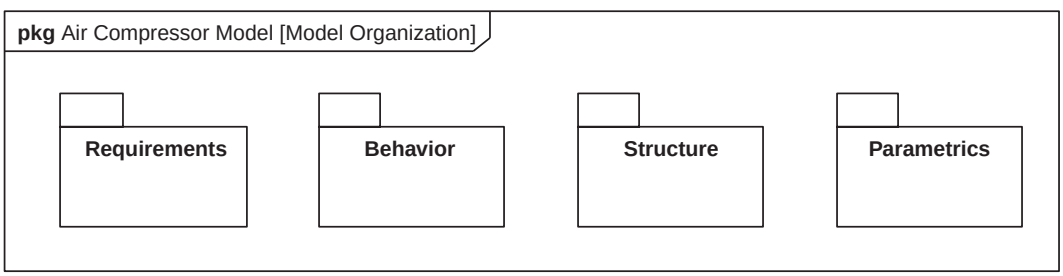

FIGURE 3.4

This package diagram is used to organize the *Air Compressor Model* into packages for *Requirements*, *Structure*, *Behavior*, and *Parametrics*. Each package contains model elements that can be related to model elements in other packages.

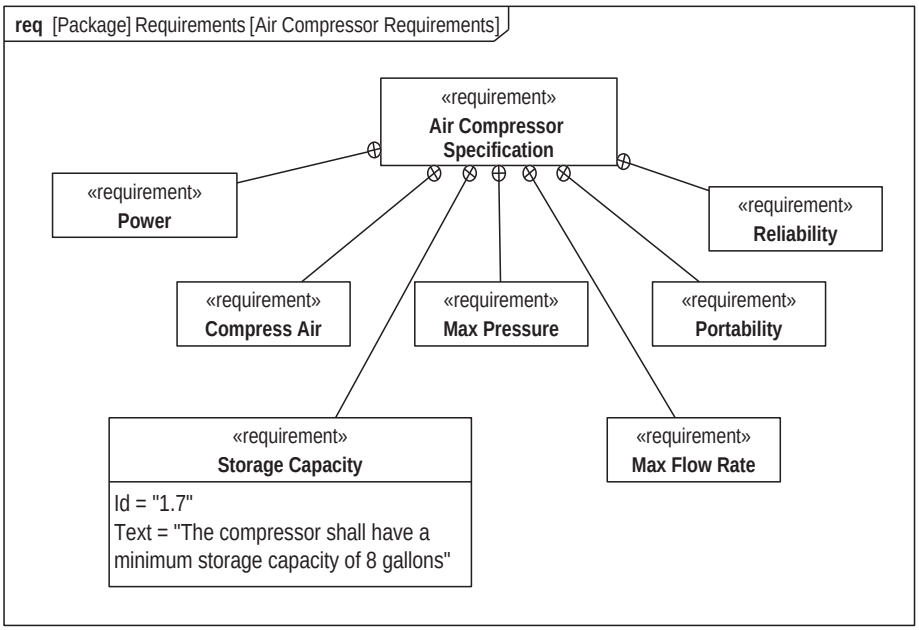

FIGURE 3.5

This requirement diagram represents the requirements contained in the *Requirements* package to specify the *Air Compressor*. Each requirement can include the requirements text that is typically found in a specification document.

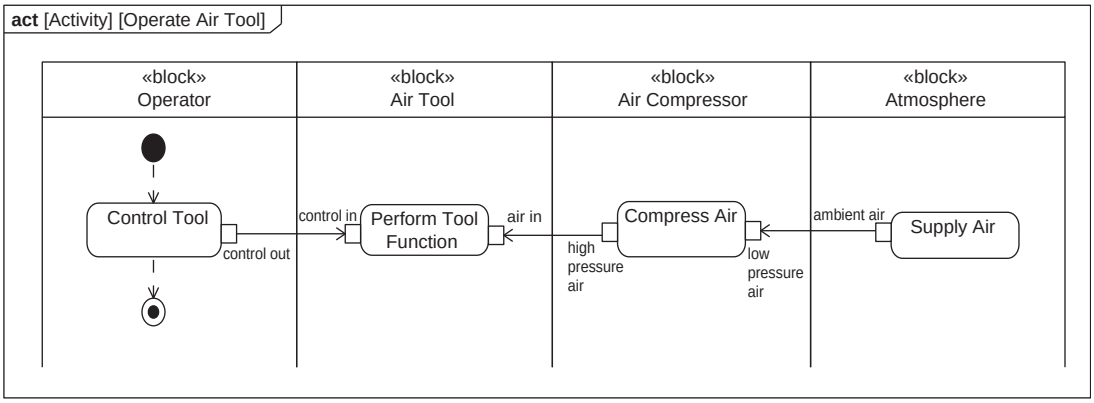

FIGURE 3.6

This activity diagram specifies the interaction between the *Air Compressor*, *Operator*, *Air Tool*, and *Atmosphere* to execute the *Operate Air Tool* activity.

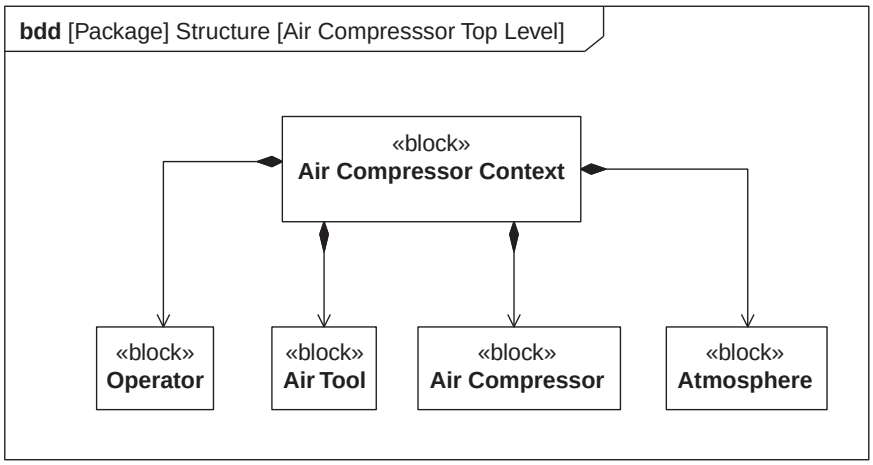

FIGURE 3.7

This block definition diagram represents the *Air Compressor*, *Operator*, *Air Tool*, and *Atmosphere* as blocks. The *Air Compressor Context* block sets the context for the *Air Compressor* and its external environment.

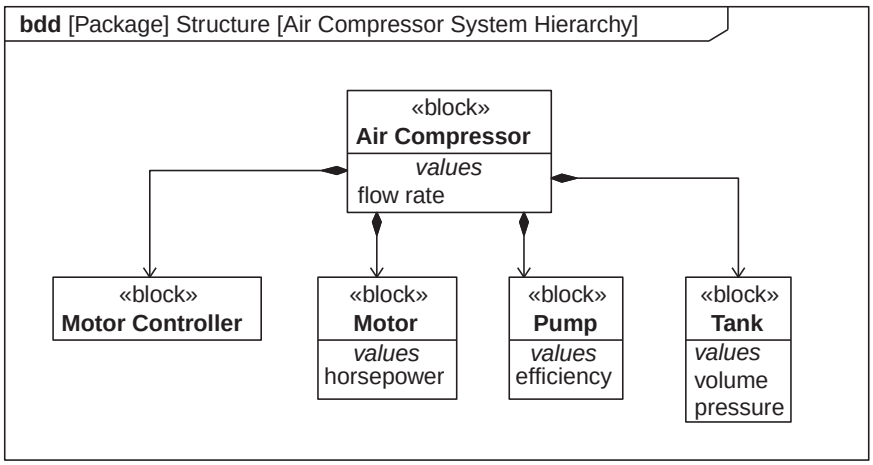

FIGURE 3.8

This block definition diagram represents the *Air Compressor* and its components. The *Air Compressor* block is the same block that is in Figure 3.7.

Atmosphere, and indirectly with the *Operator*, which are represented as activity partitions. The *Air Compressor* performs the function (i.e., action) called *Compress Air*, which has a *low pressure air* input and a *high pressure air* output. The activity begins at the initial node (i.e., dark-filled circle), and then the *Operator* executes the *Control Tool* action. The activity completes its execution at the activity final node (i.e., bulls-eye symbol), after the *Control Tool* action completes execution. The *Compress Air* action is further decomposed in Figure 3.9.

The *Structure* package contains the blocks represented in the block definition diagrams in Figure 3.7 and Figure 3.8. The block definition diagram in Figure 3.7 called *Air Compressor Top Level* includes a block called the *Air Compressor Context* that is composed of the *Air Compressor* and the blocks representing the user, external system, and the physical environment. In this example, the user is the *Operator*, the external system is the *Air Tool*, and physical environment is the the *Atmosphere*. The block definition diagram in Figure 3.8 is called *Air Compressor System Hierarchy.* The *Air Compressor* block in this figure is the same block that is shown in Figure 3.7, but this figure shows that the *Air Compressor* block is composed of components that include the *Motor Controller*, *Motor*, *Pump*, and *Tank*. The *Air Compressor*, *Motor, Tank*, and *Pump* all include value properties that are used to analyze the flow rate requirements.

The activity diagram in Figure 3.9 decomposes the action called *Compress Air* from Figure 3.6 to specify how the components of the *Air Compressor* interact to compress the air. The activity partitions in the activity diagram represent the components of the air compressor. The *Motor Controller* includes actions to *Sense Pressure* and *Control Motor.* The *Motor* performs the action to *Generate Torque*, the *Pump* performs the action to *Pump Air*, and the *Tank* performs the action to *Store Air.* The *low pressure air* input and *high pressure air* output are consistent with the input and output of the *Compress Air* action in Figure 3.6. This activity is contained in the *Behavior* package along with the *Operate Air Tool* activity.

The internal block diagram called *Interconnection* in Figure 3.10 shows how the components of the *Air Compressor* from Figure 3.8 are interconnected. The diagram frame represents the *Air Compressor* block and the ports on the diagram frame represent the external interfaces of the *Air Compressor.* The component parts shown on the internal block diagram are contained in the *Structure* package along with the blocks represented on the block definition diagram.

The block definition diagram called *Analysis Context* in Figure 3.11 is used to define the context for performing the flow rate analysis. In particular, it includes a block called *Flow Rate Analysis* that is composed of a constraint block called *Flow Rate Equations*. This constraint block defines the parameters of the equation and the equations, but the equations are not specified at this time. The *Flow Rate Analysis* block also refers to the *Air Compressor Context* block from Figure 3.7 which is the subject of the analysis. Defining the *Analysis Context* enables a parametric diagram to be created for the *Flow Rate Analysis* block as shown in Figure 3.12. The diagram shows the value properties of the *Air Compressor* and its parts that include *flow rate*, tank *volume* and *pressure*, motor *horsepower*, and pump *efficiency*, and how they are bound to the parameters of the *Flow Rate Equations*. The flow rate analysis equations can be solved by an analysis tool, to determine the property values for the *Air Compressor* and its parts. The analysis context pattern is described further in Chapter 8, Section 8.10 and in Chapter 17, Section 17.3.6.

This air compressor example illustrates how a system can be modeled with a subset of SysML diagrams and language features called SysML-Lite. Even a simple model such as this can contain many model elements, and quickly become difficult to manage. A modeling tool is needed to

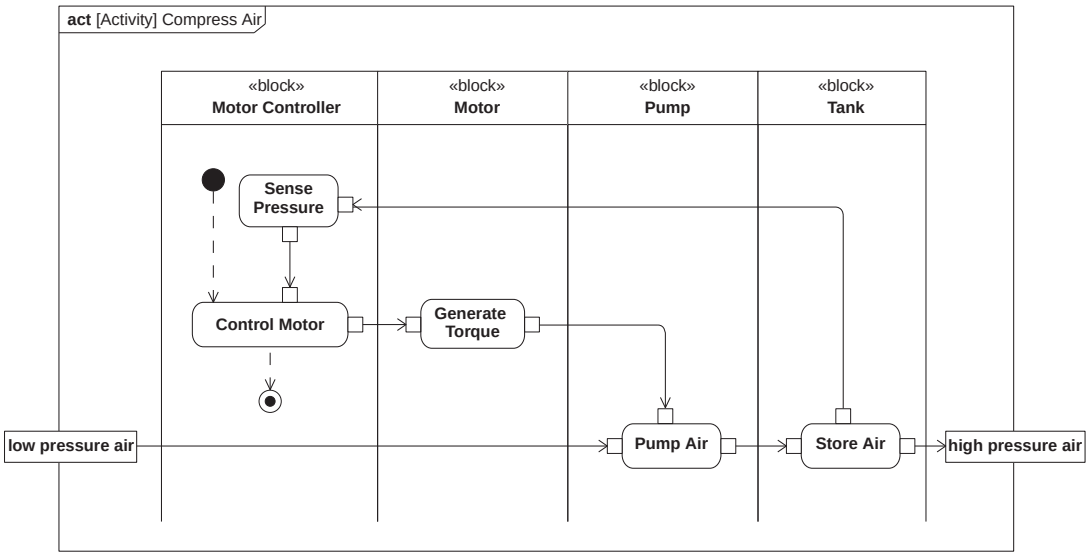

FIGURE 3.9

This activity diagram shows how the components of the *Air Compressor* interact to perform the *:Compress Air* action from Figure 3.6.

efficiently build a model that is self consistent, and to manage complexity. The following section describes how a typical SysML modeling tool is used to build this model.

3.3.3 SysML Modeling Tool Tips

This section provides a brief introduction on how to start modeling with a typical **SysML modeling tool**. The question of how to start modeling often arises when one opens a modeling tool for the first time. Although each tool may have significant differences, the tools typically share much in common from a user interface perspective. As a result, once a modeler learns how to build a SysML model in one tool, it generally takes considerably less time to learn how to model in another tool. Chapter 18 includes a discussion on SysML modeling tools including their role in a typical systems development environment, how to integrate the SysML modeling tool with other tools, and suggested criteria for selecting a SysML modeling tool.

The Tool Interface

The user interface for a typical modeling tool is shown in Figure 3.13, and typically includes a diagram area, a pallet (also known as toolbox), a model browser, and a toolbar. The diagram area is where the diagram appears. The pallet includes diagram elements that are used to create or modify a diagram. The pallet is typically context sensitive such that the diagram elements that appear in the pallet depend on the diagram that is being viewed in the diagram area. For example, if a block definition diagram is being viewed in the diagram area, then the pallet will contain blocks

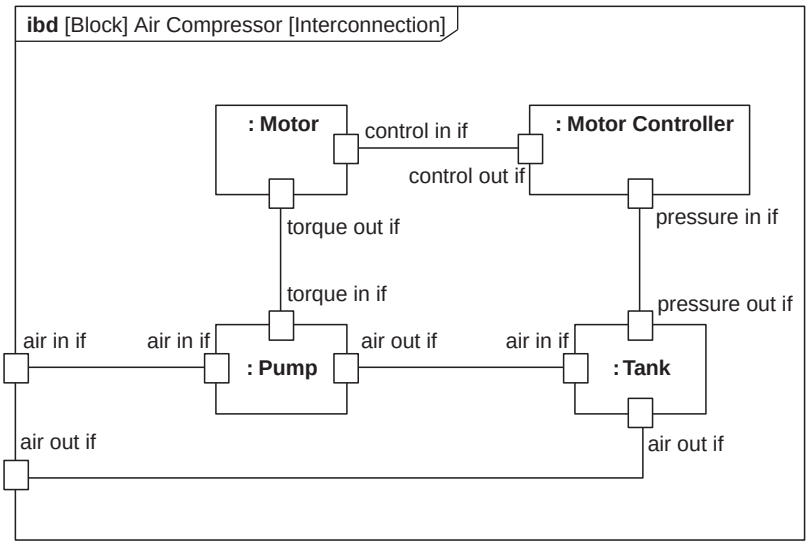

FIGURE 3.10

This internal block diagram shows how the components of the *Air Compressor* are interconnected via their ports, which are used to specify the component interfaces.

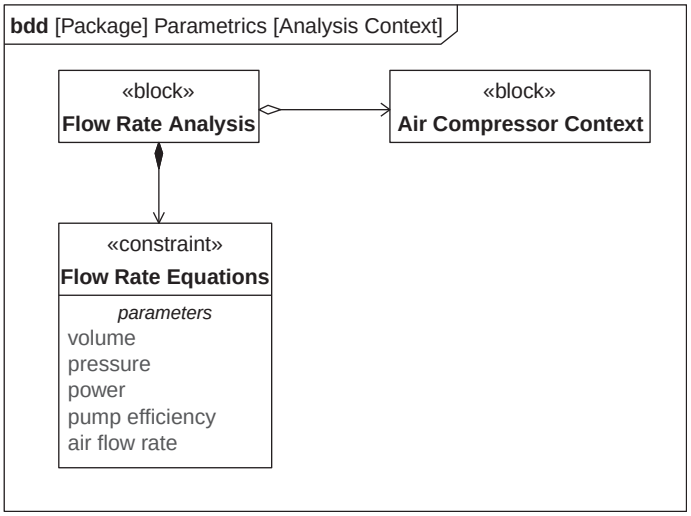

FIGURE 3.11

This block definition diagram is used to specify the *Flow Rate Analysis* in terms of a constraint block that defines the equations and parameters for the analysis (equations not shown), and the *Air Compressor Context* which is the subject of the analysis.

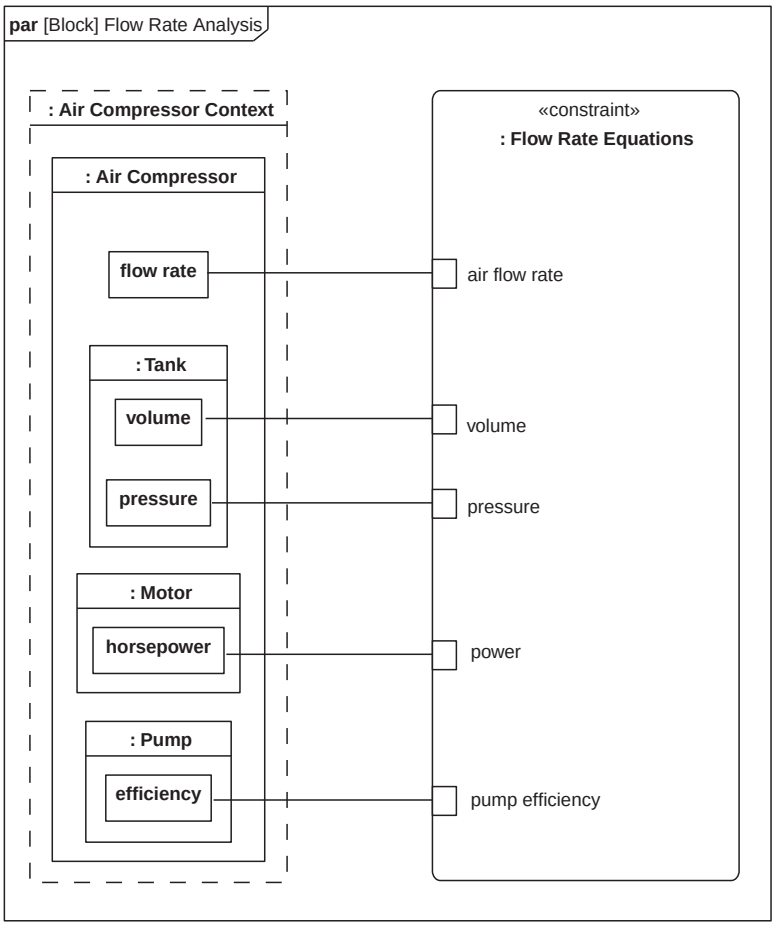

FIGURE 3.12

This parametric diagram represents the *Flow Rate Analysis,* and how the parameters of the equations are bound to the properties of the design. Once captured, this analysis can be provided to an analysis tool to perform the analysis. The equations are not shown in the figure.

and other elements used on a block definition diagram, whereas if an activity diagram is being viewed, the pallet will include actions and other elements used on an activity diagram. The model browser is a third part of the user interface which represents a hierarchical view of the model elements contained in the model. A typical view of the browser shows the model elements grouped into a package hierarchy, where each package appears like a folder that can be expanded to see its content. A package may contain other nested packages. The toolbar contains a set of menu selections that support different operator actions related to file management, editing, viewing, and other actions.

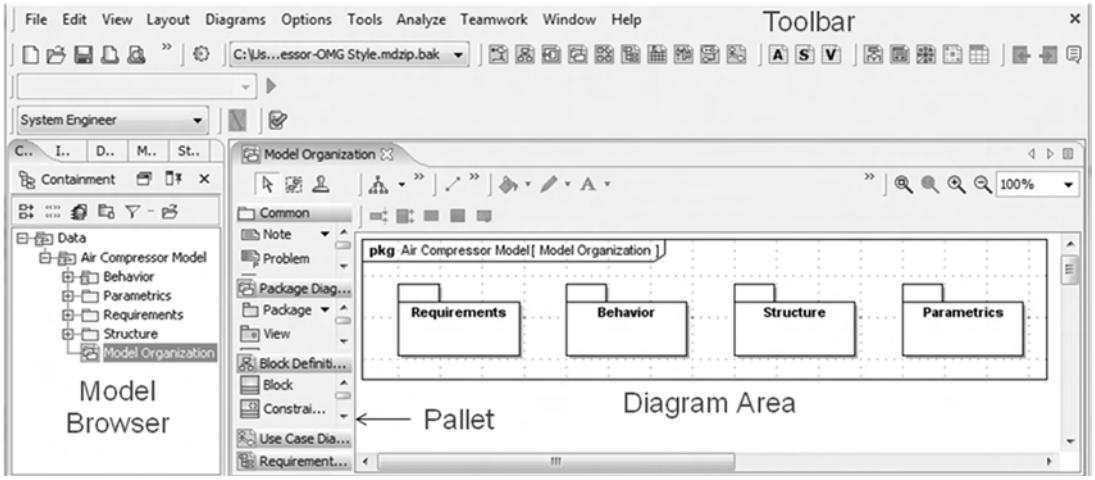

FIGURE 3.13

A typical SysML modeling tool interface consists of a diagram area, a pallet or toolbox, a model browser, and a toolbar. The model browser shows the hierarchy of model elements that are contained in the model.

To create a new diagram, a modeler selects a diagram kind and names the diagram. There are often multiple ways to select a diagram kind, such as from a diagram menu or a diagram icon from the toolbar. The new diagram appears in the diagram area without any content. The diagram header information is visible and includes the diagram kind, the diagram name, and other information about the diagram frame. The modeler can then drag a diagram element from the pallet onto the diagram in the diagram area and name the new element. Once this is done, the corresponding model element appears in the browser. As an alternative, the modeler can add the new model element directly in the browser, and then drag this model element onto the diagram. A model element appears in only one place in the browser, but may appear on zero, one or more diagrams.

A modeling tool provides mechanisms for navigating between the elements on the diagrams and the model elements in the browser. This can be important since a large model may contain hundreds of diagrams, and thousands or hundreds of thousands of model elements. Most tools allow the modeler to select the model element on the diagram and request its location in the browser. A modeler can also select a model element in the browser, and request a list of the diagrams where the model element appears.

The modeling tool allows the modeler to show and hide selected details of the model on any particular diagram. This is important for managing the complexity of the diagrams. The modeler only shows what is considered important to support the purpose of the diagram.

If the modeler wishes to delete a model element from the diagram, the tool may prompt the modeler whether to delete the model element from the diagram only, or to delete the model element from the model as well. A modeler can also choose to delete a model element from the model by selecting the model element in the browser, and then deleting it.

A modeling tool has many other capabilities that enable a modeler to develop and manage a system model. Once the model element is created, the modeler can typically select the model element and open up its specification where details of the model element can be added, modified, or deleted. The modeler can also select a model element on the diagram, and query the modeling tool to show all of the directly related model elements that can appear on that particular kind of diagram.

It is also worth noting that the modeling tool is often used in conjunction with a configuration management tool to put the model under configuration control. This is particularly important when modeling as part of a distributed team where multiple people are working on the same model. In this case, a typical configuration management tool will allow read and/or write privileges to be assigned to a user to control their access to different parts of the model. Once this is done, a modeler with read privileges assigned to a particular part of the model can view that part of the model, and a modeler with write privileges assigned to a particular part of the model can also check out and modify that part of the model.

Chapter 18 describes how the SysML modeling tool integrates into a Systems Development Environment with many other tools including configuration management, requirements management, hardware and software design, and analysis tools.

Building the Model

The following illustrates how to build the *Air Compressor Model* from Section 3.3.2 in a typical modeling tool. Each tool will have its unique user interface, and different modeling guidelines and MBSE methods may suggest different ways to get started. However, the following example provides a representative starting point, which can be further adapted to the specific modeling tool, modeling guidelines, and MBSE method.

The modeler must first install and configure the modeling tool so that it can be used to build a model that is represented in SysML. Many SysML tools also support UML and perhaps other modeling languages, so the modeler may be required to select and apply SysML. Once this is done, a modeler can create a new project and name this project, such as the Air Compressor Project.

As indicated in Figure 3.13, the first step in building the model is to create the top level package in the browser called the *Air Compressor Model.* The modeler can then select this package in the browser, and create nested packages for *Requirements*, *Behavior*, *Structure*, and *Parametrics*. Alternatively, the modeler can create a new package diagram similar to the one shown in Figure 3.4, by dragging new packages from the pallet onto the diagram and naming them accordingly.

The modeler can then select the *Requirements* package in the browser, and create a new requirements diagram and name it *Air Compressor Requirements*. Once the diagram appears in the diagram area, the modeler can drag new requirements from the pallet onto the diagram and name them to correspond to the requirements in Figure 3.5. The relationship between model elements can then be defined using the kind of relationships shown in the pallet. In the case of requirements, a parent and child requirement can be related by connecting the parent requirement to each child requirement with the cross hair symbol at the parent requirement end.

The modeler next creates the top level activity diagram *Operate Air Tool* shown in Figure 3.6. This is done by selecting the *Behavior* package, and creating a new activity diagram, and naming the

diagram *Operate Air Tool*. The modeler may drag actions from the pallet onto the activity diagram, along with the initial and final nodes, and connect the actions with the appropriate flow. The control flow is used to connect the initial node to *Control Tool*, and another control flow connects *Control Tool* to the activity final node. The object flows connect the inputs and outputs for each of the actions. The activity partitions can be added after the next step in the process.

The modeler next creates the block definition diagram for the *Air Compressor Context* shown in Figure 3.7. This is accomplished by selecting the *Structure* package in the browser and creating a new block definition diagram, and naming it *Air Compressor Top Level*. A new block can be dragged from the pallet onto the diagram and called *Air Compressor Context* block. The other blocks can then be defined similarly. The composition relationship between the *Air Compressor Context* block and the other blocks can be established in a similar way as described for the requirements diagram but using the composition relationship designated by the black diamond on one end of the line.

Once the blocks are defined, the activity partitions (i.e., swim lanes) in the activity diagram in Figure 3.6 can be defined to represent these blocks. This activity diagram specifies the interaction between the *Air Compressor*, *Operator*, *Air Tool*, and *Atmosphere* to execute the *Operate Air Tool* activity. This is accomplished by selecting the previously created activity diagram, *Operate Air Tool,* to view in the diagram area. The modeler then drags the activity partitions from the pallet onto the diagram. In order to represent an activity partition by a particular block, the modeler typically opens the activity partition specification, and then selects the particular block to represent the partition. Each action is then placed within the activity partition corresponding to the block that is responsible for performing the action.

The modeler can then decompose the system into its component parts by creating the block definition diagram shown in Figure 3.8. This is done by selecting the *Structure* package, and creating a new block definition diagram, and naming it *Air Compressor System Hierarchy.* New blocks can be dragged from the pallet onto the diagram, and the relationships are established in a similar way as described for the block definition diagram called *Air Compressor Top-Level*. The ports on each of the blocks can then be created by dragging a port from the pallet onto the block, or, alternatively, by selecting a block and opening up its specification, and then adding the ports. In addition, the properties of the block can be created be opening up each block's specification on the diagram or from the browser, adding a new property, and naming it. In this example, both the ports and the properties are included in the model, but not shown on the diagram, in order to further simplify the diagram.

The modeler next creates the activity diagram to show the interaction between the parts of the *Air Compressor* as shown in Figure 3.9. This activity diagram is created in a similar way as the previous activity diagram *Operate Air Tool*. However, this activity is used to decompose the *Compress Air* action that the *Air Compressor* performs in the *Operate Air Tool* activity. The new activity is created by first ensuring the *Compress Air* action is a special type of action called a call behavior action, which then calls the new activity called *Compress Air.* The activity partitions can then be dragged from the pallet onto the diagram, and can now represent the component blocks created in the *Air Compressor System Hierarchy* block definition diagram.

The modeler next creates the internal block diagram shown in Figure 3.10. This is accomplished by selecting the *Air Compressor* block in the *Structure* package in the browser, and creating

a new internal block diagram. When the composition relationships were previously created between the *Air Compressor* and its component blocks, the tool should create new model elements in the browser under the *Air Compressor* block. These elements are called parts, and are used in the internal block diagram for the *Air Compressor.* The parts of the *Air Compressor* block are dragged from the browser onto the internal block diagram, and then connected to one another via their ports. The ports on the parts may not be visible on the diagram. Many tools require the modeler to select the part, and select a menu item to display the ports. The ports can be connected to one another once the ports are visible on the diagram. A modeler may also connect the parts without ports, and add ports later if desired.

The modeler next creates the block definition diagram in Figure 3.11 to specify the constraints used in the parametric diagram. This is done by selecting the *Parametrics* package in the browser, and creating a new block definition diagram and naming the diagram *Analysis Context*. The *Flow Rate Analysis* block is created, and the *Air Compressor Context* block that is contained in the *Structure* package is dragged onto the diagram and referenced (i.e., white diamond aggregation) by the *Flow Rate Analysis* block. A new constraint block called *Flow Rate Equations* is then created, and related to the *Flow Rate Analysis* block with a composition relationship. The parameters of the constraint block are then defined in a similar way as the properties of blocks described earlier. The equations can be specified as part of the constraint block.

The modeler next creates the parametric diagram shown in Figure 3.12. The constraint property which is typed by the *Flow Rate Equations* constraint block and a part which is typed by the *Air Compressor Context* block are dragged from the browser onto the diagram. The *Air Compressor Context* is selected on the diagram, and its nested parts and value properties are displayed on the diagram. Different tools accomplish this in different ways. Once this is done, the value properties contained in the *Air Compressor*, *Tank*, *Motor,* and *Pump* can be connected to the parameters of the *Flow Rate Equations* constraint property.

Creating this example in the modeling tool is an important first step to learning how to model. Once this is understood, one can learn additional SysML language features, and explore additional tool capabilities such as documentation generation, tabular representations, diagram layout functions, etc. The automobile example in Chapter 4 introduces the remaining three (3) SysML diagrams and additional language features that can serve as a next step in the learning process.

3.4 A SIMPLIFIED MBSE METHOD

In addition to learning the modeling language and a tool, a modeler must apply a disciplined model-based systems engineering (MBSE) method to adhere to sound systems engineering practice and build quality system models. SysML provides a means to capture the system modeling information without imposing a specific MBSE method. A selected method determines which modeling activities are performed, the ordering of the activities, and the types of modeling artifacts used to represent the system. For example, traditional structured analysis methods can be used to decompose the functions and then allocate the functions to components. Alternatively, one can apply a use case driven approach that derives functionality based on scenario analysis and the associated interactions among the parts. The two methods may involve different activities and produce different combinations of diagrams to represent the system specification and design. Several MBSE methods are documented in the Survey

of Model-based Systems Engineering Methodologies [5]. Chapters 16 and 17 provide two examples with different MBSE methods.

The top level activities for a simplified MBSE method are highlighted in Figure 3.14. The activities are consistent with the systems engineering process introduced in Chapter 1, Section 1.2. The method also represents a simplified variant of the object-oriented systems engineering method (OOSEM) that is described in detail as it is applied to the residential security example in Chapter 17. This method includes one or more iterations of the following activities to specify and design the system:

- *Organize the Model*
  - Define the package diagram for the system model
- *Analyze Stakeholder Needs* to understand the problem to be solved, the goals the system is intended to support, and the effectiveness measures needed to evaluate how well the system supports the goals
  - Identify the stakeholders and the problems to be addressed
  - Define the domain model to identify the system and external systems and users
  - Define the top level use cases to represent the goals the system is intended to support
  - Define the effectiveness measures that can be used to quantify the value of a proposed solution
- *Specify System Requirements* including the required system functionality, interfaces, physical and performance characteristics, and other quality characteristics to support the goals and effectiveness measures
  - Capture text-based requirements in a requirements diagram that support the system goals and effectiveness measures
  - Model each use case scenario (e.g., activity diagram) to specify the system behavior requirements
  - Create the system context diagram (internal block diagram) to specify the system external interfaces
- *Synthesize Alternative System Solutions* by partitioning the system design into components that can satisfy the system requirements
  - Decompose the system using the block definition diagram
  - Define the interaction among the parts using activity diagrams
  - Define the interconnection among the parts using the internal block diagram
- *Perform Trade-off Analysis* to evaluate and select a preferred solution that satisfies the system requirements and maximizes value based on the effectiveness measures
  - Capture the analysis context to identify the analysis to be performed such as performance, mass properties, reliability, cost, and other critical properties
  - Capture each analysis as a parametric diagram
  - Perform the engineering analysis to determine the values of the system properties (Note: the analysis is performed in engineering analysis tools)
- *Maintain Requirements Traceability* to ensure the proposed solution satisfies the system requirements and associated stakeholder needs
  - Capture the traceability between the system requirements and the stakeholder needs
  - Show how the system design satisfies the system requirements
  - Identify test cases needed to verify the system requirements and capture the verification results

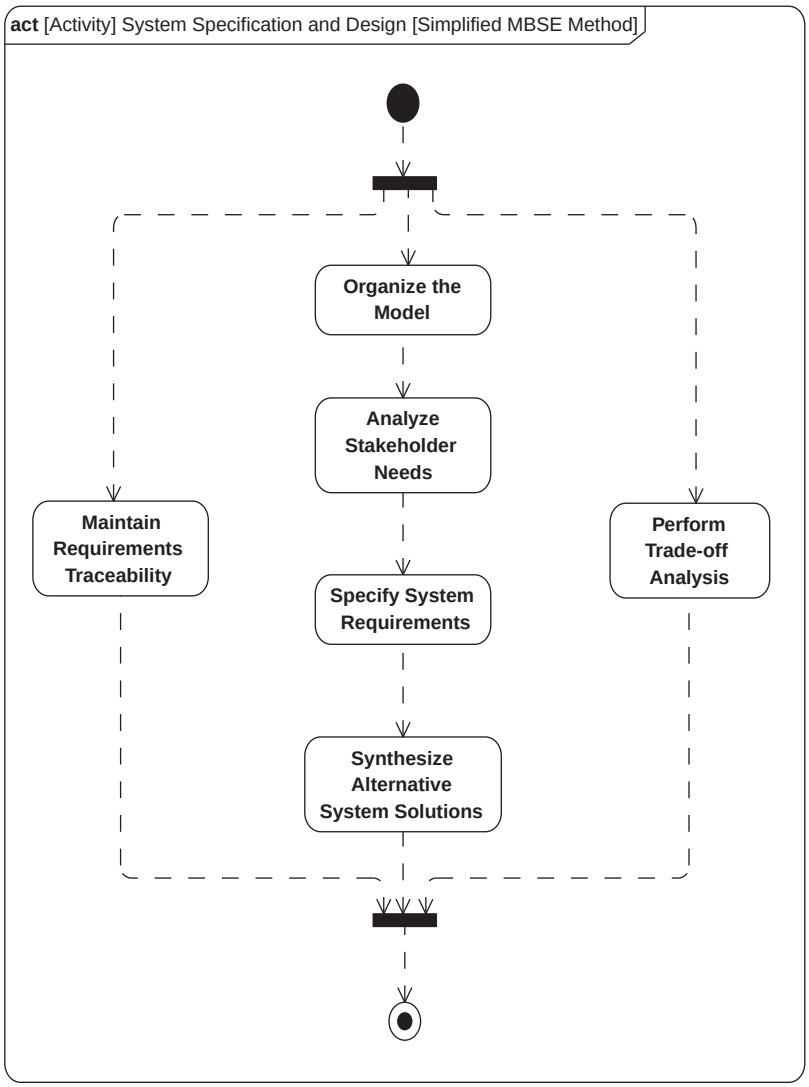

FIGURE 3.14

A simplified MBSE method that is consistent with the systems engineering process described in Chapter 1, Section 1.2. The method is used to produce the modeling artifacts that constitute the system model.

Other systems engineering management activities, such as planning, assessment, risk management, and configuration management are performed in conjunction with the modeling activities described above. Detailed examples of how SysML can be used to support a functional analysis and allocation method and the object-oriented systems engineering method (OOSEM) are included in the modeling

examples in Part III Chapters 16 and 17, respectively. A simplified example is described in the next chapter, which illustrates some of the model-based artifacts that are generated when applying a typical MBSE method.

3.5 THE LEARNING CURVE FOR SysML AND MBSE

Learning SysML and MBSE requires a commitment similar to what is expected of learning modeling for mechanical, electrical, software, and other technical disciplines. The challenges to learning SysML and MBSE have some additional factors that contribute to its learning curve. In particular, a major focus for model-based systems engineering approaches is the ability to understand a system from multiple perspectives, and to ensure integration across the different perspectives. In SysML, the system requirements, behavior, structure, and parametrics each represents different aspects of the system that need to be understood individually and together.

Each of the individual perspectives introduces its own complexity. For example, the modeler may represent behavior in activity diagrams to precisely specify how a system responds to a stimulus. This involves specifying the details of how the system executes each use case scenario. These activity diagrams may be integrated into a composite system behavior that is captured in a state machine diagram. The process for representing detailed behavior and integrating different behavior formalisms can be quite complex.

As stated above, the modeler must maintain consistency from one perspective to another. SysML is often used to represent hierarchies for requirements, behavior, structure, and parametrics. A consistent model of a system must ensure consistency between the model elements in each hierarchy. Some of these relationships were highlighted in the examples in Sections 3.3.1 and 3.3.2. Additional discipline-specific views may cross cut the requirements, behavior, structure, and parametrics perspectives, such as a reliability view, security view, or manufacturing view. Again, this introduces complexity to system modeling and MBSE.

Another aspect of complexity is that an effective MBSE approach not only requires a language such as SysML to represent the system, but also a method that defines the activities and artifacts, and a tool to implement the modeling language and method. The language, method, and tool each introduce their own concepts, and must be learned to master model-based systems engineering. This skill must then be applied to a particular domain, such as designing aircraft, automobiles, telecommunication systems, medical devices, and others.

Additional modeling challenges are associated with scaling the modeling effort to larger projects, and in the context of a diverse development environment. The challenges of managing the model come into play. There may be multiple modelers in multiple locations. Disciplined processes and associated tools must be put in place to manage changes to models. There are also many different types of models involved in an MBSE effort beyond the SysML model, such as a multitude of analysis models, hardware models, and software models. The integration among the different models and tools, and other engineering artifacts is another challenge associated with MBSE.

Model-based systems engineering formalizes the practice of how systems engineering is performed. The complexity and associated challenges for learning MBSE reflect the inherent complexity and challenges of applying systems engineering to the development of complex systems. Some of this complexity was highlighted in the automobile design example in Chapter 1, Section 1.3 independent of

the MBSE approach. When starting out on the MBSE journey, it is important to set expectations for the challenges of learning MBSE and how to apply it to the domain of interest. However, in addition to reaping the potential benefits of MBSE described in Chapter 2, embracing these challenges and becoming proficient in SysML and MBSE can provide a deeper understanding of systems and systems engineering concepts.

3.6 SUMMARY

SysML is a general-purpose graphical language for modeling systems that may include hardware, software, data, people, facilities, and other elements within the physical environment. The language supports modeling of requirements, structure, behavior, and parametrics to provide a robust description of a system, its components, and its environment.

The language includes nine diagram kinds each with and many features. The semantics of the language enable a modeler to develop an integrated model of a system, where each kind of diagram can represent a different view of the system being modeled. The model elements on one diagram can be related to model elements on other diagrams. The diagrams enable capturing the information in a model repository, and viewing the information from the repository, to help specify, design, analyze, and verify systems. To facilitate the learning process, SysML-Lite is introduced, which includes six of the nine SysML diagrams and a relatively small subset of the language features for each diagram kind. Learning how to model this subset of the language in a modeling tool can provide a sound foundation to build on.

The SysML language is a critical enabler of MBSE. Effective use of the language requires a well-defined MBSE method. SysML can be used with a variety of MBSE methods. This chapter introduced a simplified MBSE method to aid in getting started.

SysML enables representation of a system from multiple perspectives. Each of the individual perspectives may be complex in their own right, but ensuring a consistent model that integrates across the different perspectives introduces additional challenges to learning SysML and MBSE. When learning SysML as part of an overall MBSE approach, the process, methods, and tools introduce their own concepts and complexity. Using SysML in support of MBSE formalizes the practice for how systems engineering is performed. Ultimately, the challenges of SysML and MBSE reflect the inherent complexities of applying systems engineering to developing complex systems. The learning expectations should be set accordingly.

3.7 QUESTIONS

1. What are five aspects of a system that SysML can represent?
2. What is a package diagram used for?
3. What is a requirement diagram used for?
4. What is an activity diagram used for?
5. What is the block definition diagram used for?
6. What is an internal block diagram used for?
7. What is a parametric diagram used for?
8. What are some of the common elements of the user interface of a typical SysML modeling tool?

9. Which element of the user interface reflects the model repository?
10. What is the purpose of applying an MBSE method?
11. What are the primary activities of the simplified MBSE method?

Discussion Topics

What are some factors that contribute to the challenges of learning SysML and MBSE, and how do they relate to the general challenges of learning systems engineering?