CHAPTER

4 AN AUTOMOBILE EXAMPLE USING THE SysML BASIC FEATURE SET

This chapter introduces the **basic feature set** of SysML. The basic feature set applies to all nine SysML diagrams and provides an expanded subset of the language features beyond the features of SysML-Lite that were introduced in the previous chapter. The basic feature set provides significant functionality of the language without adding the complexity associated with the full feature set of SysML.

In this chapter, a system model of an automobile similar to the one that was introduced in Chapter 1, Section 1.3, illustrates the use of the basic feature set. This example includes references to the chapters in Part II that provide a more detailed description of the diagrams and language concepts. The subset of the SysML constructs that comprise the basic feature set are highlighted by shaded paragraphs in Part II and in the notation tables in Appendix A.

4.1 THE SysML BASIC FEATURE SET AND SysML CERTIFICATION

The basic and full feature set provides language functionality that can be learned in steps and are the basis for SysML certification. The SysML certification program is called the OMG Certified Systems Modeling Professional (OCSMP) [39]. The OCSMP has four levels of certification. The first two levels of certification cover the basic feature set of SysML. These two levels are referred to as Model User and Model Builder-Fundamental. A modeler certified at the Model User level is expected to be able to interpret SysML diagrams that use the basic feature set, while a modeler certified at the Model Builder-Fundamental level is expected to be able to build models that use the basic feature set. The third level covers the full feature set of SysML. An individual certified at this level is called a Model Builder-Intermediate and is expected to be able to build models that use the full feature set of SysML. The fourth level covers additional modeling concepts that extend beyond SysML.

4.2 AUTOMOBILE EXAMPLE OVERVIEW

The following simplified example illustrates how the basic feature set of SysML can be applied as part of a model-based approach to specify and design an automobile system. This example is similar

A Practical Guide to SysML. http://dx.doi.org/10.1016/B978-0-12-800202-5.00004-7

to the automobile example that was introduced in Chapter 1, Section 1.3, which described how the systems engineering process can be applied to the specification and system level design of an automobile. In Chapter 1, no assumptions were made regarding the use of a model-based approach. The example in this chapter highlights how a typical MBSE method can be used to generate modeling artifacts to help specify and design a system. The MBSE method is similar to the one introduced in Chapter 3, Section 3.4. Chapters 16 and 17 introduce much more detailed examples of how MBSE methods can be applied.

This example illustrates most of the SysML basic feature set and includes at least one diagram for each SysML diagram kind. A few features in the example extend beyond the basic feature set of SysML—including continuous and streaming flows and generalization sets—because they illustrate important features for this particular example. These additional features are noted in the example where they are used. References are also included in this section to the chapters and sections in Part II that provide a detailed description of these features.

This example also includes user-defined language concepts referred to as **stereotypes**. Chapter 15 describes how stereotypes are used to customize the language for domain-specific applications. The user defined concepts used in this example are shown below using the name of the concept in brackets:

```
«hardware»
«software»
«store»
«system of interest»
```

All SysML diagrams include a **diagram frame** that encloses the diagram header and diagram content. The **diagram header** describes the kind of diagram, the diagram name, and additional information that provides context for the **diagram content**. Detailed information on diagram frames and diagram headers is described in Chapter 5, Section 5.2.

4.2.1 PROBLEM SUMMARY

This example describes the use of SysML to specify and design an automobile system. As mentioned earlier, the modeling artifacts included in this example are representative of the kinds of modeling artifacts that are generated from a typical MBSE method similar to the one described in Chapter 3, Section 3.4. Only a small subset of the system requirements and design are addressed in this example to highlight the use of the language. The diagrams used in this example are shown in Table 4.1.

A marketing analysis that was conducted indicated the need to increase the automobile's acceleration and fuel efficiency from its current capability. In this simplified example, selected aspects of the design are considered to support an initial trade-off analysis. The trade-off analysis includes an evaluation of alternative vehicle configurations that included a 4-cylinder engine and a 6-cylinder engine to determine if they can satisfy the acceleration and fuel efficiency requirement.

Table 4.1 Diagrams Used in Automobile Example

Figure	Diagram Kind	Diagram Name
4.1	Package diagram	Model Organization
4.2	Requirement diagram	Automobile System Requirements
4.3	Block definition diagram	Automobile Domain
4.4	Use case diagram	Operate Vehicle
4.5	Sequence diagram	Drive Vehicle
4.6	Sequence diagram	Turn On Vehicle
4.7	Activity diagram	Control Power
4.8	State machine diagram	Drive Vehicle States
4.9	Internal block diagram	Vehicle Context
4.10	Block definition diagram	Vehicle Hierarchy
4.11	Activity diagram	Provide Power
4.12	Internal block diagram	Power Subsystem
4.13	Block definition diagram	Analysis Context
4.14	Parametric diagram	Vehicle Acceleration Analysis
4.15	Timing diagram (not SysML)	Vehicle Performance Timeline
4.16	Block definition diagram	Engine Specification
4.17	Requirement diagram	Max Acceleration Requirement Traceability
4.18	Package diagram	Architect and Regulator Viewpoints

4.3 AUTOMOBILE MODEL

The following subsections describe the system model for the automobile example.

4.3.1 PACKAGE DIAGRAM FOR ORGANIZING THE MODEL

The concept of an integrated system model is a foundational concept for MBSE, as described in Chapter 2, Section 2.1.2. The model contains the model elements, which are stored in a model repository. A particular model element may appear on zero, one, or multiple diagrams. In addition, a model element often has relationships to other model elements that may appear on the same diagram or other diagrams.

A model organization is essential to managing the model. A well-organized model is analogous to having a set of drawers to organize your supplies, where each supply element is contained in a drawer, and each drawer is contained in a particular cabinet. The model organization facilitates understandability, access control, change management, and reuse of the model.

The package diagram for the automobile example is shown in Figure 4.1. The diagram kind is shown as *pkg* and the name of the diagram is *Model Organization*. The package diagram shows how the model is organized into **packages**. This model organization includes an expanded set of packages

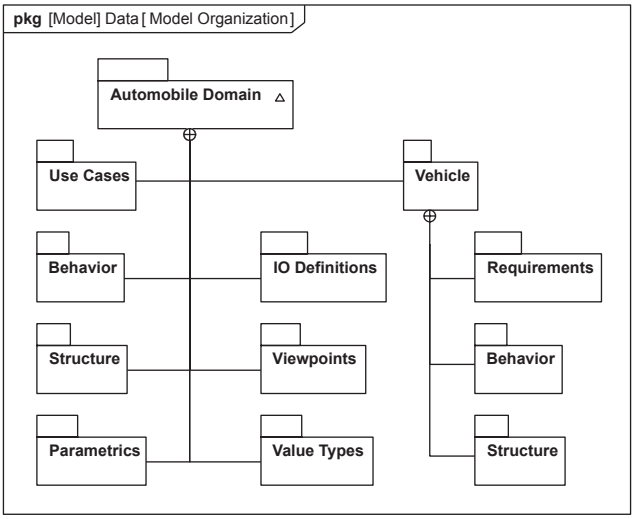

FIGURE 4.1

Package diagram showing how the model is organized into packages that contain the model elements that comprise the *Automobile Domain*.

over those that were introduced in the air compressor example using SysML-Lite in Chapter 3, Section 3.3.2. Each package **contains** a set of model elements, and each model element is contained in only one package. The package is said to own the elements that are contained within it. The package is also a namespace for the contained model elements, giving each model element a unique name within the model that is called its fully qualified name. A model element in one package can have relationships to model elements in other packages. Details on how to organize a model with packages are provided in Chapter 6.

The model organization for this example includes a package called the *Automobile Domain.* This package is the top-level model (designated by a triangle) that contains all the other model elements for the automobile example. The *Automobile Domain* contains nested packages for *Use Cases, Behavior*, *Structure*, *Parametrics*, *IO Definitions*, *Viewpoints, Value Types,* and *Vehicle*. The *Vehicle* package contains additional nested packages for *Requirements*, *Behavior*, and *Structure*. The *Use Cases*, *Behavior*, *Structure*, and *Parametrics* packages contain model elements about the vehicle context and its external environment, whereas the *Vehicle* package contains model elements about the vehicle design. The *IO Definitions* package contains elements to specify the interfaces, such as port definitions and inputs and output definitions. The *Viewpoints* package defines selected views of the model that address specific stakeholder concerns. The *Value Types* package contains definitions that are used to specify units for quantitative properties called **value properties**.

The rest of this example describes the content of these packages. Model elements contained in packages can be referenced by their fully qualified name as described above. The qualified name includes the path name relative to the model in which it is contained using a double colon (::) as a separator. For example, an activity called Provide Power in the vehicle behavior package in Figure 4.1 is designated as *Automobile Domain::Vehicle::Behavior::*Provide Power.

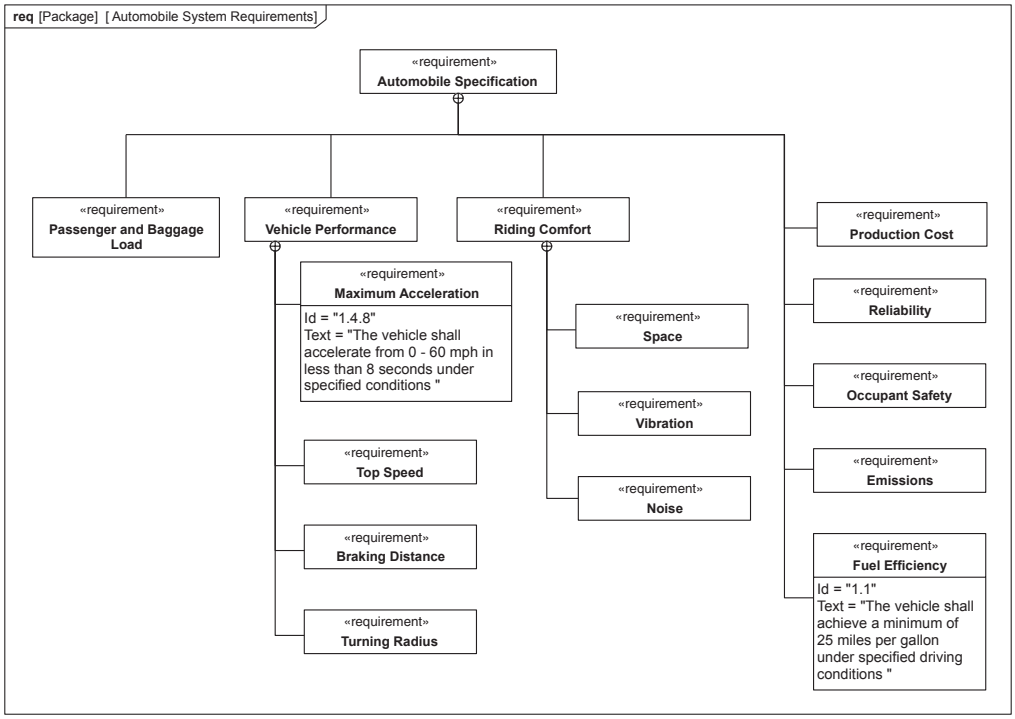

FIGURE 4.2

Requirement diagram showing the system requirements contained in the *Automobile Specification*.

4.3.2 CAPTURING THE *AUTOMOBILE SPECIFICATION* IN A REQUIREMENT DIAGRAM

The **requirement diagram** for the Automobile System is shown in Figure 4.2. The upper left of the diagram shows *req* to indicate its kind as a requirement diagram and displays the diagram name as *Automobile System Requirements*. The diagram header also indicates that the diagram frame corresponds to a *Package.*

The diagram presents the requirements that are typically captured in a text specification. The requirements are shown in a containment hierarchy to represent their parent–child relationships. The line with the crosshairs symbol at the top denotes **containment**. The *Automobile Specification* is the top-level requirement that contains the other requirements.

The *Automobile Specification* contains requirements for *Passenger and Baggage Load, Vehicle Performance, Riding Comfort, Emissions, Fuel Efficiency, Production Cost, Reliability,* and *Occupant Safety*. The *Vehicle Performance* requirement contains requirements for *Maximum Acceleration, Top Speed, Braking Distance*, and *Turning Radius*. Each requirement includes a unique identification and the text of the requirement, and can also include other user-defined properties that are typically associated with requirements, such as verification status and risk. The text for the *Maximum Acceleration* requirement is "The vehicle shall accelerate from 0 to 60 mph in less than 8 seconds under specified conditions" and the text for the *Fuel Efficiency* requirement is "The vehicle shall achieve a minimum of 25 miles per gallon under specified driving conditions."

The requirements may have been created in the SysML modeling tool or, alternatively, in a requirements management tool or a text document and imported into the model. Once captured in the model, the requirements can be related to other requirements, design elements, analysis, and test cases using **derive, satisfy, verify, refine, trace,** and **copy** relationships. These relationships can be used to establish requirements traceability to ensure requirements are satisfied and verified, and to manage change to the requirements and design. Some relationships are highlighted in Section 4.3.18.

Requirements can be presented using multiple display options to view the requirements, their properties, and their relationships. A tabular presentation is one display option. Chapter 13 provides a detailed description of how requirements are modeled in SysML, and Chapter 17, Section 17.3.7, gives additional guidance for modeling requirements.

4.3.3 DEFINING THE *VEHICLE* AND ITS EXTERNAL ENVIRONMENT USING A BLOCK DEFINITION DIAGRAM

In system design, it is important to identify what is external to the system that may either directly or indirectly interact with the system. The **block definition diagram** for the *Automobile Domain* in Figure 4.3 defines the *Vehicle* and the external systems, users, and other entities with which the vehicle may interact.

A **block** is a very general modeling concept in SysML that is used to model entities that have structure, such as systems, hardware and equipment, software, or physical object. That is, a block can represent any real or abstract entity that can be conceptualized as a structural unit with one or more distinguishing features. The block definition diagram captures the relation between blocks, such as a block hierarchy.

In the block definition diagram in Figure 4.3, the *Automobile Domain* is the top-level block that provides the context for the *Vehicle*. The *Automobile Domain* block is composed of other blocks that include the *Vehicle* block (designated as the «system of interest») and other blocks that are external to the *Vehicle*. The other blocks include the *Driver, Passenger, Baggage,* and *Physical Environment.*

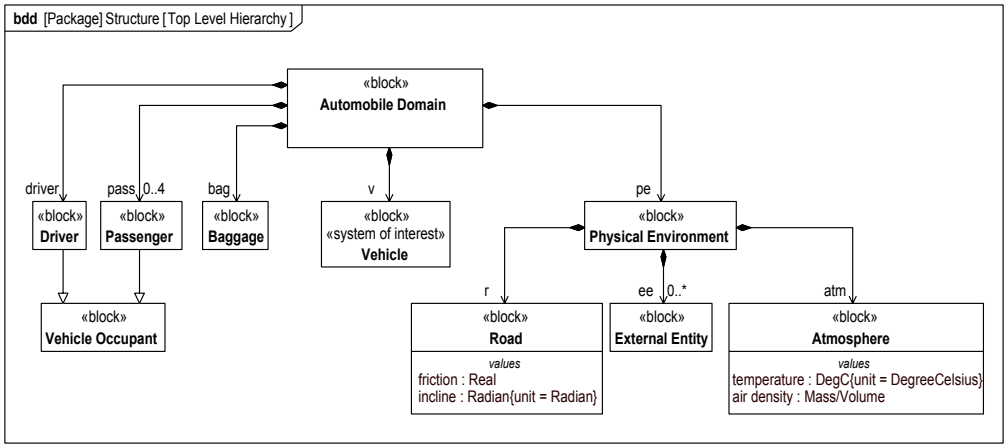

FIGURE 4.3

Block definition diagram of the *Automobile Domain* showing the *Vehicle* as the *system of interest*, along with the *Vehicle Occupants* and the Environment. Selected value properties for the *Road* and *Atmosphere* are also shown.

Notice that even though the *Driver, Passenger,* and *Baggage* are assumed to be physically enclosed by the *Vehicle,* they are not part of the *Vehicle*, and therefore are external to it.

This whole–part relationship is called a **composite association** and is indicated by the black diamond symbol and a line with the arrowhead pointing to the blocks that compose it. The name next to the arrow on the part side of the composite association identifies a particular usage of a block as described in Sections 4.3.10 and 4.3.12. The composition hierarchy is explained in Chapter 7, Section 7.3.1. It is different from containment (crosshair symbol), which connects parent to child requirements as shown in Figure 4.2. Requirement containment hierarchies are described in Chapter 13, Section 13.9.

The *Driver* and *Passenger* are **subclasses** of *Vehicle Occupant* as indicated by the hollow triangle symbol. This means that they inherit common features from *Vehicle Occupant.* In this way, a classification can be created by specializing blocks from more generalized blocks.

The *Physical Environment* is composed of the *Road*, *Atmosphere*, and multiple *External Entities.* The *External Entity* can represent any physical object, such as a traffic light or another vehicle, with which the *Driver* interacts. The interaction between the *Driver* and an *External Entity* can impact how the *Driver* interacts with the *Vehicle,* such as when the *Driver* sees the traffic light change from green to yellow or red, and then applies the brakes. The **multiplicity** symbol *0*..* represents an undetermined maximum number of external entities. The multiplicity symbol can also express a positive integer such as 4, or a range, such as the multiplicity of 0..4, for the number of *Passengers.*

Each block defines a structural unit, such as a system, hardware, software, data element, or other conceptual entity. A block can have a set of **features**. The features of the block include its **value properties** (e.g., weight), its **behavior** in terms of activities **allocated** to the block or **operations** of the block, and its interfaces as defined by its **ports.** Together, these features enable a modeler to specify the block at the level of detail that is appropriate for the intended use.

The *Road* is a block that has a value property called *incline* with units of *Radians* and a value property called *friction* that is defined as a real number. Similarly, *Atmosphere* is a block that has two value properties for *temperature* and *air density.* These value properties and others are used to support the analysis of vehicle acceleration and fuel efficiency, which are discussed in Sections 4.3.13–16.

The block definition diagram specifies the blocks and their interrelationships. It is often used in systems modeling to depict multiple levels of the system hierarchy from the top-level domain or context block (e.g., *Automobile Domain)* down to the blocks representing the vehicle components. Chapter 7 provides a detailed description of how blocks are modeled in SysML, including their features and relationships.

4.3.4 USE CASE DIAGRAM FOR *OPERATE VEHICLE*

The **use case diagram** for *Operate Vehicle* in Figure 4.4 depicts some of the high-level functionality involved in operating the vehicle. The **use cases** are contained in the *Use Cases* package and include *Enter Vehicle, Exit Vehicle, Control Vehicle Accessory,* and *Drive Vehicle.* The *Vehicle* is the **subject** of the use cases and is depicted as a rectangle. The *Vehicle Occupant* is an **actor** that is external to the vehicle and is shown as a stick figure. In a use case diagram, the subject (e.g., *Vehicle)* is used by the actor (e.g., *Vehicle Occupant)* to achieve the actor goals defined by the use cases (e.g., *Drive Vehicle)*. The actors are allocated to the blocks with the same name in Figure 4.3 to establish equivalence between them. The allocation is not shown in the diagrams.

The *Passenger* and *Driver* are both a type of *Vehicle Occupant*. All vehicle occupants participate in entering and exiting the vehicle and controlling vehicle accessories, but only the *Driver* participates in *Drive Vehicle.*

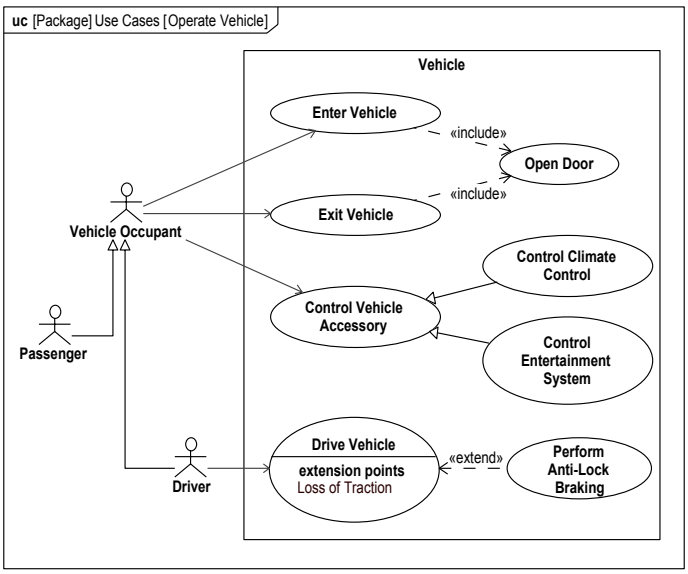

FIGURE 4.4

The use case diagram describes the major functionality in terms of how the *Vehicle* is used by the *Vehicle Occupants* to *Operate Vehicle*. The *Vehicle* and *Vehicle Occupants* are defined on the block definition diagram in Figure 4.3.

SysML provides the ability to specify relationships between use cases. The *Enter Vehicle* and *Exit Vehicle* use cases include the *Open Door* use case. The *Open Door* use case defines common functionality that is always performed when the *Enter Vehicle* and *Exit Vehicle* use cases are performed. *Enter Vehicle* and *Exit Vehicle* are referred to as the base use cases, and *Open Door* is referred to as the included use case. The relationship is called the **include** or **inclusion** relationship. The *Perform Anti-Lock Braking* use case extends the base use case called *Drive Vehicle*. Anti-lock braking is only performed under certain conditions as specified by the extension point called *Loss of Traction*. This relationship is called **extension** or **extends**, which relates the extending use case (i.e., *Perform Anti-Lock Braking)* to the base use case (i.e., *Drive Vehicle*). In addition to inclusion and extension relationships, use cases can be specialized as indicated by the subclasses of the *Control Vehicle Accessory* use case. The specialized use cases for *Control Climate Control* and *Control Entertainment System* all share the common functionality of *Control Vehicle Accessory* use case, but also have their own specific functionality associated with the particular accessory.

Use cases define the goals for using the system across the system lifecycle, such as the goals associated with manufacturing, operating, and maintaining the vehicle. The primary emphasis for this example is the operational use case for *Drive Vehicle* to address the acceleration and fuel efficiency requirements. Chapter 12 provides a detailed description of how use cases are modeled in SysML.

Use cases are often related to requirements, since use cases represent the high-level functionality or goals for the system. A use case often refines a set of requirements. Sometimes, a use case textual description is defined to accompany the use case definition. The steps in the use case description can also be captured as SysML requirements and related to the use case using a refine relationship.

The use cases are realized through interactions between the actors (e.g., *Driver)* and the subject (e.g., *Vehicle)* as described in the next section.

4.3.5 SPECIFYING *DRIVE VEHICLE* BEHAVIOR WITH A SEQUENCE DIAGRAM

The behavior for the *Drive Vehicle* use case in Figure 4.4 is shown in the **sequence diagram** in Figure 4.5. The sequence diagram specifies the **interaction** between the *Driver* and the *Vehicle* as indicated by the names at the top of the **lifelines.** Time proceeds vertically down the diagram. The first interaction is *Turn On Vehicle.* This is followed by *Driver* and *Vehicle* interactions to *Control Power, Control Brake,* and *Control Direction.* These three interactions occur in parallel as indicated by **par**. The **alt** on the *Control Power* interaction stands for alternative and indicates that the *Control Neutral Power, Control Forward Power,* or *Control Reverse Power* interaction occurs as a condition of the *vehicle state* shown in brackets. The state machine diagram in Section 4.3.8 specifies the *vehicle state.* The *Turn Off Vehicle* interaction occurs following these interactions.

The **interaction uses** in the figure each reference a more detailed interaction as indicated by **ref**. The referenced interaction for *Turn On Vehicle* is another sequence diagram that is illustrated in Section 4.3.6. The sequence diagrams for the *Drive Vehicle* and other referenced interactions are contained in the *Automobile Domain::Behavior* package. The references for *Control Neutral Power, Control Forward Power,* and *Control Reverse Power* are allocated to an activity diagram that is described in Section 4.3.7.

4.3.6 REFERENCED SEQUENCE DIAGRAM TO *TURN ON VEHICLE*

The *Turn On Vehicle* sequence diagram in Figure 4.6 is an interaction that is referenced in the sequence diagram in Figure 4.5. As stated previously, time proceeds vertically down the diagram. In this example, the sequence diagram shows the driver sending an *ignition on* **signal** to start the vehicle. The vehicle sends a *vehicle on* signal to the driver that the vehicle has started.

The sequence diagram can include multiple kinds of **messages**. In this example, the message is **asynchronous** as indicated by the open arrowhead. For asynchronous messages, the sender does not wait for a reply. A **synchronous** message is shown with a filled arrowhead. A synchronous message is an operation call that specifies a request for service, where the sender waits for a reply. The arguments of the operation call are the input data and return.

The example in Figure 4.6 is very simple. More complex sequence diagrams can include multiple message exchanges between multiple lifelines that represent interacting entities. The sequence diagram also provides additional capability to express behavior that includes other kinds of messages, timing constraints, additional control logic, and the ability to decompose the behavior of a lifeline into the interaction of its parts. Chapter 10 provides a detailed description of how interactions are modeled with sequence diagrams.

4.3.7 *CONTROL POWER* ACTIVITY DIAGRAM

The sequence diagram is effective for expressing behavior that emphasizes control flow and discrete signal flow, such as the *Turn On Vehicle* sequence diagram in Figure 4.6. However, behaviors that emphasize input and output flow as well as control flow, such as the interactions to *Control Power, Control Brake,* and *Control Direction*, can sometimes be more effectively expressed with activity diagrams.

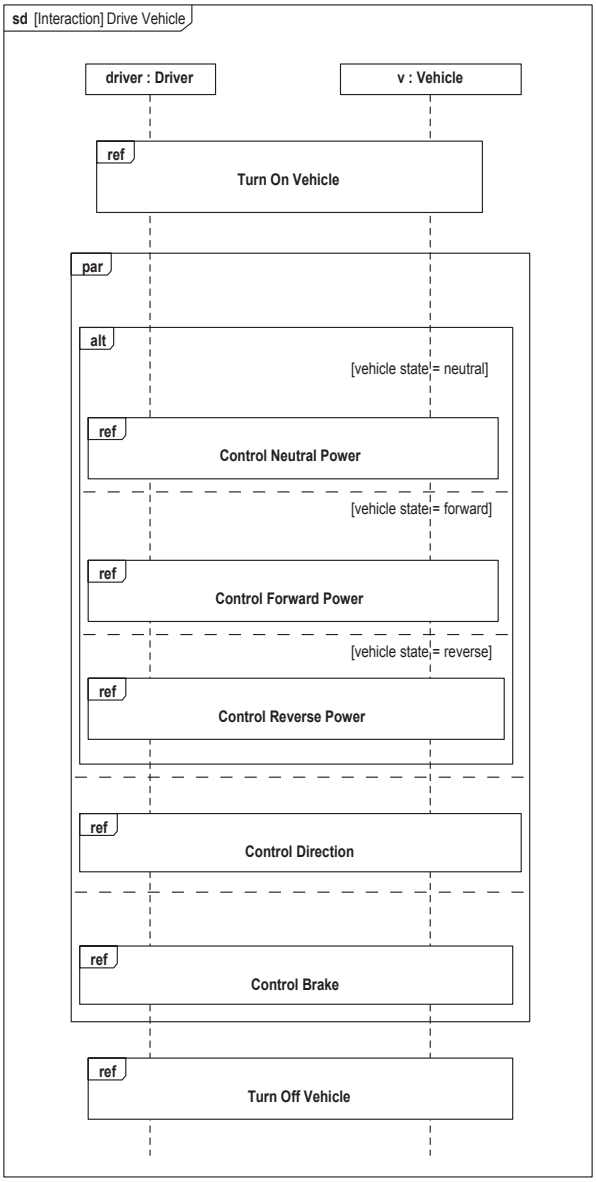

FIGURE 4.5

The *Drive Vehicle* sequence diagram describes the interaction between the *Driver* and the *Vehicle* to realize the *Drive Vehicle* use case in Figure 4.4.

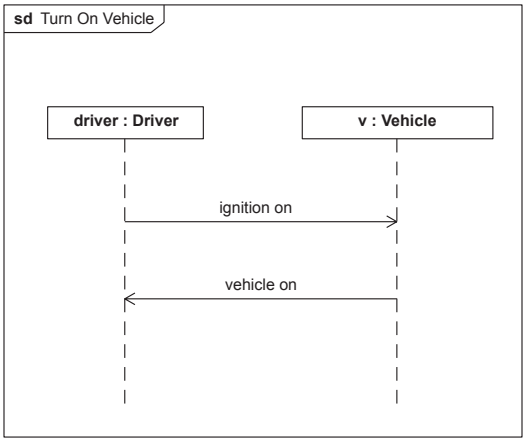

FIGURE 4.6

Sequence diagram for the *Turn On Vehicle* interaction that was referenced in the *Drive Vehicle* sequence diagram in Figure 4.5, showing the message from the *Driver* requesting *Vehicle* to start, and the *Vehicle* responding with the *vehicle on* reply.

The *Drive Vehicle* sequence diagram in Figure 4.5 includes the references to *Control Neutral Power, Control Forward Power,* and *Control Reverse Power*. Activity diagrams can be used to express the details of these interactions. To accomplish this, the *Control Neutral Power, Control Forward Power,* and *Control Reverse Power* interactions are **allocated** to a corresponding *Control Power* activity using the SysML allocate relationship (not shown). This activity is contained in the *Behavior* package of the *Automobile Domain*.

The **activity diagram** in Figure 4.7 shows the **actions** required of the *Driver* and the *Vehicle* to *Control Power.* The **activity partitions** (or **swim lanes)** correspond to the *Driver* and the *Vehicle.* The actions in the activity partitions specify functional requirements that the *Driver* and *Vehicle* must perform.

When the activity is initiated, it starts execution at the **initial node** (filled in circle)**,** and then proceeds to the **fork node** to enable the start of both the *Control Accelerator Position* action and the *Control Gear Select* action that is performed by the *Driver.* The output of the *Control Accelerator Position* action is the *Accelerator Cmd*, which is a continuous input to the *Provide Power* action that the *Vehicle* must perform. The *Control Gear Select* action produces an output called *Gear Select*. The output of the *Provide Power* action is the continuous *torque out* to accelerate the *Vehicle.* When the *Ignition Off* signal is received by the *Vehicle* (called an **accept event action**), the activity terminates at the **activity final node** (bulls-eye symbol). Based on this scenario, the *Driver* is required to *Control Accelerator Position* and *Control Gear Select*, and the *Vehicle* is required to *Provide Power*. The *Provide Power* action is a **call behavior action** that invokes a more detailed behavior when it executes, which is shown in Figure 4.11. (Note: «continuous» is not part of the basic feature set.)

Activity diagrams include semantics for precisely specifying the behavior in terms of the flow of control and flow of inputs and outputs. A control flow is used to specify the sequence of actions and is depicted as a dashed line with an arrowhead (as shown in Figure 4.7) going to and from the fork node. An object flow is used to specify the flow of inputs and outputs, which are depicted by the rectangular

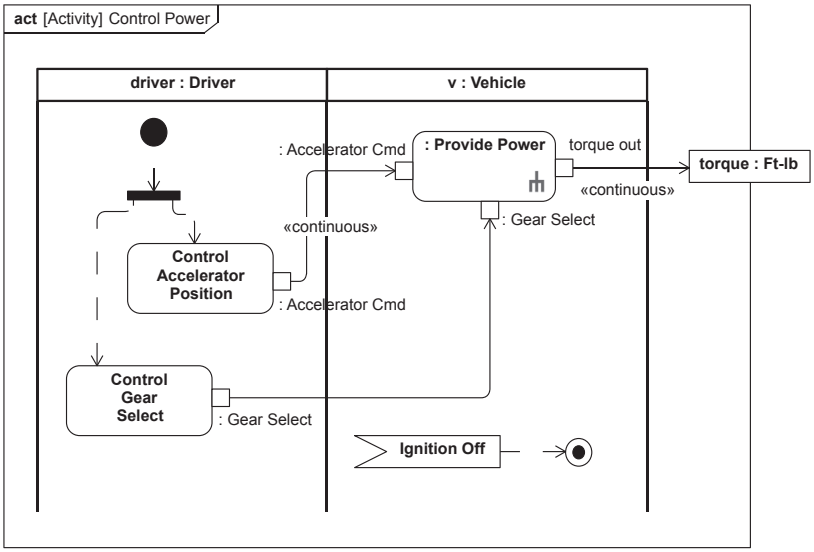

FIGURE 4.7

Activity diagram allocated from the *Control Neutral, Forward*, and *Reverse Power* interaction uses that are referenced in the *Drive Vehicle* sequence diagram in Figure 4.5. It shows the continuous *Accelerator Cmd* input and the *Gear Select* input from the *Driver* to the *Provide Power* action that the *Vehicle* must perform.

pins on the actions. The object flow (solid line with arrowhead) connects the output pin from one action to the input pin of another action. Chapter 9 provides a detailed description of how activities are modeled.

4.3.8 STATE MACHINE DIAGRAM FOR *DRIVE VEHICLE STATES*

The **state machine diagram** for the *Drive Vehicle States* is shown in Figure 4.8. This diagram shows the states of the *Vehicle* and the **events** that can **trigger** a **transition** between the **states**.

When the *Vehicle* is ready to be driven, it is initially in the *vehicle off* state. The receipt of the *ignition on* signal from the sequence diagram in Figure 4.6 is an event that triggers a transition to the *vehicle on* state. The text on the transition indicates that the *Start Vehicle* behavior is executed prior to entering the *vehicle on* state.

Upon entry to the *vehicle on* state, an **entry behavior** is performed, *Check Status*, to confirm the health of the vehicle. Following completion of the entry behavior, the *Vehicle* initiates the *Provide Power* behavior called a **do behavior** that is referred to in the activity diagram in Figure 4.7.

Once the *Vehicle* has entered the *vehicle on* state, it immediately transitions to the *neutral* state. A *forward select* event triggers a transition to the *forward* state if the **guard condition** *[speed>=0]* is true. The *neutral select* event triggers the transition from the *forward* state to return to the *neutral* state. The state machine diagram shows the additional transitions between the *neutral* and *reverse* states. An *ignition off* event triggers the transition back to the *vehicle off* state. Prior to exiting the *vehicle on* state and transitioning to the *vehicle off* state, the *Vehicle* performs an **exit behavior** to *Turn Off Accessories*. From the *vehicle off* state, the *Vehicle* can re-enter the *vehicle on* state when an *ignition on* event occurs.

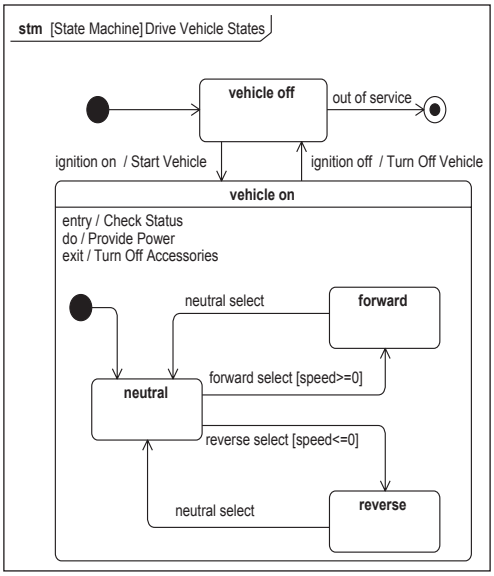

FIGURE 4.8

A state machine diagram that shows the *Drive Vehicle States* and the transitions between them.

This state machine can be owned by the *Vehicle* block, in which case it resides in the same package as the *Vehicle* block, or it can be owned by the vehicle's *Behavior* package and reside within that package.

A state machine can specify the lifecycle behavior of a block in terms of its discrete states and transitions, and is often used with sequence and/or activity diagrams, as shown in this example. State machines have many other features, which are described in Chapter 11, including support for multiple regions to describe concurrent behaviors and additional transition semantics.

4.3.9 *VEHICLE CONTEXT* USING AN INTERNAL BLOCK DIAGRAM

The *Vehicle Context* Diagram is shown in Figure 4.9. The diagram shows the interfaces between the *Vehicle,* the *Driver,* and the *Physical Environment* (i.e., *Road*, *Atmosphere,* and *External Entity*) that were defined in the block definition diagram in Figure 4.3. The *Vehicle* directly interfaces with the *Driver*, the *Atmosphere*, and the *Road.* The *Driver* interfaces with the *External Entities* such as a traffic light or another vehicle via the *Sensor Input* to the *Driver*. However, the *Vehicle* does not directly interface with the *External Entities.* The multiplicity on the *External Entity* is consistent with the multiplicity shown in the block definition diagram in Figure 4.3.

This context diagram is an **internal block diagram** that shows how the **parts** of the *Automobile Domain* block from Figure 4.3 are connected. It is called an internal block diagram because it represents the internal structure of a higher-level block, which in this case is the *Automobile Domain* block. The *Vehicle* **ports** are shown as the small squares on the boundary of the parts and specify interfaces with other parts. **Connectors** are shown as lines between the ports and define how parts connect to one another. Parts can also be connected without ports when the details of the interface are not of interest to the modeler as indicated by the connections to the *Atmosphere* and *External Entity*.

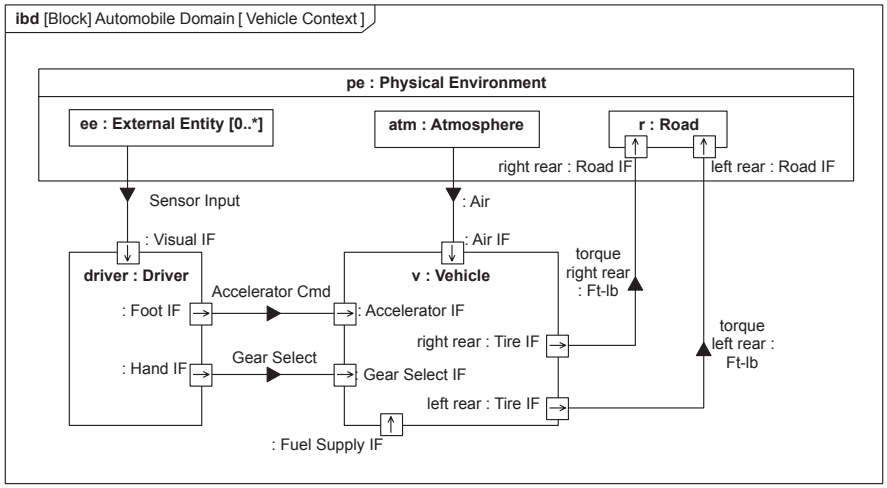

FIGURE 4.9

The internal block diagram for the *Automobile Domain* describes the *Vehicle Context*, which shows the *Vehicle* and its external interfaces with the *Driver* and the *Physical Environment* that were defined in Figure 4.3.

The external interfaces that enable the *Vehicle* to provide power are shown in Figure 4.9. The interfaces between the rear tires and the road are shown, since the *Vehicle* is assumed to be rear wheel drive. The interfaces to both rear tires are shown, because the power can be distributed differently to the left and right rear wheels depending on tire-to-road traction and other factors. The interfaces between the front tires and the road are not shown in this diagram. It is common modeling practice to present only the information relevant to the purpose of the diagram, even though additional information may be included in the model.

The black-filled arrowheads on the connector are called **item flows**. They represent the items flowing between parts. The items that flow may include mass, energy, and/or information. In this example, the *Accelerator Cmd* that was previously defined in the activity diagram in Figure 4.7 flows from the *Driver Foot IF* to the *Vehicle Accelerator IF,* and the *Gear Select* flows from the *Driver Hand IF* to the *Vehicle Gear Select IF.* The object flows that connect the inputs to the outputs on the activity diagram in Figure 4.7 can be **allocated** to the item flows on the connectors in the internal block diagram. Allocations are discussed as a general-purpose relationship for mapping one model element to another in Chapter 14.

SysML ports provide substantial capability to model interfaces. Ports can specify the items that can flow in or out of a part, and the services that are either required or provided by a part. The port provides a mechanism to integrate the behavior of the system with its structure by enabling access to a part's behavior and other features. (Refer to the discussion on ports in Chapter 7, Section 7.6.)

The internal block diagram enables the modeler to specify both the external and internal interfaces of a block and shows how its parts are connected. Details of how to connect parts on an internal block diagrams are described in Chapter 7, Section 7.3.

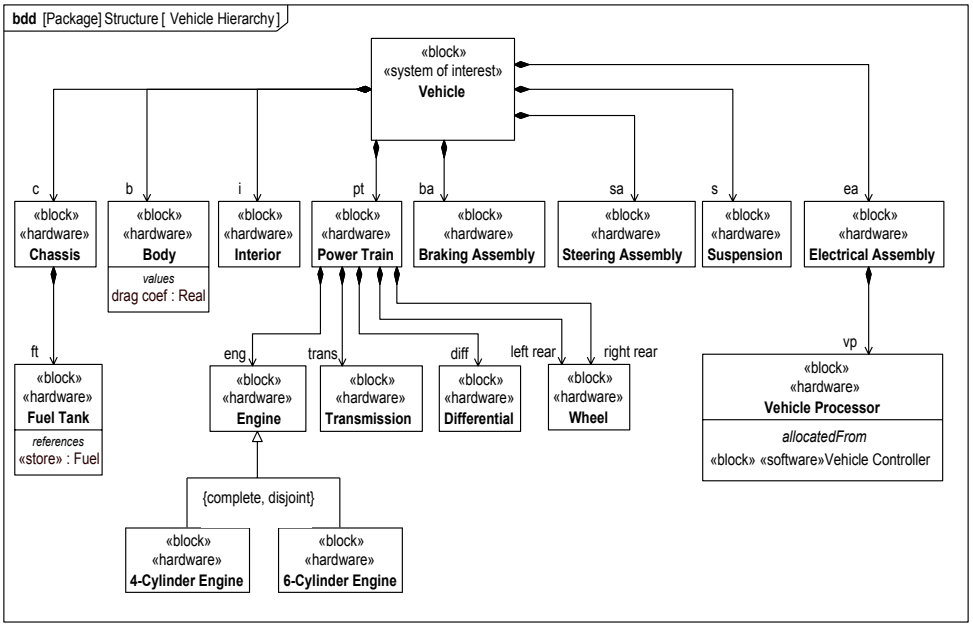

FIGURE 4.10

A block definition diagram of the *Vehicle Hierarchy* that shows the *Vehicle* and its components. The *Power Train* is further decomposed into its components, and the *Vehicle Processor* includes the *Vehicle Controller* software.

4.3.10 *VEHICLE HIERARCHY* REPRESENTED ON A BLOCK DEFINITION DIAGRAM

The example to this point has focused on specifying the vehicle in terms of its external interactions and interfaces. The *Vehicle* package shown in Figure 4.1 contains the description of the *Vehicle* and its parts in terms of its requirements, structure, and behavior. The *Vehicle* block is contained in the *Automobile Domain::Vehicle::Structure* package.

The *Vehicle Hierarchy* in Figure 4.10 is a block definition diagram that shows the decomposition of the *Vehicle* into its components. The *Vehicle* is composed of the *Chassis, Body, Interior, Power Train,* and other components. Each hardware component is designated as «hardware».

The *Power Train* is further decomposed into the *Engine, Transmission, Differential,* and *Wheel.* Note that the *right rear* and *left rear* indicate different usages of a *Wheel* in the context of the *Power Train.* Thus, each rear wheel has a different role and may be subject to different forces, such as is the case when one wheel loses traction. The front wheels are not shown in this diagram.

The *Engine* may be either 4 or 6 cylinders as indicated by the specialization relationship. The 4- and 6-cylinder engine configurations are alternatives being considered to satisfy the acceleration and fuel efficiency requirements. The *engine size* is *{complete, disjoint}*, which asserts that the 4- and 6-cylinder engines represent all possible engine types for this *Vehicle*, and that the 4- and 6-cylinder engines are mutually exclusive. (Note: This construct is called a **generalization set** and is not part of the SysML basic feature set.)

The *Vehicle Controller* «software» specifies a software component that is allocated to the *Vehicle Processor* as shown in its allocation compartment. In this example, the software controls many of the automobile engine and transmission functions to optimize engine performance and fuel efficiency, and the *Vehicle Processor* is the execution platform for the vehicle control software. The *allocatedFrom*, label indicates that the allocation is from the software to the processor.

The *Fuel* is shown in a *references* compartment of the *Fuel Tank* block. It is indicated as a reference because it is stored by the *Fuel Tank* but is not physically part of the *Fuel Tank*.

The internal vehicle interactions and interconnections between the components are represented in a way similar to the external *Vehicle* interactions and interconnections described above. The modeling artifacts for this next lower level of design are used to specify the components of the *Vehicle* system as described in the next sections.

4.3.11 ACTIVITY DIAGRAM FOR *PROVIDE POWER*

The activity diagram in Figure 4.7 shows that the vehicle must *Provide Power* in response to the driver accelerator command and generate *torque out* at the road surface. The *Provide Power* activity diagram in Figure 4.11 shows how the vehicle components generate this torque.

The external inputs to the activity include the *:Accelerator Cmd* and *:Gear Select* from the *Driver,* and *:Air* from the *Atmosphere* to support engine combustion. The outputs from the activity are the *torque right rear* and *torque left rear* from the right and left rear wheels respectively to the road to accelerate the *Vehicle.* The inputs and outputs for the *:Provide Power* action in Figure 4.7 are elaborated as a result of further refinement of the model, and now include *:Air* as an input, and torque from each rear wheel. Some of the other inputs and outputs, such as exhaust from the engine, are not included for simplicity. The activity partitions represent usages of the vehicle components shown in the block definition diagram in Figure 4.10.

The *Vehicle Controller* accepts *Driver* inputs including the *:Accelerator Cmd* and *:Gear Select*, and provides outputs to the *Engine* and *Transmission*. The *Fuel Tank* stores and dispenses the *:Fuel* to the *Engine.* The *:Fuel-Air Cmd* from the *Vehicle Controller* and *:Air* from the *Atmosphere* are inputs to the *Generate Torque* action. The engine torque is input to the *Amplify Torque* action performed by the *Transmission.* The amplified torque is input to the *Distribute Torque* action performed by the *Differential,* which distributes torque to the right and left rear wheels. The wheels *Provide Traction* to the road surface to generate the torque to accelerate the *Vehicle.* The *Differential* monitors and controls the difference in torque to the rear wheels. If one of the wheels loses traction, the *Differential* sends a *Loss of Traction* signal to the braking system to adjust braking. The *Loss of Traction* signal is sent using a send signal action.

A few other items are worth noting in this example. The flows are shown to be continuous for all but the *Gear Select.* The inputs and outputs continuously flow in and out of the actions. *Continuous* means that the delta time between arrival of the inputs or outputs approaches zero. Continuous flows build on the concept of streaming inputs and output parameters, which means that the inputs are accepted and outputs are produced while the action is executing. Conversely, nonstreaming inputs are only available prior to the start of the action execution, and nonstreaming outputs are produced only at the completion of the action execution. The ability to represent streaming and continuous flows adds a significant capability to classic behavioral modeling using functional flow diagrams. The continuous flows are assumed to be streaming but this is not shown in the diagram. (Note: Continuous and streaming are not part of the basic feature set.)

Modeling of activities provides the capability to specify behavior precisely in terms of the flow of control and data. This is explained in Chapter 9.

4.3.12 INTERNAL BLOCK DIAGRAM FOR THE *POWER SUBSYSTEM*

The previous activity diagram describes how the parts of the system interact to *Provide Power.* The parts of the system are represented by the activity partitions in the activity diagram. The internal block diagram for the *Vehicle* in Figure 4.12 shows how the parts are interconnected via their ports to achieve this functionality. This is a structural view of the system, as opposed to the behavioral view that was expressed in the activity diagram.

The internal block diagram shows the *Power Subsystem* that includes the parts of the *Vehicle* that interact to *Provide Power.* The frame of the diagram corresponds to the *Vehicle* black box. The ports on the diagram frame in Figure 4.12 correspond to the same ports shown on the *Vehicle* in the *Vehicle Context* diagram in Figure 4.9. The external interfaces are preserved as the internal structure of the *Vehicle* is further elaborated.

The *Engine, Transmission, Differential, right rear*:*Wheel* and *left rear:Wheel, Vehicle Processor,* and *Fuel Tank* are interconnected via their ports. The *Fuel* is stored in the *Fuel Tank* as indicated by «store». *Fuel* is shown as a dashed rectangle to indicate that the fuel is not part of the *Fuel Tank*, but is referenced by it. Only selected item flows are shown on the connectors. The item flows are allocated from the inputs and outputs on the *Provide Power* activity diagram in Figure 4.11.

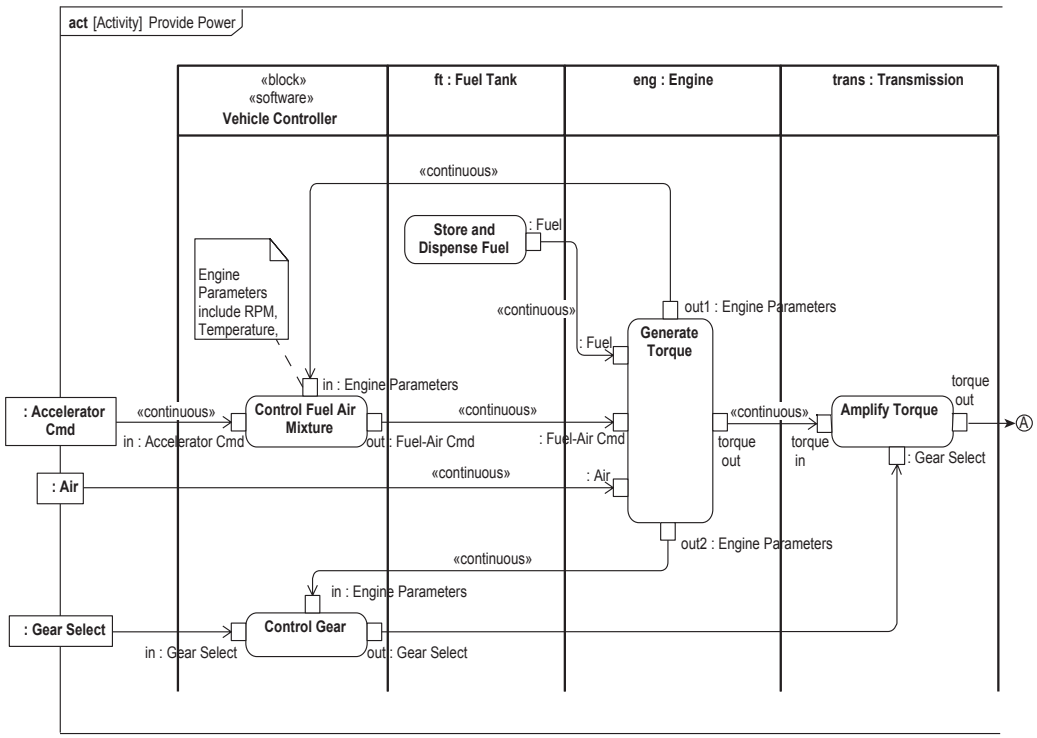

FIGURE 4.11 Cont'd

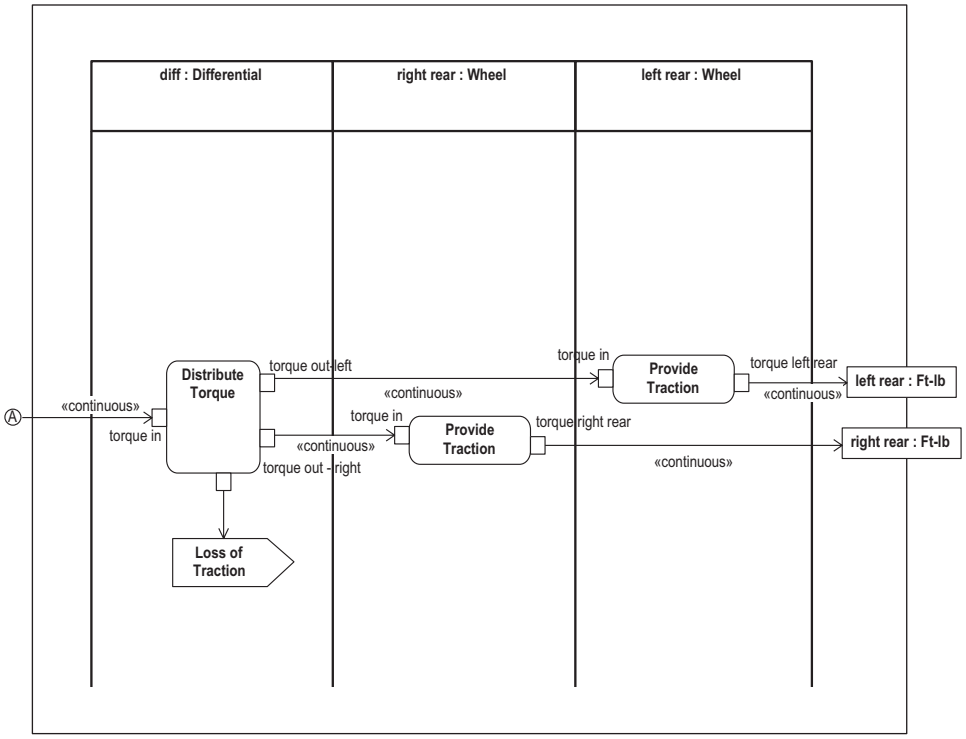

FIGURE 4.11

The activity diagram for *Provide Power* shows how the *Vehicle* components generate the torque to move the vehicle. This activity diagram realizes the *Provide Power* action in Figure 4.7 with activity partitions that correspond to the components in Figure 4.10.

Each subsystem can be expressed in a way similar to the *Power Subsystem* to realize specific functionality, such as braking and steering. The enclosing frame for each internal block diagram can be the same *Vehicle* block, but each diagram shows only the parts relevant to the particular subsystem. This approach can be used to present a subsystem view of the vehicle's internal structure. As an example, to express an internal block diagram for a steering subsystem, additional components would need to be defined beyond those shown on the block definition diagram in Figure 4.10, including the steering wheel, steering column, power steering pump, steering linkage, and front wheels. A composite view of all of the interconnected parts for all subsystems can also be presented on a single internal block diagram, but this would likely contain so much information that it would not communicate effectively.

An important concept in SysML is the distinction between **definition** and **usage**. Certain kinds of model elements, such as blocks, can be defined one time, but their usage in different contexts can be uniquely identified. In Section 4.3.10, the *right rear* and *left rear* are described as different usages of a *Wheel* in the context of the *Power Train*. A block represents the generic definition of the part, and the part represents a usage of a block in a particular context. More formally, a block is the type of the part, and a part is typed by a block.

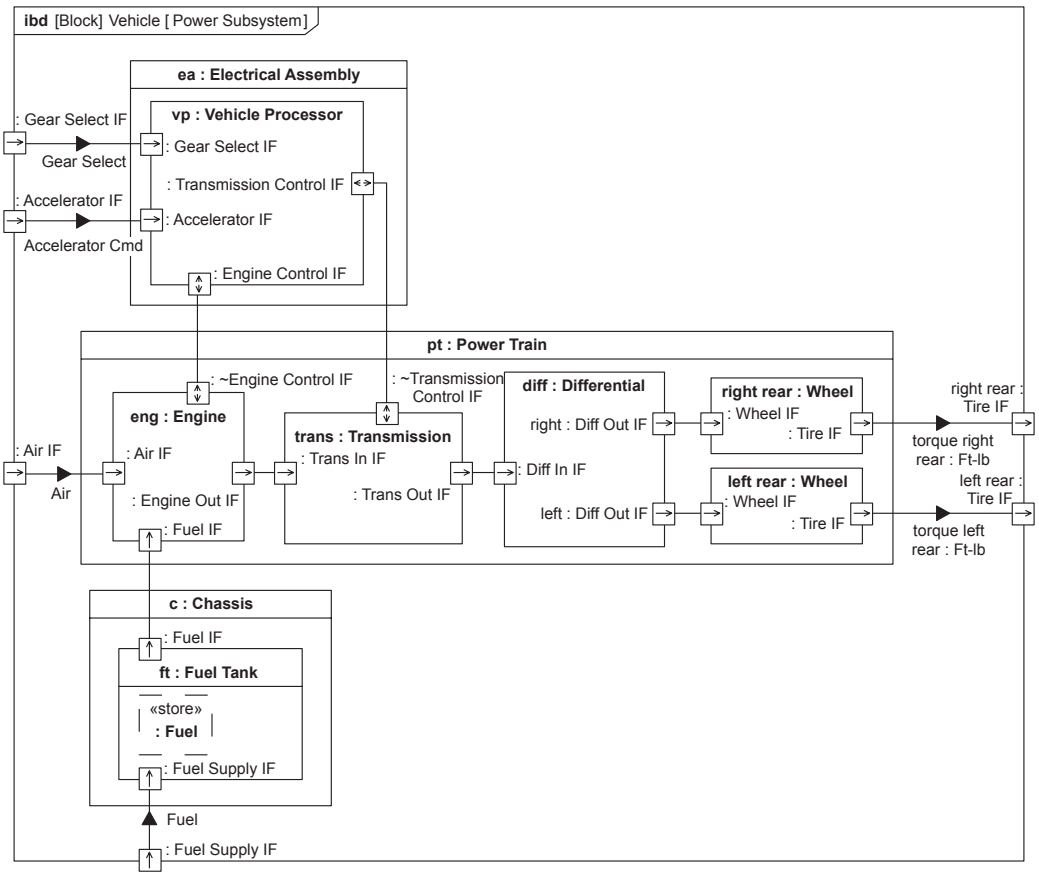

FIGURE 4.12

The internal block diagram for the *Power Subsystem* shows how the parts of the *Vehicle* that *Provide Power* are interconnected. The parts interact as specified by the activity diagram in Figure 4.11.

In Figure 4.10 and Figure 4.12, the *right rear* and *left rear* are different parts that represent distinct usages of *Wheel* in the context of the *Power Train.* Each usage of the block requires a composition relationship on the block definition diagram, such as the right rear wheel and left rear wheel in Figure 4.10. The colon (:) notation is used in Figure 4.12 to distinguish the part (i.e., usage) from the block (i.e., definition). The name to the right of the colon, *Wheel,* is the block. The names to the left of the colon, *right rear* and *left rear,* are particular parts or usages of *Wheel.* By convention, the usage names begin in lower case and the definitions begin with upper case.

A part enables the same block, such as a *Wheel*, to be reused in different contexts and be uniquely identified by its usage, such as *right rear* and *left rear*. Each part may be further redefined to have behaviors, value properties, and constraints that apply to its particular usage.

The concept of definition and usage applies to parts and blocks, but also applies to many other SysML language constructs. One example is that item flows can have both a definition and usage. For example,

the item flow entering the fuel tank in Figure 4.12 can be *in: Fuel* and the item flow exiting the fuel tank can be *out : Fuel*. Both flows are defined by *Fuel*, but in and out represent different usages of *Fuel* in the *Vehicle* context (Note: the usages are not shown in the figure). The ports on blocks and pins on actions can also have definitions that specify detailed interface information that can be reused. As an example, the interface that enables the flow of 110 volt 60 cycle power can be defined one time and reused. For the *Automobile* example, most of the pins and ports have been typed and are contained in the *IO Definitions* package.

As mentioned previously, Chapter 7 provides the detailed language description for both block definition diagrams and internal block diagrams, and the key concepts for modeling blocks, parts, ports, and connectors.

4.3.13 DEFINING THE EQUATIONS TO ANALYZE VEHICLE PERFORMANCE

Critical requirements for the design of this automobile are to accelerate from 0 to 60 mph in less than 8 seconds, while achieving a fuel efficiency of greater than 25 miles per gallon. These two requirements impose conflicting requirements on the design space, because increasing the maximum acceleration capability of the vehicle can result in a design with lower fuel efficiency. Two alternative configurations (4- and 6-cylinder engine) are evaluated to determine which configuration is the preferred solution to meet the acceleration and fuel efficiency requirements.

The *4-Cylinder Engine* and *6-Cylinder Engine* alternatives are shown in the *Vehicle Hierarchy* in Figure 4.10. There are many possible impacts to the automobile design that may result from the selection of different engines, such as the impact on vehicle weight, body shape, and electrical power. This simplified example only considers some of the impacts on the *Power Subsystem.* The vehicle controller is assumed to control the fuel and air mixture. It also controls when the automatic transmission changes the gear to optimize engine and overall performance.

The *Analysis Context* block definition diagram in Figure 4.13 is used to define the equations for these analyses. This diagram introduces another kind of block called a **constraint block**. Instead of defining systems and components, the constraint block defines constraints in terms of reusable equations and their **parameter** definitions that can be used by one or more analyses.

In this example, the *Vehicle Acceleration Analysis* block is in the *Parametrics* package, as indicated by the diagram header, and comprises several constraint blocks that are used to analyze the vehicle acceleration. This analysis is performed to determine whether either the 4- or 6-cylinder vehicle configuration can satisfy its acceleration requirement. The constraint blocks define generic equations for *Gravitational Force*, *Drag Force*, *Power Train Force*, *Total Force*, *Acceleration*, and an *Integrator*. The *Total Force* equation, as an example, shows that *ft* is the sum of *fi*, *fj*, and *fk*. Note that the parameters are defined along with their units in the constraint block.

The *Power Train Force* is further decomposed into other constraint blocks that express the torque equations for the *Engine, Transmission, Differential,* and *Wheels.* The equations are not explicitly defined, but the critical parameters of the equations are identified. It is often useful in the early stages of an analysis to identify the critical parameters but defer definition of the equations until the detailed analysis is performed.

The *Vehicle Acceleration Analysis* block also references the *Automobile Domain* block that was originally shown in the block definition diagram in Figure 4.3. The *Automobile Domain* is the subject of the analysis. By referencing the *Automobile Domain*, the value properties of the *Vehicle* and the *Physical Environment* can be accessed and bound to the parameters of the generic equations, as described in the next section.

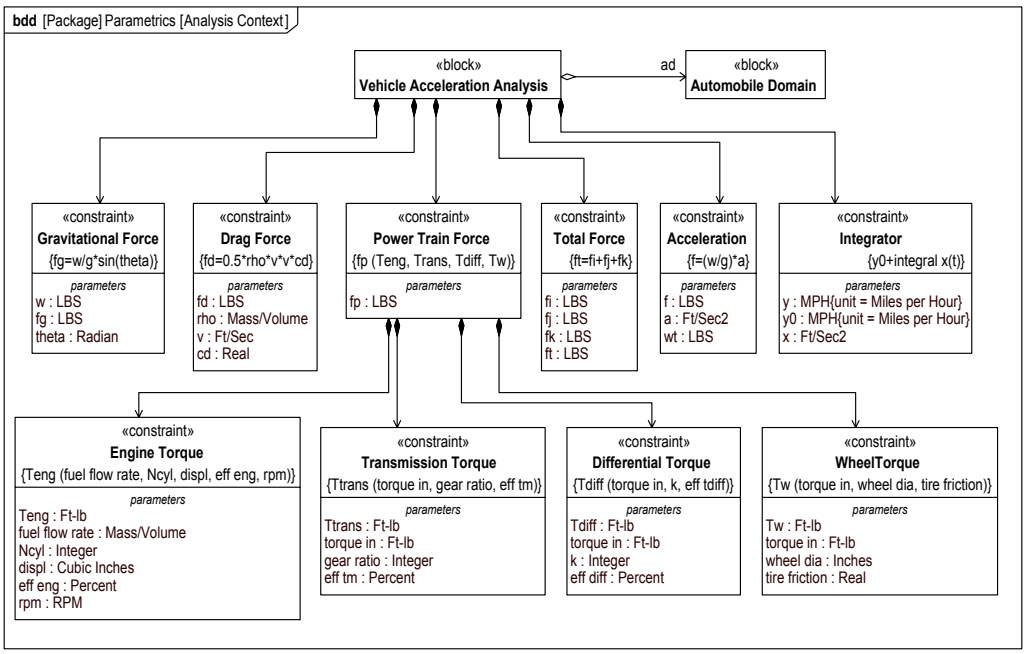

FIGURE 4.13

The block definition diagram for the *Analysis Context* that defines the equations for analyzing the vehicle acceleration requirement. The equations and their parameters are specified using constraint blocks. The *Automobile Domain* block from Figure 4.3 is referenced since it is the subject of the analysis.

4.3.14 ANALYZING VEHICLE ACCELERATION USING THE PARAMETRIC DIAGRAM

The previous block definition diagram defined the equations and associated parameters needed to analyze the system. The **parametric diagram** in Figure 4.14 shows how these equations are used to analyze the time for the *Vehicle* to accelerate from 0 to 60 mph and satisfy the maximum acceleration requirement. The diagram frame corresponds to the *Vehicle Acceleration Analysis* block from the block definition diagram in Figure 4.13.

The parametric diagram shows a network of constraints. Each constraint is a usage of a constraint block defined in the block definition diagram in Figure 4.13. The equations for some of the constraints are shown on this parametric diagram. The parameters of the equations are shown as small rectangles flush with the inside boundary of the constraint.

A parameter in one equation can be bound to a parameter in another equation by a **binding connector**. An example of this is the parameter *ft* in the *Total Force* equation, which is bound to the parameter *f* in the *Acceleration* equation. This means that *ft* in the *Total Force* equation is equal to *f* in the *Acceleration* equation.

The parameters can also be bound to **value properties** of blocks to equate the parameter of an equation to a value property of the system or environment. The value properties are shown as rectangles nested within the *ad:Automobile Domain*. An example is the binding of the coefficient of drag

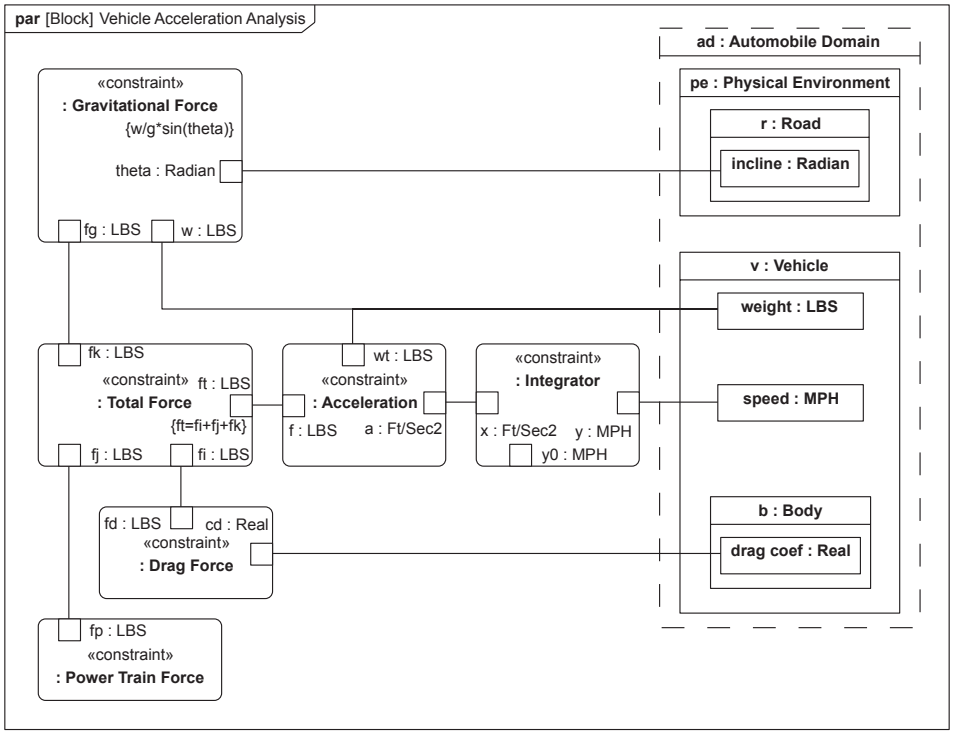

FIGURE 4.14

The parametric diagram that uses the equations defined in Figure 4.13 to analyze vehicle acceleration. The parameters of the equations are bound to other parameters and to value properties of the *Vehicle* and its *Physical Environment*, some of which were defined in Figure 4.3.

parameter *cd* in the *Drag Force* equation to the drag property called *drag coef,* which is a value property of the vehicl*e Body*. Sometimes it is more convenient not to show the nested parts but identify the value properties using the dot notation. The drag coefficient would be shown as *ad.v.b.drag coef* to indicate that this is a value property of the body *b*, which is part of the vehicle *v* that is part of the Automobile Domain *ad.* Another example is the binding of the road *incline* angle to the angle *theta* in the gravity force equation. This binding enables values of parameters of generic equations to be set equal to values of specific value properties of the blocks. In this way, generic equations can be reused to analyze different designs by binding the parameters of the generic equations to value properties of different designs.

The parametric diagram and related modeling information can be used to specify an analysis that is executed in separate simulation or analysis tools as describe in Chapter 18, Sections 18.2.2 and 18.4. The simulation or analysis tools can be used to perform sensitivity analysis and determine the property values that are required to satisfy the acceleration requirements. In this example, only some of the vehicle properties are shown. However, a more complete depiction would show the binding of other vehicle value properties to other constraint parameters. Although not shown in Figure 4.14, the *Power*

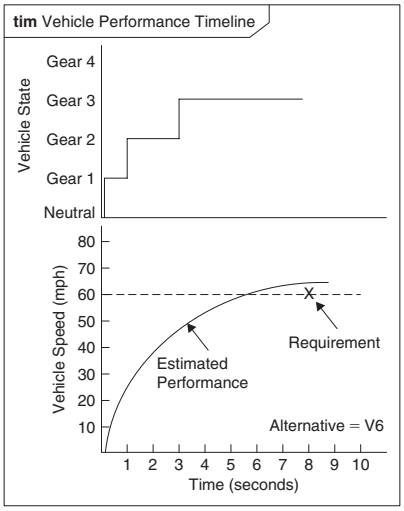

FIGURE 4.15

Analysis results from executing the constraints in the parametric diagram in Figure 4.14, showing the *Vehicle Speed* and *Vehicle State* as a function of time. This is captured in a UML timing diagram.

Train Force constraint includes nested constraints consistent with the constraint blocks that compose it from the *Analysis Context* block definition diagram in Figure 4.13.

In addition to the acceleration and fuel efficiency requirements, other analyses may address requirements for braking distance, vehicle handling, vibration, noise, safety, reliability, production cost, and others. These analyses can be performed to determine the required property values of the system components (e.g., *Body, Chassis, Engine, Transmission, Differential, Brakes, Steering Assembly*) to satisfy the overall system requirements. The parametrics enable the critical value properties of the system design to be identified and integrated with parameters in the analytical models. Details of how to model constraint blocks and their usages in parametric diagrams are described in Chapter 8.

4.3.15 ANALYSIS RESULTS FROM ANALYZING VEHICLE ACCELERATION

As mentioned in the previous section, the parametric diagram is expected to specify an analysis that is executed in an engineering analysis tool to provide the analysis results. This may be a separate specialized analysis tool, such as a simple spreadsheet or a high-fidelity performance simulation, or it may be a capability that the SysML modeling tool provides. The results from the execution then provide values that can be used to update the value properties in the SysML model.

The analysis results from executing the constraints in the parametric diagram are shown in Figure 4.15. This example uses the **UML timing diagram** to display the results. The timing diagram is not one of the SysML diagram kinds. It can be used with SysML, along with other more robust visualization methods such as response surfaces, to show multi-parameter relationships. In this timing diagram, the *Vehicle Speed* is shown as a function of time, and the *Vehicle State* is shown as a function of time. The *Vehicle* states correspond to nested states within the *forward* state in Figure 4.8. Based on the analysis performed, the 6-cylinder (V6) vehicle configuration is able to satisfy its acceleration

requirement. A similar analysis showed that the 4-cylinder (V4) vehicle configuration does not satisfy the requirement.

4.3.16 DEFINING THE *VEHICLE CONTROLLER* ACTIONS TO OPTIMIZE ENGINE PERFORMANCE

The analysis results showed that the V6 configuration is needed to satisfy the vehicle acceleration requirement. Additional analysis is needed to assess whether the V6 configuration can satisfy the fuel efficiency requirement of a minimum of 25 miles per gallon under the stated driving conditions, as specified in the *Fuel Efficiency* requirement in Figure 4.2.

The activity diagram to *Provide Power* in Figure 4.11 is used to support the analysis needed to optimize fuel efficiency and engine performance. The :*Vehicle Controller* «software» is allocated to the *Vehicle Processor*, as described in Section 4.3.10, and includes an action to *Control Fuel Air Mixture* that controls the engine accelerator command. The inputs to this action include the *Accelerator Cmd* from the *Driver* and *Engine Parameters* such as revolutions per minute (rpm) and engine temperature from the *Engine*. The *Vehicle Controller* also includes the *Control Gear* action to determine when to change gears based on engine speed (i.e., rpm) to optimize performance and fuel efficiency. The specification of the *Vehicle Controller* software can include a state machine diagram that changes state in response to the inputs consistent with the state machine diagram in Figure 4.8.

The specification of the algorithms to realize the *Vehicle Controller* actions requires further analysis. The algorithm can be defined by further specifying the actions as mathematical and logical expressions that can be captured in a more detailed activity diagram or directly in code. A parametric diagram can also be developed to specify the algorithm performance requirements that constrain the input and output of the *Vehicle Controller* actions. For example, the constraints may specify the required fuel and air mixture as a function of rpm and engine temperature to achieve optimum fuel efficiency. The algorithms are used to control fuel flow rate and air intake, and perhaps other parameters, to satisfy these constraints. Based on the engineering analysis, whose details are omitted here, the V6 engine is able to satisfy the fuel efficiency requirements as well as the acceleration requirements, and is selected as the preferred vehicle system configuration.

4.3.17 SPECIFYING THE *VEHICLE* AND ITS COMPONENTS

The block definition diagram in Figure 4.10 defined the blocks for the *Vehicle* and its components. The model is used to specify the *Vehicle* and each of its components in terms of the functions they perform, their interfaces, and their performance and physical properties. Other aspects of the specification may include a state machine to represent the state-based behavior of the system and its components, and specification of the items that are stored by the system and its components, such as fuel in the fuel tank or data in computer memory.

A simple example is the specification of the *6-Cylinder Engine* block shown on the block definition diagram in Figure 4.16. The *Engine* block and the *6-Cylinder Engine* block were originally shown in the *Vehicle Hierarchy* block definition diagram in Figure 4.10.

In this example, the *Engine* hardware element performs a function called *generate torque*, which is shown as an operation of the block in the operations compartment. This operation corresponds to the *Generate Torque* action in Figure 4.11. The ports on the *Engine* specify its interfaces

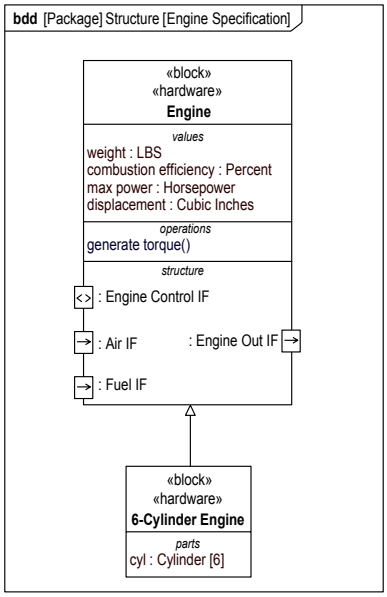

FIGURE 4.16

A block definition diagram that shows the *Engine* block and the features used to specify the block. This block was previously shown in the *Vehicle Hierarchy* block definition diagram in Figure 4.10.

as *Air IF, Fuel IF, Engine Control IF,* and *Engine Out IF.* Selected value properties of the engine are shown in the values compartment that represent its performance and physical properties, including its *displacement, combustion efficiency, max power,* and *weight*. Each value property is typed by a **value type** that specifies its data type (e.g., integer, real) and units (e.g., *Percent*, *Cubic Inches*).

The *6-Cylinder Engine* block is a subclass of the generic *Engine* block and inherits all of the features from *Engine*. However, the *6-Cylinder Engine* is a specialized engine that contains six *Cylinders*, as indicated in its parts compartment. In addition, the *6-Cylinder Engine* may define values for each value property contained in the generic *Engine,* such as the *max power* and *weight.* This information is derived from the parametric analysis discussed in Sections 4.3.13–15.

Other components of the vehicle can be specified in a similar way. If desired, text requirements can be written to correspond to the functional, interface, performance, and physical requirements associated with each block to create traditional text specifications from the model.

4.3.18 REQUIREMENTS TRACEABILITY

The *Automobile System Requirements* were shown in Figure 4.2. Capturing the text-based requirements in the SysML model provided the means to establish traceability between the text-based requirements and other specification, design, analysis, and verification elements of the model.

The requirements traceability for the *Maximum Acceleration* requirement is shown in Figure 4.17. This requirement traces to a *Market Analysis*, which was conducted in support of the system

requirements analysis. The requirement is satisfied by the *Provide Power* activity that was shown in Figure 4.11. The *Max Acceleration* **test case** is also shown as the method to verify that the requirement is satisfied. In addition, the *Engine Power* requirement is derived from the *Maximum Acceleration* requirement and contained in the *Engine Specification.* The **rationale** for deriving the requirement refers to the *Vehicle Acceleration Analysis* parametric diagram in Figure 4.14. The *6-Cylinder Engine* block refines the *Engine Specification* by more precisely expressing the text requirements. The above relationships enable traceability from the system requirements with the supporting rationale to the system design, test cases, and analysis.

The direction of the arrows points from the *Provide Power* activity, *Max Acceleration* test case, and *Engine Power* requirement to the *Maximum Acceleration* as the source requirement. This is in the opposite direction from what is traditionally used to depict requirements flow-down. The direction reflects the

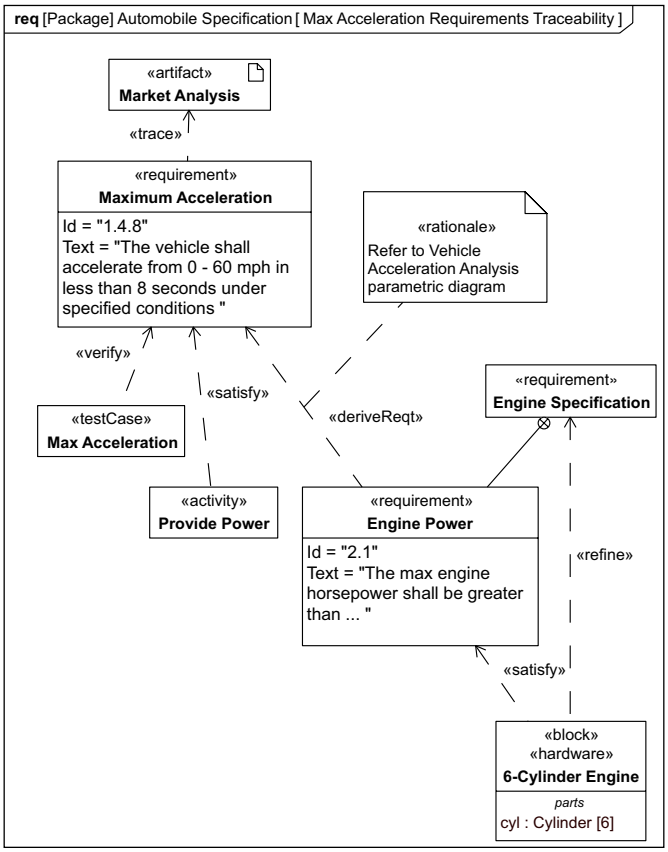

FIGURE 4.17

The requirement diagram showing the traceability of the *Maximum Acceleration* requirement that was displayed in the *Automobile Specification* in Figure 4.2. The traceability to a text-based requirement includes the design elements to satisfy it, other requirements derived from it, and a test case to verify it. Rationale for the deriveReqt relationship based on parametric analysis is also shown.

dependency of the design, test case, and derived requirement on the source requirement, such that if the source requirement changes, the design, test case, and derived requirement may also need to change.

The requirements are supported by multiple notation options including the direct, callout, and tabular presentation. Details of how SysML requirements and their relationships are modeled are described in Chapter 13.

4.3.19 VIEW AND VIEWPOINT

SysML includes the concept of **view** and **viewpoint** to reflect perspectives of different stakeholders. In Figure 4.18, the *Architect* and *Regulator* viewpoints reflect perspectives of the *System Architect* and *National Highway Traffic Safety Administration* stakeholders, respectively. These viewpoints include identification of the **stakeholders**, purpose, language, and methods for constructing a view of the model to address their concerns. In this example, the *System Architect* is concerned about the fuel economy versus acceleration trade-offs, and the *Government Regulator* is concerned about the vehicle's ability to meet safety requirements. The view is constructed by performing a query of the model that is specified by the viewpoint method and then presenting this information in a specified format. As indicated in the figure, the *Vehicle Performance* view **conforms** to the *Architect* viewpoint by providing traceability to the fuel efficiency and acceleration requirements and the associated design rationale in a requirements diagram. The *Vehicle Safety Regulations* view conforms to the *Regulator* viewpoint by providing the safety requirements, test cases, and test results in tabular format. The modeling tool can provide the query results to a rendering application to present the information in different formats, including documents with text, diagrams, tables, and plots.

Further details on modeling view and viewpoints can be found in Chapter 5, Section 5.6 and Chapter 15, Section 15.8.

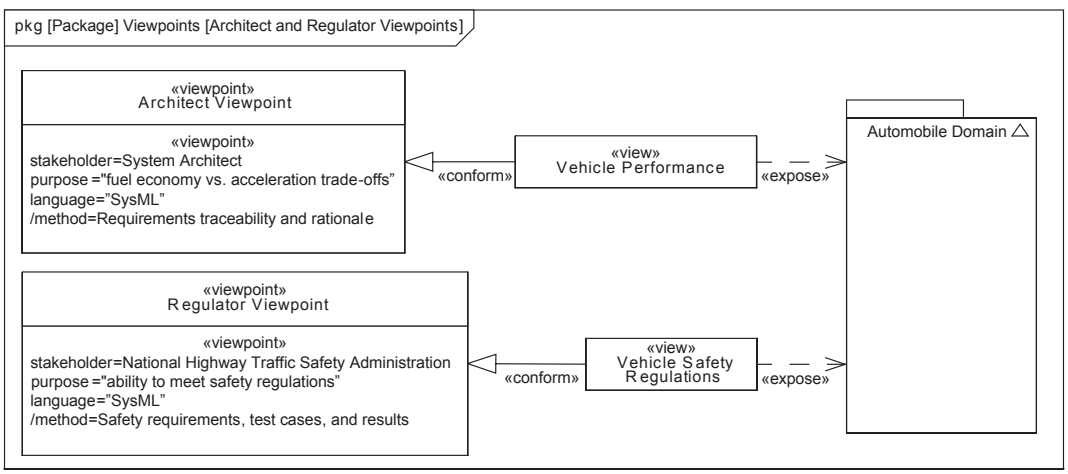

FIGURE 4.18

The package diagram showing the *Architect* viewpoint to address concerns related to fuel economy versus acceleration trade-offs, and a *Regulator* viewpoint to address concerns related to meeting safety requirements.

4.4 MODEL INTERCHANGE

An important aspect of systems modeling is the ability to exchange model information among tools. A SysML model that is captured in a model repository can be imported and exported from a SysML-compliant tool in a standard format called **XML metadata interchange** (XMI). This enables other tools to exchange this information if they also support XMI. Examples may be the ability to export selected parts of the SysML model to a UML tool to support software development of the *Vehicle Controller* software, or to import and export the requirements from a requirements management tool, or to import and export the parametric diagrams and related information to engineering analysis tools. The ability to achieve seamless model interchange capability may be limited by the quality of the model and by the limitations of tool conformance with the standard. Other interchange mechanism may use the tool's application programming interface (API) to access model information. Chapter 18, Section 18.3, includes a description of XMI and other data exchange mechanisms.

4.5 SUMMARY

The SysML basic feature set is a subset of the language features that applies to all nine SysML diagrams. It provides significant capability for representing systems, without introducing all of the language complexity associated with the full feature set. The basic feature set is required learning for the first two levels of SysML certification, called the Model User and Model Builder-Fundamental levels.

The automobile example demonstrates how a SysML model using the basic feature set can help to specify, design, analyze, and verify a system. It enables the requirements, behavior, structure, and parametric aspects of the system to be represented in a precise, consistent, and comprehensive manner. It is also clear from the example that the modeler must apply a systematic method to build a system model that addresses the modeling objectives associated with its intended use.

4.6 QUESTIONS

1. Show how a stopping distance requirement would be captured in Figure 4.2.

In the following questions, assume a change in the stopping distance is required.

2. Would you anticipate any changes to the block definition diagram in Figure 4.3?
3. Would you anticipate any significant changes to the use case diagram in Figure 4.4?
4. Would you anticipate any significant changes to the sequence diagram in Figure 4.5?
5. Describe an activity diagram analogous to Figure 4.7 to address the braking requirements.
6. Describe an internal block diagram analogous to Figure 4.9 to address the braking requirements.
7. Describe additions to the vehicle hierarchy in Figure 4.10 to address the braking requirements.
8. Describe an activity diagram analogous to Figure 4.11 to address how vehicle braking is performed.
9. Describe an internal block diagram analogous to Figure 4.12 for the vehicle braking subsystem.

10. Describe a block definition diagram analogous to Figure 4.13 to define the equations needed to analyze vehicle braking distance performance.
11. Describe a parametric diagram analogous to Figure 4.14 to describe the analysis used to analyze braking distance performance.

DISCUSSION TOPICS

What are some observations about the changes to the model that occur as a result of a requirements change such as the one described above (i.e., change in stopping distance)?