CHAPTER

SysML Language Overview 3

This chapter provides an overview of SysML that includes a simple example showing how the language is applied to the system design of an automobile that was introduced in Chapter 1. The example includes references to chapters in Part II that provide a detailed description of the diagrams and language concepts. Part III includes two more detailed examples of how MBSE methods can be used with SysML to specify and design a system.

3.1 SysML Purpose and Key Features

SysML 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 its components that can then be designed using other domain-specific languages such as UML for software design and VHDL for hardware design.

SysML can represent systems, components, and other entities as follows:

- 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 (e.g., functions allocated to components)
- 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:

- *Requirement diagram* represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability (not in UML)

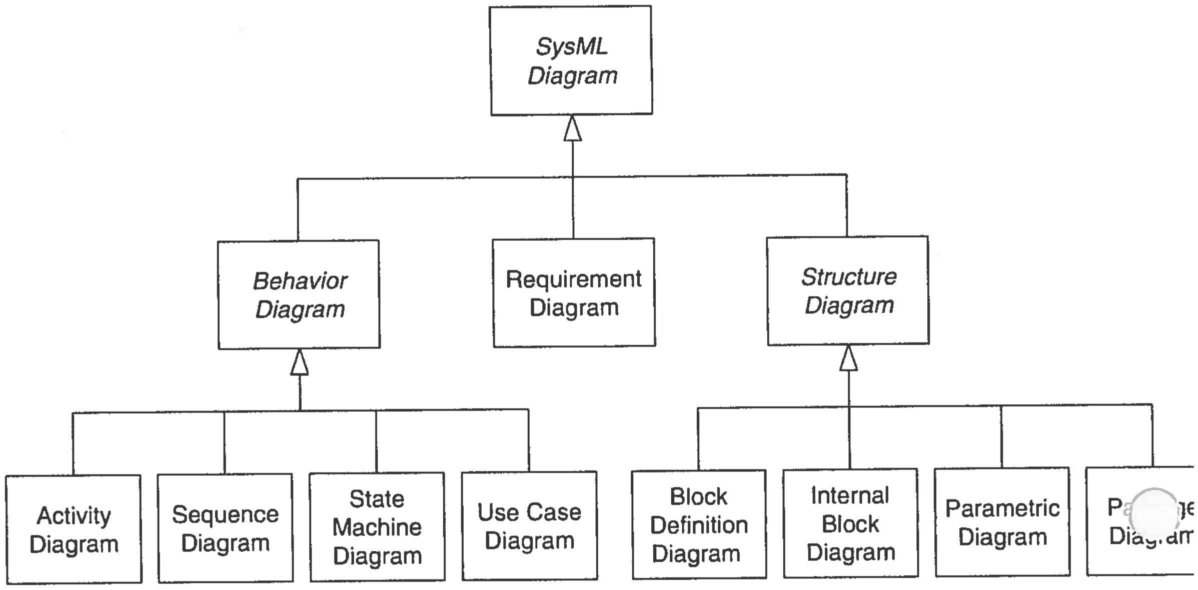

FIGURE 3.1

SysML diagram taxonomy.

- *Activity diagram* represents behavior in terms of the ordering of actions based on the availability of 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 parts (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 or other entity 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)
- *Package diagram* represents the organization of a model in terms of packages that contain model elements (same as UML package diagram)

A diagram graphically represents a particular aspect of the system model as described in Section 2.1.2. The diagram type constrains the type of model elements and associated symbols that can appear on a diagram. For example, an activity diagram can include diagram elements that represent actions, control flow, and input/output 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. Tabular representations are also supported in SysML as a complement to diagram representations to capture model information such as allocation tables.

3.3 Using SysML in Support of MBSE

SysML provides a means to capture the system modeling information as part of an MBSE approach without imposing a specific method on how this is performed. The selected method determines which activities are performed, the ordering of the activities, and which modeling artifacts are created to represent the system. For example, traditional structured analysis methods can be used to decompose the functions and allocate the functions to components. Alternatively, one can apply a use case driven approach that derives functionality based on scenario analysis and associated interactions among parts. The two methods may produce different combinations of diagrams in different ways to represent the system specification and design.

A typical use of the language may include one or more iterations of the following activities to specify and design the system:

- Capture and analyze black box system requirements
  - Capture text-based requirements in a requirements management tool
  - Import requirements into the SysML modeling tool
  - Identify top-level functionality in terms of system use cases
  - Capture the traceability between the use cases and requirements
  - Model the use case scenarios as activity diagrams, sequence diagrams, and/or state machine diagrams
  - Create the system context diagram
  - Identify system test cases to support system verification
- Develop one or more candidate system architectures to satisfy the requirements
  - Decompose the system using the block definition diagram
  - Define the interaction among the parts using activity or sequence diagrams
  - Define the interconnection among the parts using the internal block diagram
- Perform engineering and trade-off analysis to evaluate and select the preferred architecture
  - Capture the constraints on system properties using the parametric diagram to support analysis of performance, reliability, cost, and other critical properties
  - Perform the engineering analysis to determine the budgeted values of the system properties (typically done in separate engineering analysis tools)

- Specify component requirements and their traceability to system requirements
  - Capture the functional, interface, and performance requirements for each component (block) in the architecture
  - Trace component requirements to the system requirements
- Verify that the system design satisfies the requirements by executing system-level test cases

Other systems engineering activities are performed in conjunction with the preceding modeling activities such as configuration management and risk management. Detailed examples of how SysML can be used to support two different MBSE methods are included in the modeling examples in Part III. A simplified example is described next.

3.4 A Simple Example Using SysML for an Automobile Design

The following example was introduced in Chapter 1 without introducing the model-based approach. It is a simplified example that illustrates how SysML can be applied to specify and design a system.

3.4.1 Example Background and Scope

The example includes at least one diagram for each SysML diagram type, but only highlights selected features of the language. It includes multiple references to the chapters in Part II for the detailed language description. The way the diagrams are used to represent the system and the ordering of the diagrams is intended to be representative of applying a typical model-based approach. However, this will vary depending on the specific process and method used.

3.4.2 Problem Summary

The sample problem describes the use of SysML as it applies to the design of an automobile. A marketing analysis that was done indicated the need to increase the automobile's acceleration and fuel efficiency from its current capability. A variant of the process described in Section 3.3 is used to design the system to satisfy the requirements. In this simplified example, selected aspects of the design are considered to support an initial trade-off analysis. The trade-off analysis included evaluation of alternative vehicle configurations that included a 4-cylinder engine and a 6-cylinder engine to determine whether they can satisfy the acceleration and fuel efficiency requirement.

In addition, the proposed vehicle design includes a vehicle controller and associated software to control the fuel-air mixture and maximize fuel efficiency and engine performance. Only a small subset of the design is addressed to

Table 3.1 Diagrams Used in the Automobile Example

Figure	Diagram Kind	Diagram Name
3.2	Requirement diagram	Automobile System Requirements
3.3	Block definition diagram	Automobile Domain
3.4	Use case diagram	Operate Vehicle
3.5	Sequence diagram	Drive Vehicle
3.6	Sequence diagram	Start Vehicle
3.7	Activity diagram	Control Power
3.8	State machine diagram	Drive Vehicle States
3.9	Internal block diagram	Vehicle Context
3.10	Block definition diagram	Vehicle Hierarchy
3.11	Activity diagram	Provide Power
3.12	Internal block diagram	Power Subsystem
3.13	Block definition diagram	Analysis Context
3.14	Parametric diagram	Vehicle Acceleration Analysis
3.15	Timing diagram (not SysML)	Vehicle Performance Timeline
3.16	Activity diagram	Control Engine Performance
3.17	Block definition diagram	Engine Specification
3.18	Requirement diagram	Max Acceleration Requirement Traceability
3.19	Package diagram	Model Organization

highlight the use of the language. The diagrams used in this example are shown in Table 3.1.

SysML diagrams include a **diagram frame**. The **diagram header** in the diagram frame describes the kind of diagram, the diagram name, and some additional information that provides context for the **diagram content**. Detailed information on diagram frames, diagram headers, and other common diagram elements that apply to all SysML diagrams is described in Chapter 4.

The example includes the following user-defined notations, called a stereotype, that are added for this example. Chapter 14 describes how stereotypes are used to further customize the language for domain-specific applications.

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

3.4.3 Capturing the *Automobile Specification* in a Requirement Diagram

The **requirement diagram** for the *Automobile System Requirements* is shown in Figure 3.2. The kind of diagram (e.g., req) and the diagram name are shown in the diagram header at the upper left. The diagram depicts the requirements that are typically captured in a text specification. The requirements are shown in a containment hierarchy to depict the hierarchical relationship among them. The *Automobile Specification* is the top-level requirement that contains the lower-level requirements. The line with the crosshairs symbol at the top is the **containment** relationship.

The specification contains requirements for *Passenger and Baggage Load*, *Vehicle Performance*, *Riding Comfort*, *Emissions*, *Fuel Efficiency*, *Production Cost*, *Vehicle 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, its text, and can include other user-defined properties, such as verification status and risk, that are typically associated with requirements. The text for the *Maximum Acceleration* requirement is "the vehicle shall accelerate from 0 to 60 mph in less than 8 seconds" and the text for the *Fuel Efficiency* requirement is "the vehicle shall achieve at least 25 miles per gallon under the stated driving conditions."

The requirements may have been created in the modeling tool, or alternatively, they may have been imported from a requirements management tool or a text document. The requirements can be related to other requirements, design elements, and test cases using **derive, satisfy, verify, refine, trace,** and **copy** relationships. These relationships can be used to establish requirements traceability with a high degree of granularity. Some of these relationships are highlighted in Section 3.4.19. Chapter 12 provides a detailed description of how requirements are modeled in SysML. Requirements can be represented using multiple display options to view the requirements, their properties, and their relationships, which includes a tabular representation.

3.4.4 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 it. The **block definition diagram** for the *Automobile Domain* in Figure 3.3 defines the *Vehicle* and the external systems, users, and other entities that the vehicle may directly or indirectly interact with.

A **block** is a very general modeling concept in SysML that is used to model a wide variety of entities that have structure such as systems, hardware, software, physical objects, and abstract entities. 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.

The *Automobile Domain* is the top-level block in the block definition diagram in Figure 3.3. It is **composed** of other blocks as indicated by the black diamond

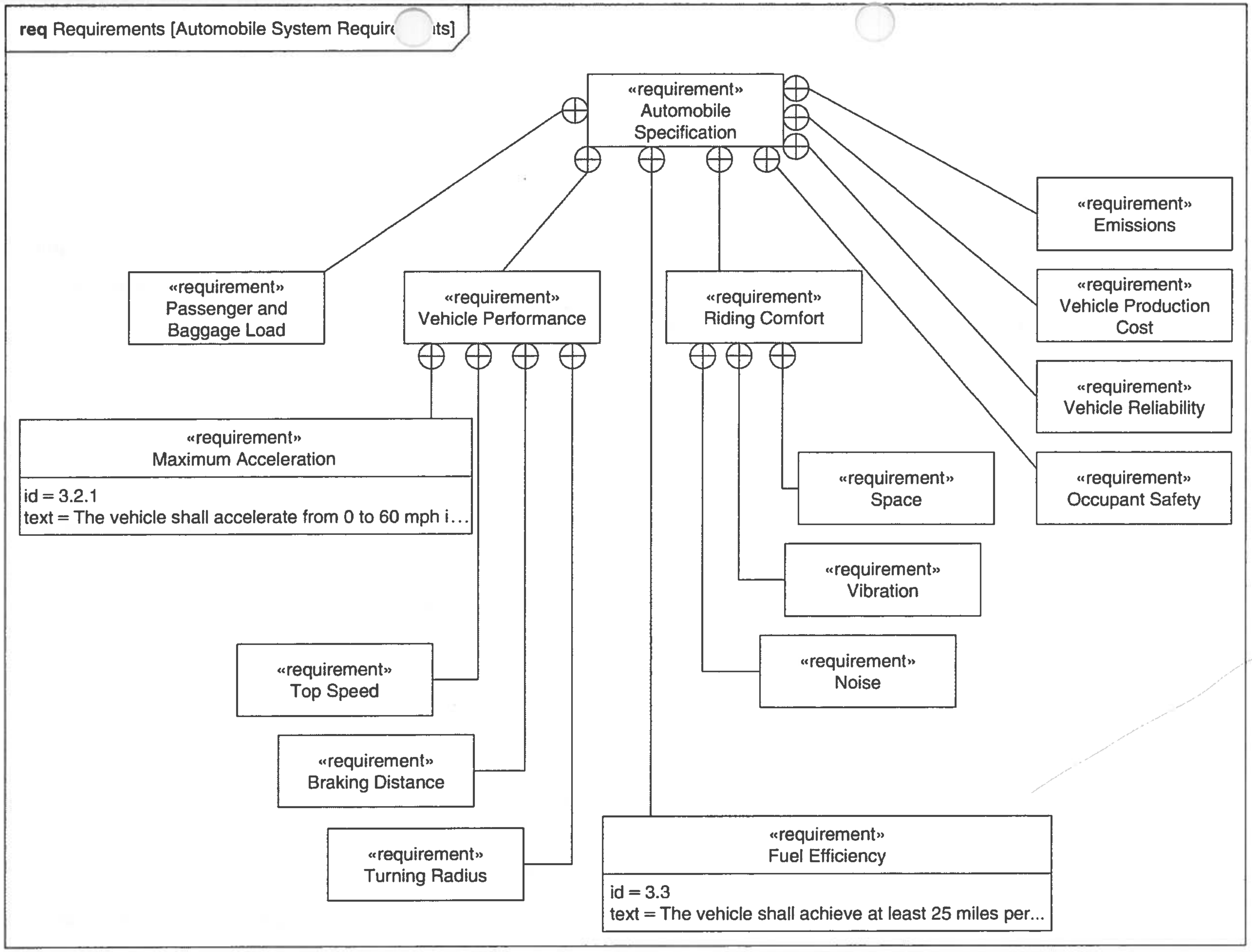
req Requirements [Automobile System Requir its]
«requirement» Automobile Specification
«requirement» Passenger and Baggage Load
«requirement» Vehicle Performance
«requirement» Riding Comfort
«requirement» Emissions
«requirement» Vehicle Production Cost
«requirement» Vehicle Reliability
«requirement» Occupant Safety
«requirement» Maximum Acceleration
id = 3.2.1
text = The vehicle shall accelerate from 0 to 60 mph i…
«requirement» Space
«requirement» Vibration
«requirement» Noise
«requirement» Top Speed
«requirement» Braking Distance
«requirement» Turning Radius
«requirement» Fuel Efficiency
id = 3.3
text = The vehicle shall achieve at least 25 miles per...

FIGURE 3.2

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

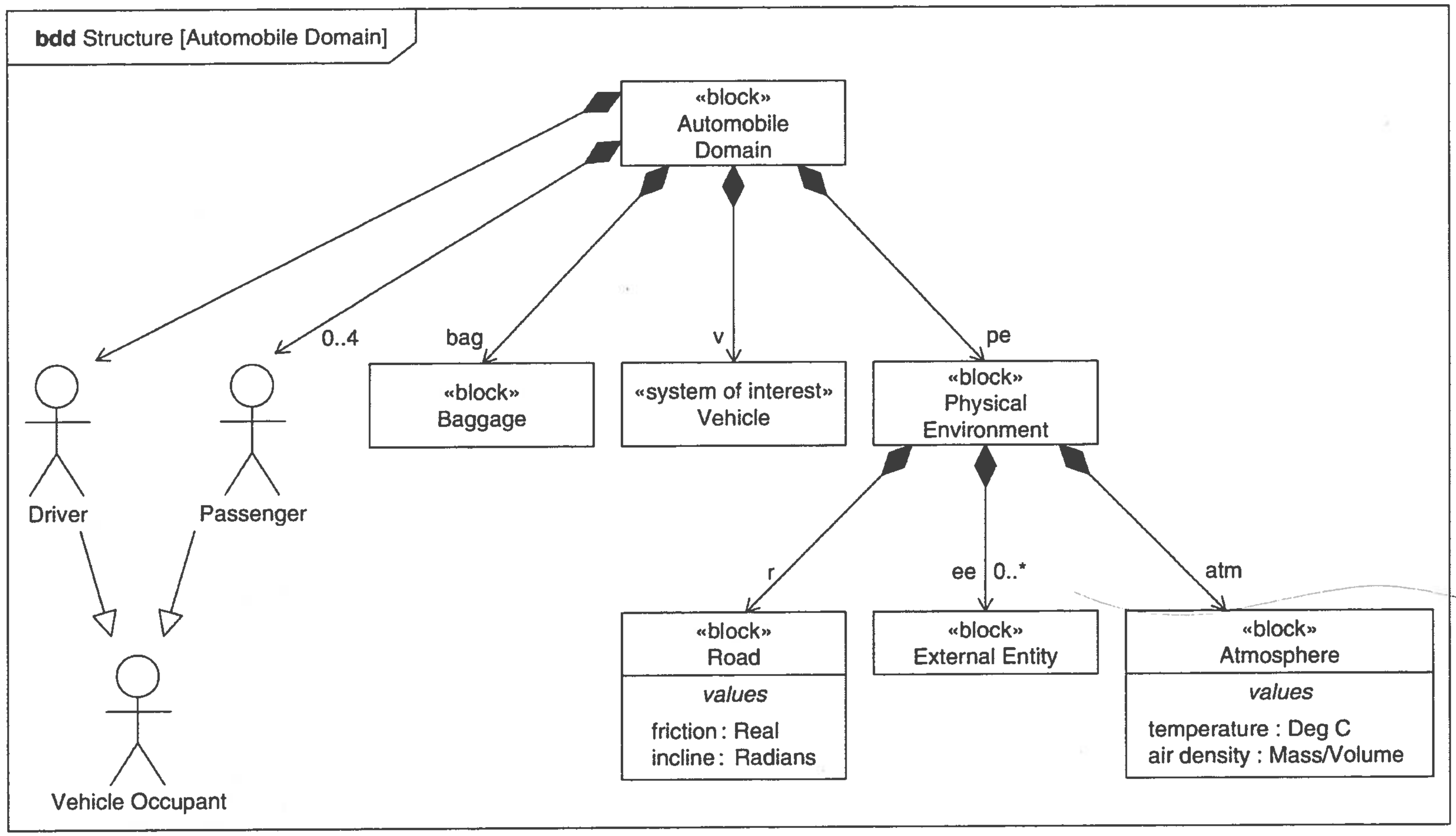

FIGURE 3.3

Block definition diagram of the *Automobile Domain* showing the *Vehicle* and its external users and physical environment.

symbol and line with the arrowhead pointing to the blocks that compose it. The differences between the composition hierarchy (i.e., black diamond) and the containment hierarchy (i.e., crosshairs symbol) shown in Figure 3.2 are explained in Part II. The name next to the arrow identifies a particular usage of a block as described later in this section. The *Vehicle* block is referred to as the «system of interest» using the bracket symbol called **guillemet**. The other blocks are external to the vehicle. These include the *Driver*, *Passenger*, *Baggage*, and *Physical Environment*. The *Driver* and *Passenger* are shown using stick-figure symbols. Notice that even though the *Driver*, *Passenger*, and *Baggage* are assumed to be physically inside the *Vehicle*, they are not part of the *Vehicle* structure, and therefore are external to it.

The *Driver* and *Passenger* are **subclasses** of *Vehicle Occupant* as indicated by the hollow triangle symbol. This means that they are kinds of vehicle occupants that 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, that the *Driver* interacts with. The interaction between the *Driver* and an *External Entity* can impact how the *Driver* interacts with the *Vehicle*, such as when the traffic light changes color. The **multiplicity** symbol *0..** represents an undetermined maximum number of external entities. The multiplicity symbol can also represent a single number 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, as described earlier. A block can have a set of **features** that further define it. The features of the block define its **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 each block at the level of detail that is appropriate for the application.

The *Road* is a block that has a property called *incline* with units of *Radians* and a property called *friction* that is defined as a real number. Similarly, *Atmosphere* is a block that has two properties for *temperature* and *air density*. These properties are used along with other properties to support analysis of vehicle acceleration and fuel efficiency, which are discussed in Sections 3.4.12 and 3.4.13.

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 block (e.g., *Automobile Domain*) down to vehicle components. Chapter 6 provides a detailed description of how blocks are modeled in SysML, including their features and relationships.

3.4.5 Use Case Diagram for *Operate Vehicle*

The **use case diagram** for *Operate Vehicle* in Figure 3.4 depicts the major functionality for operating the vehicle. The **use cases** include *Enter Vehicle*, *Exit Vehicle*, *Control Vehicle Accessory*, and *Drive Vehicle*. The *Vehicle* is the **subject**

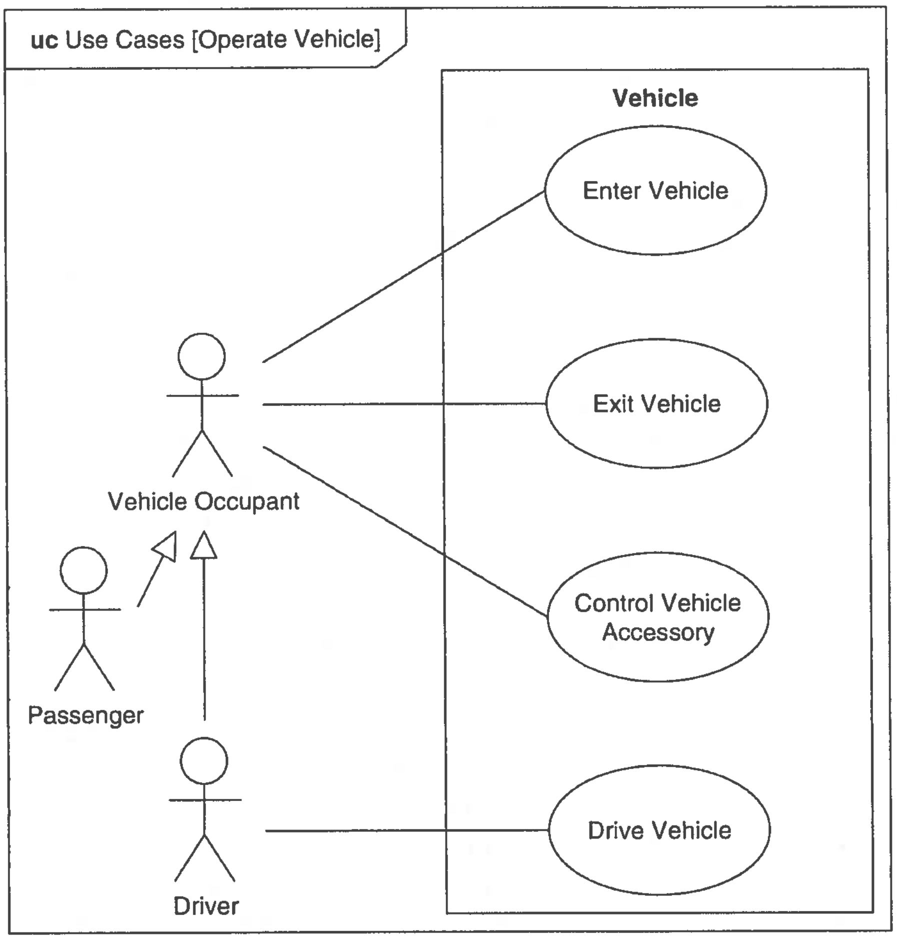

FIGURE 3.4

Use case diagram describes the major functionality in terms of how the *Vehicle* is used by the actors to *Operate Vehicle*. The actors are defined on the block definition diagram in Figure 3.3.

of the use cases and is represented by the rectangle. The *Vehicle Occupant* is an **actor** who is external to the vehicle and is represented as the stick figure. The subject and actors correspond to the blocks in Figure 3.3. In a use case diagram, the subject (e.g., *Vehicle*) is used by the actors (e.g., *Vehicle Occupant*) to achieve the goals defined by the use cases (e.g., *Drive Vehicle*).

The *Passenger* and *Driver* are both kinds of vehicle occupants as described in the previous section. All vehicle occupants participate in entering and exiting the vehicle and controlling vehicle accessories, but only the driver participates in *Drive Vehicle*. There are several other relationships between use cases that are not shown here. Chapter 11 provides a detailed description of how use cases are modeled in SysML.

Use cases define how the system is used to achieve a user goal. Other use cases can represent how the system is used across its life cycle, such as when manufacturing, operating, and maintaining the vehicle. The primary emphasis for this example is on the *Drive Vehicle* use case to address the acceleration and fuel efficiency requirements.

The requirements are often related to use cases since use cases represent the high-level functionality or goals for the system. Sometimes, use case textual descriptions are defined to accompany the use case definition. One approach to relate requirements to use cases is to capture the use case descriptions as SysML requirements and relate them to the use case using a refine relationship.

The use cases describe the high-level goals of the system as described previously. The goals are accomplished by the interactions between the actors (e.g., *Driver*) and the subject (e.g., *Vehicle*). These interactions are realized through more detailed descriptions of behavior as described in the next section.

3.4.6 Representing *Drive Vehicle* Behavior with a Sequence Diagram

The behavior for the *Drive Vehicle* use case in Figure 3.4 is represented by the **sequence diagram** in Figure 3.5. 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 *Start Vehicle*. This is followed by the *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 3.4.9 specifies the *vehicle state*. The *Turn-off Vehicle* interaction occurs following these interactions.

The **interaction occurrences** in the figure each reference a more detailed interaction as indicated by **ref**. The referenced interaction for *Start Vehicle* is another sequence diagram that is illustrated in Section 3.4.7. The remaining interaction occurrences are references allocated to activity diagrams as described in Section 3.4.8.

3.4.7 Referenced Sequence Diagram to *Start Vehicle*

The *Start Vehicle* sequence diagram in Figure 3.6 is an interaction that is referenced in the sequence diagram in Figure 3.5. As stated previously, time proceeds vertically down the diagram. In this example, the more detailed interaction shows the driver sending a **message** requesting the vehicle to start. The vehicle responds with the *vehicle on* **reply message** shown as a dashed line. Once the reply has been received, the driver and vehicle can proceed to the next interaction.

The sequence diagram can include multiple types of messages. In this example, the message is **synchronous** as indicated by the filled arrowhead on the message. The messages can also be **asynchronous** represented by an open arrowhead, where the sender does not wait for a reply. The synchronous messages represent

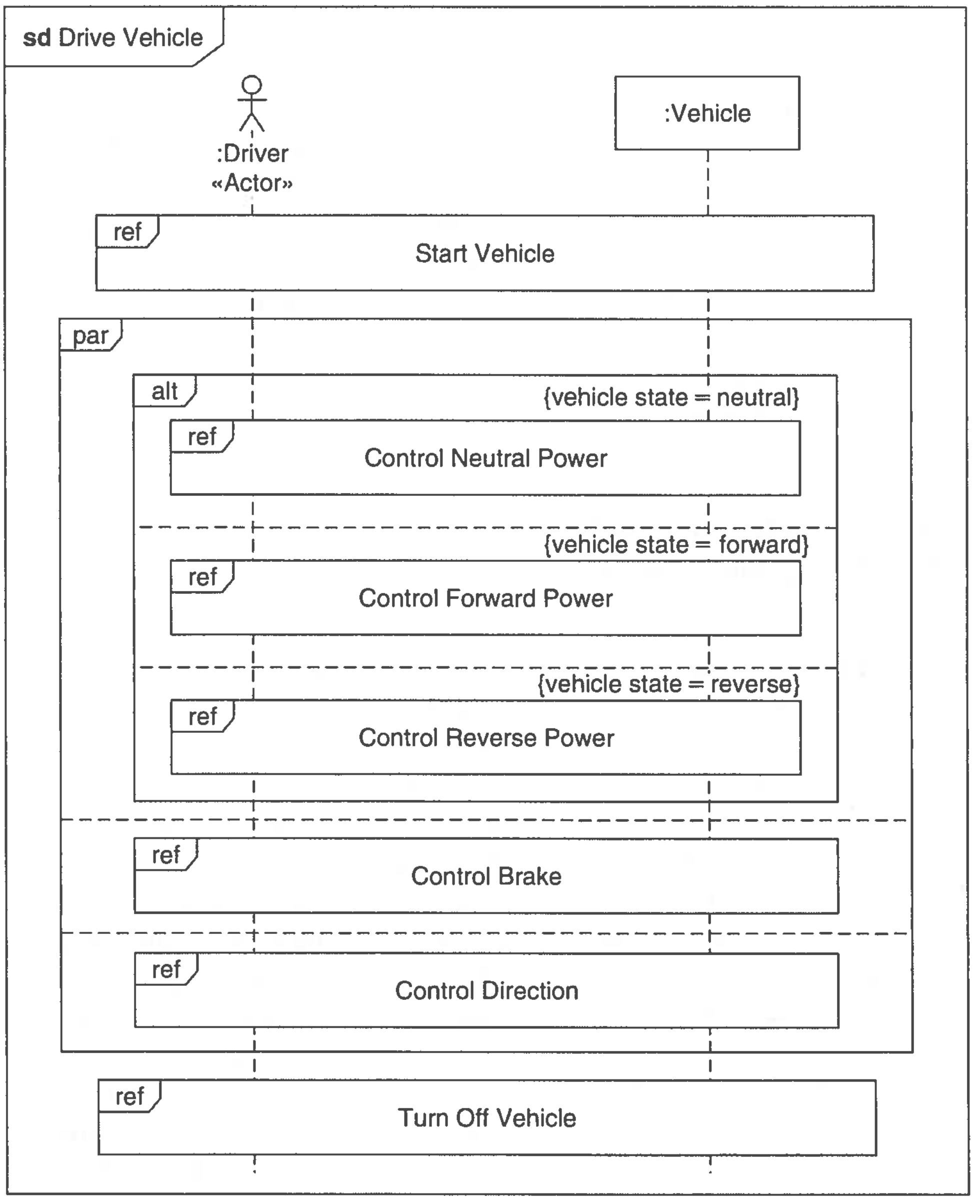

FIGURE 3.5

Drive Vehicle sequence diagram describes the interactions between the *Driver* and the *Vehicle* to realize the *Drive Vehicle* use case in Figure 3.4.

an operation call that specifies a request for service. The arguments of the operation call represent the input data and return.

Sequence diagrams can include multiple message exchanges between multiple lifelines that represent interacting entities. The sequence diagram also provides considerable additional capability to express behavior that includes other message types, timing constraints, additional control logic, and the ability to decompose the behavior of a lifeline into the interaction of its parts. Chapter 9 provides a detailed description of how interactions are modeled with sequence diagrams.

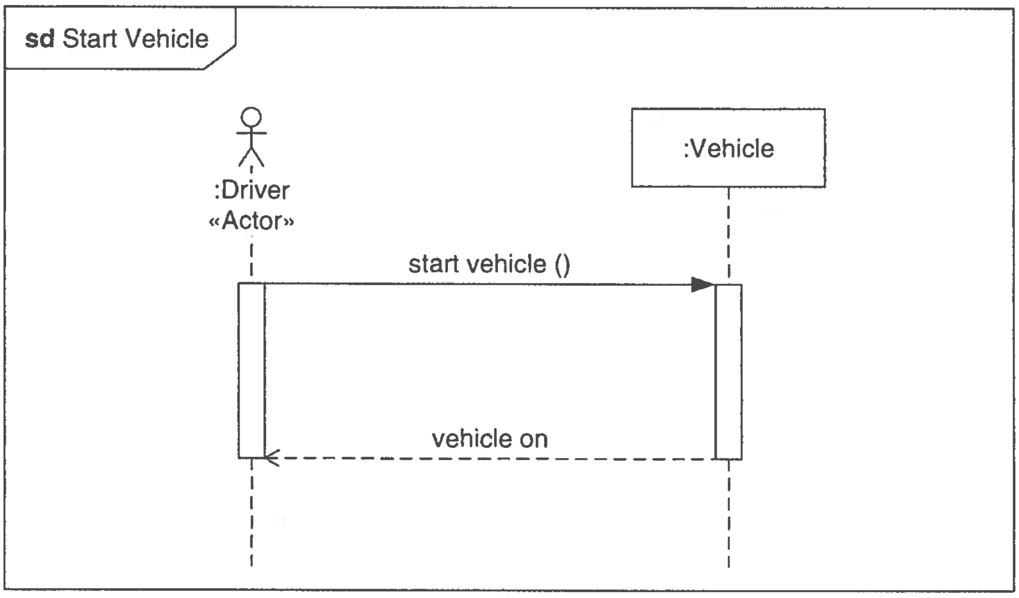

FIGURE 3.6

Sequence diagram for the *Start Vehicle* interaction that was referenced in the *Drive Vehicle* sequence diagram, showing the message from *Driver* requesting *Vehicle* to start and the *vehicle on* reply from the *Vehicle.*

3.4.8 *Control Power* Activity Diagram

The sequence diagram is effective for communicating discrete types of behavior as indicated with the *Start Vehicle* sequence diagram in Figure 3.6. However, continuous types of behaviors associated with the interactions to *Control Power*, *Control Brake*, and *Control Direction* can sometimes be more effectively represented with activity diagrams.

The *Drive Vehicle* sequence diagram in Figure 3.5 includes the control power interactions that we would like to represent with an activity diagram instead of a sequence diagram. To accomplish this, the *Control Neutral Power*, *Control Forward Power*, and *Control Reverse Power* interactions in Figure 3.5 are **allocated** to a corresponding *Control Power* activity diagram using the SysML allocation relationship (not shown).

The **activity diagram** in Figure 3.7 shows the **actions** required of the *Driver* and the *Vehicle* to *Control Power*. The **activity partitions** (or **swimlanes**) represent 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** and transitions to the *Control Accelerator Position* action that is performed by the *Driver*. The output of this action is the *Accelerator Cmd*, which is a continuous input to the *Provide Power* action that the *Vehicle* must perform. The output of the *Provide Power* action is the *Torque* generated by the wheels to the road to produce the force that accelerates the *Vehicle*. When the *ignition off* signal is received by the *Vehicle,* the activity terminates at the **Activity Final**. Based

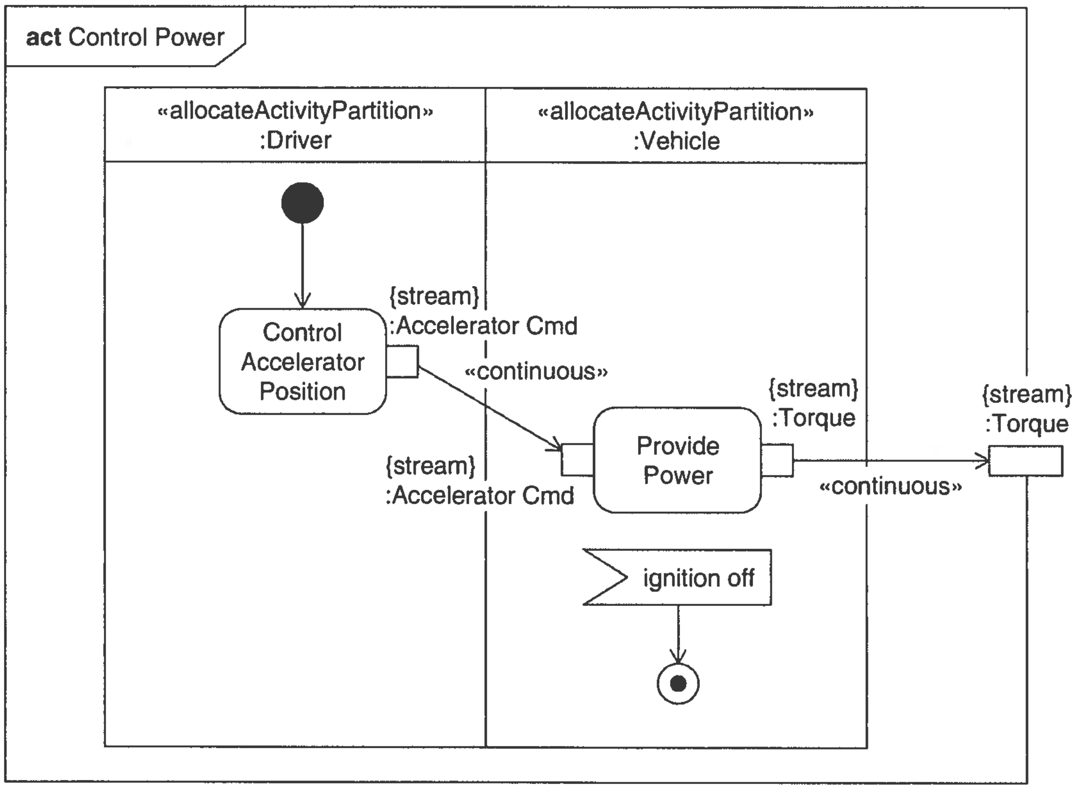
act Control Power
«allocateActivityPartition» :Driver
«allocateActivityPartition» :Vehicle
Control Accelerator Position
{stream} :Accelerator Cmd
«continuous»
{stream} :Accelerator Cmd
Provide Power
{stream} :Torque
{stream} :Torque
«continuous»
ignition off

FIGURE 3.7

Activity diagram allocated from the *Control Power* interaction that was referenced in the *Drive Vehicle* sequence diagram in Figure 3.5. It shows the continuous *Accelerator Cmd* input from the *Driver* to the *provide power* action that the *Vehicle* must perform.

on this scenario, the *Driver* is required to *Control Accelerator Position* and the *Vehicle* is required to *Provide Power*.

Activity diagrams include semantics for precisely specifying the behavior in terms of the flow of control and inputs and outputs. Chapter 8 provides a detailed description of how activities are modeled.

3.4.9 State Machine Diagram for *Drive Vehicle States*

The **state machine diagram** for the *Drive Vehicle States* is shown in Figure 3.8. 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 starts in the *vehicle off* state. The *ignition on* event 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. After entry to the *vehicle on* state, the *Vehicle* 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. While in the *forward* state, the *Vehicle* performs the *Provide Power* behavior that was referred to in the activity diagram in Figure 3.7. The *neutral select* event triggers the transition from

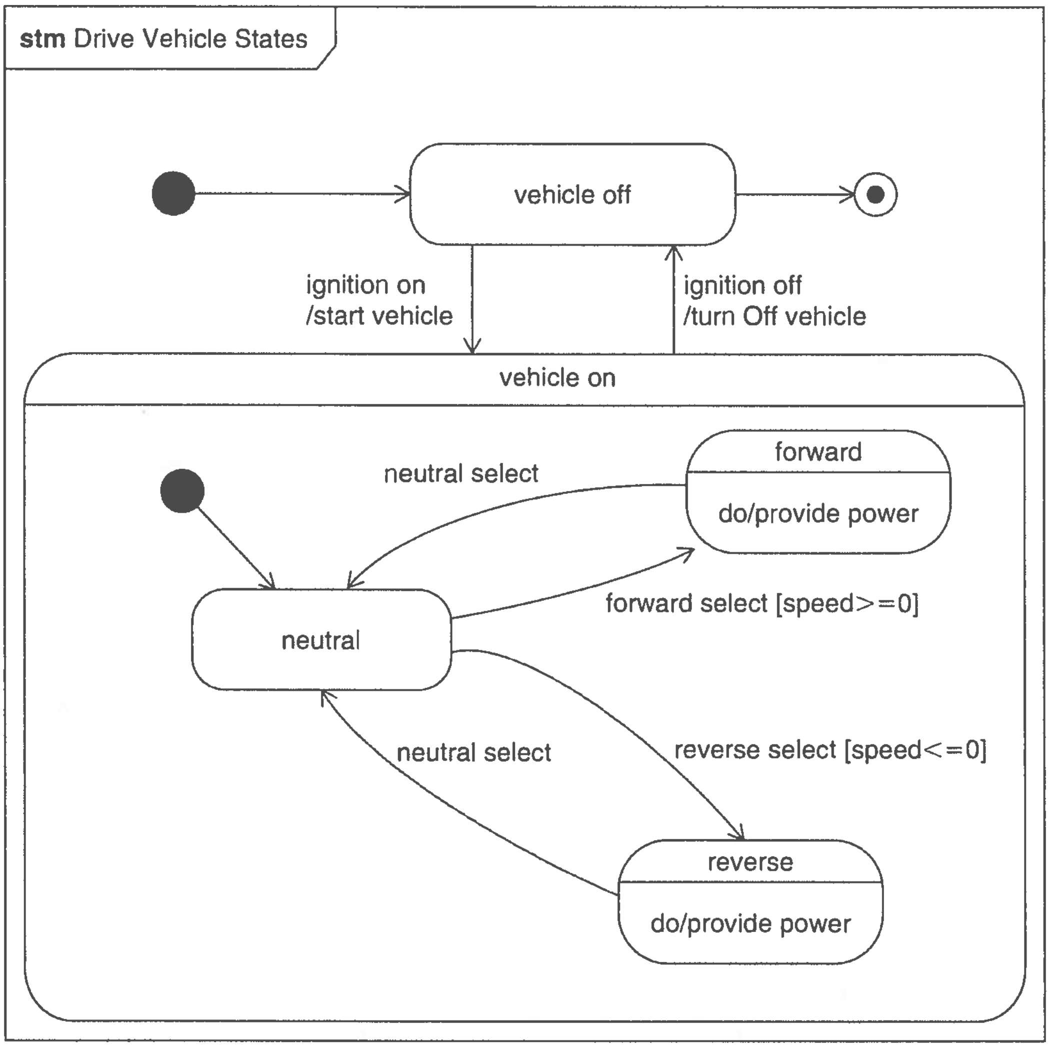

FIGURE 3.8

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

the *forward* state back 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. The *Vehicle* can reenter the *vehicle on* state when an *ignition on* event occurs.

A state machine can specify the life-cycle behavior of a block in terms of its states and transitions, and are often used with sequence and activity diagrams, as shown in this example. State machines have many other features including **orthogonal regions** and additional transition semantics that are described in Chapter 10.

3.4.10 *Vehicle Context* Using an Internal Block Diagram

The *Vehicle Context Diagram* is shown in Figure 3.9. 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 3.3. The *Vehicle* has interfaces with the *Driver*, the *Atmosphere*,

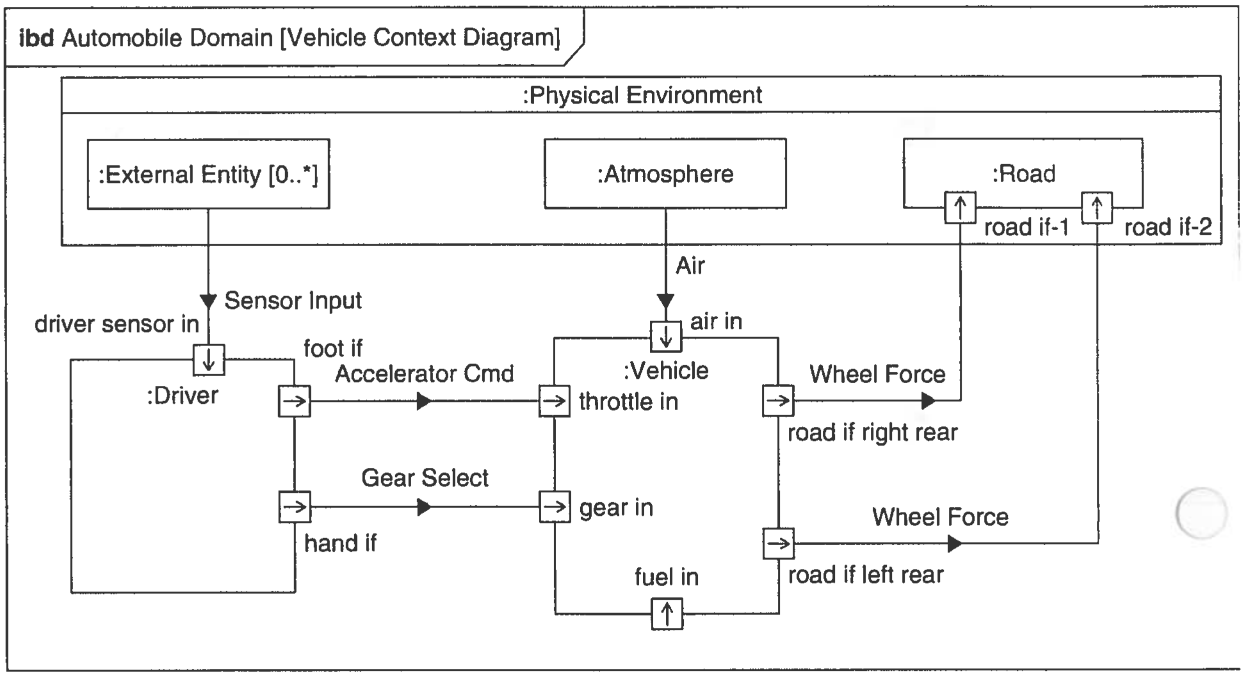

FIGURE 3.9

Internal block diagram for the *Vehicle Context* shows the *Vehicle* and its external interfaces with the *Driver* and *Physical Environment* that were defined in Figure 3.3.

and the ***Road***. The ***Driver*** has interfaces with the ***External Entities*** such as a traffic light or another vehicle, via the driver sensor inputs (e.g., seeing, hearing). 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 3.3.

This context diagram is an **internal block diagram** that shows how the **parts** of the *Automobile Domain* block from Figure 3.3 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** specify interaction points with other parts and are represented as the small squares on the boundary of the parts. **Connectors** define how the parts connect to one another via their ports and are represented as the lines between the ports. Parts can also be connected without ports as indicated by some of the interfaces in the figure when the details of the interface are not of interest to the modeler.

In Figure 3.9, only the external interfaces needed for the ***Vehicle*** to provide power are shown. For example, the interfaces between the rear tires and the road are shown. It is assumed to be a rear wheel-drive vehicle where power can be distributed differently to the rear wheels depending on tire-to-road traction and other factors. The interface between the front tires and the road is not shown in this diagram, but it would be shown when representing the external interfaces for the steering subsystem where the front tires would play a significant role. It is common modeling practice to only represent the aspects of interest on a particular diagram, even though additional information is included in the model.

The black-filled arrowheads on the connector are called **item flows** that represent the items flowing between parts and may include mass, energy, and/or information. In this example, the *Accelerator Cmd* that was previously defined in the activity diagram in Figure 3.8 flows from the *Driver* port to the *throttle in* port of the *Vehicle*, and the *Gear Select* flows from another *Driver* port to the *gear in* port on the *Vehicle*. The inputs and outputs from the activity diagram are **allocated** to the item flows on the connectors. Allocations are discussed as a general-purpose relationship for mapping one model element to another in Chapter 13.

In SysML, there are two different kinds of ports. The **flow port** specifies the kind of item that can flow in or out of an interaction point, and a **standard port** specifies the services that either are required or provided by the part. The port provides the mechanism to integrate the behavior of the system with its structure.

The internal block diagram enables the modeler to specify both external and internal interfaces of a system or component. An internal block diagram shows how parts are connected, as distinct from a block definition diagram that does not show connectors. Details of how to model internal block diagrams are described in Chapter 6.

3.4.11 *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 Hierarchy* in Figure 3.10 is a block definition diagram that shows the decomposition of the *Vehicle* into its components. The *Vehicle* is composed of the *Body*, *Chassis*, *Interior*, *Power Train*, and other types of components. Each component type 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 looses traction. The front wheels are not shown, but could be part of the chassis or part of the steering assembly and would have different roles as well.

The engine may be either 4 or 6 cylinders as indicated by the specialization relationship. The 4- and 6-cylinder vehicle configuration alternatives are being considered to satisfy the acceleration and fuel efficiency requirements. Only one engine type is part of a particular *Vehicle* as indicated by the *{complete, disjoint}* constraint. This implies that the 4- and 6-cylinder engines represent a complete set of subclasses and are mutually exclusive or disjoint.

The *vehicle:Controller* «software» has been allocated to the *Vehicle Processor* as indicated by the allocation compartment. The *Vehicle Processor* is the execution platform for the vehicle control software. The software is being enhanced to control many of the automobile engine and transmission functions to optimize engine performance and fuel efficiency.

The interaction and interconnection between these components is analyzed in a similar way to what was done at the *Vehicle* black box level, and is used to specify the components of the *Vehicle* system as described in the next sections.

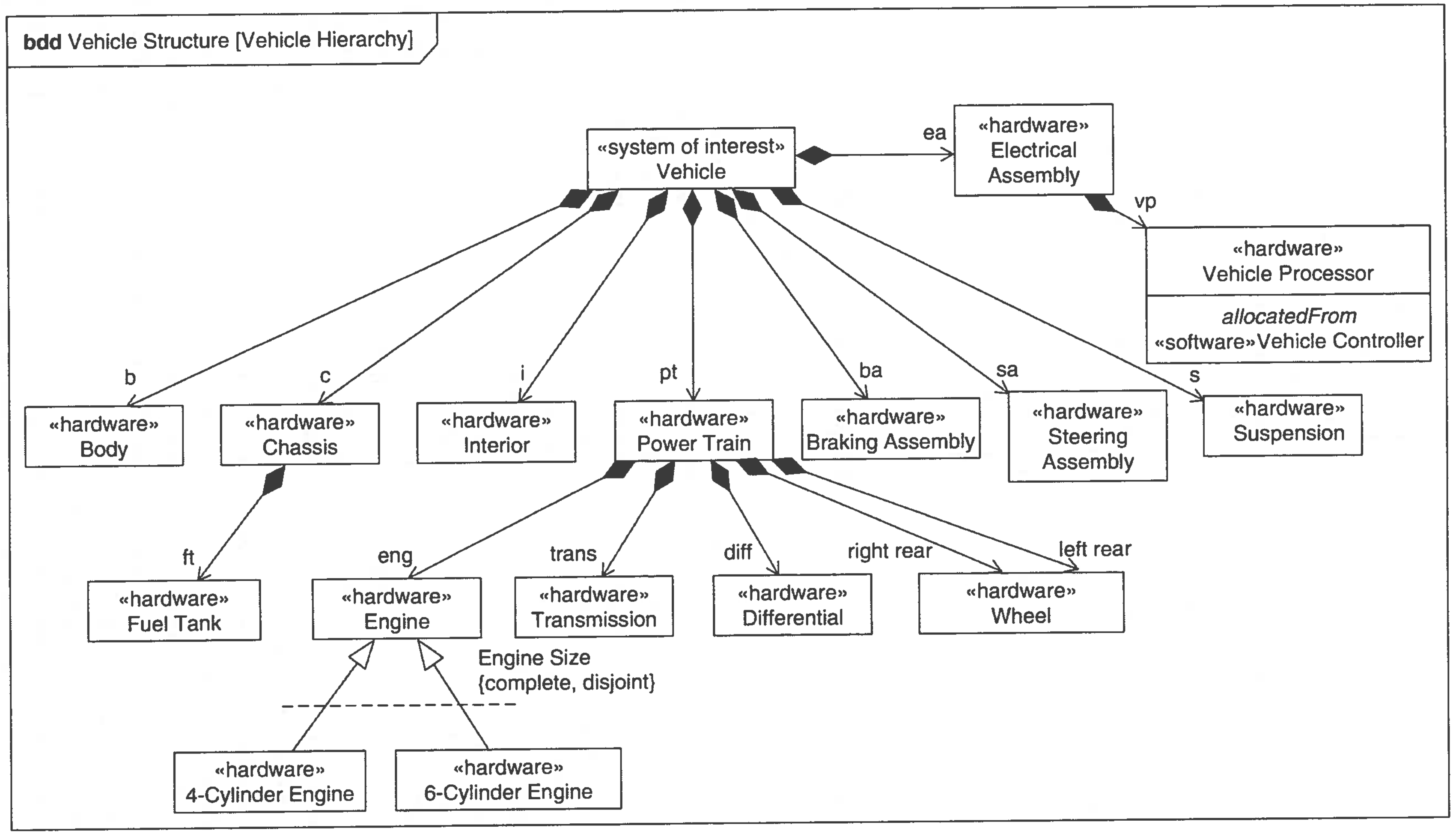

FIGURE 3.10

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

3.4.12 Activity Diagram for *Provide Power*

The activity diagram in Figure 3.7 showed that the vehicle must *provide power* in response to the driver accelerator command and generate wheel–tire torque at the road surface. The *Provide Power* activity diagram in Figure 3.11 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 are the torque from the right and left rear wheels to the road that provides the force to accelerate the *Vehicle*. Some of the other inputs and outputs, such as exhaust from the engine, are not included for simplicity. The activity partitions represent the vehicle components shown in the block definition diagram in Figure 3.10.

The fuel tank stores and dispenses the fuel to the *Engine*. The accelerator command and air and fuel are input 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* that distributes torque to the right and left rear wheels to *provide traction* to the road surface to generate the force to accelerate the *Vehicle*.

The actions that are allocated to the *Vehicle* components realize the *provide power* action that the *Vehicle* performs, as shown in Figure 3.7. This approach is used to decompose the system behavior.

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 outputs, which means that 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 associated with functional flow diagrams. The continuous flows are assumed to be streaming.

Many other activity diagram features are explained in Chapter 8; they provide a capability to precisely specify behavior in terms of the flow of control and data, and the ability to reuse and decompose behavior.

3.4.13 Internal Block Diagram for the *Power Subsystem*

The previous activity diagram described 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 3.12 shows how the parts are interconnected to achieve this functionality and is used to specify the interfaces between the parts. This is a structural view of the system versus the behavioral view that was expressed in the activity diagram.

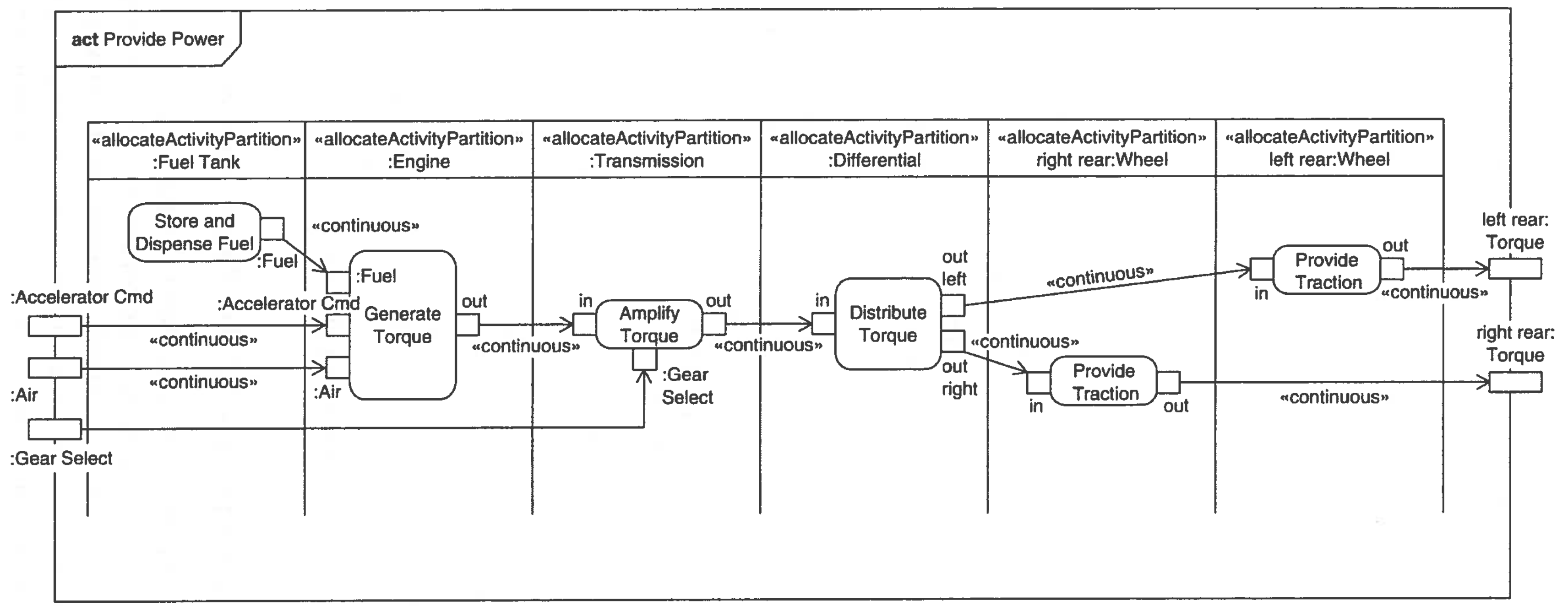

FIGURE 3.11

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 3.7 with activity partitions that correspond to the components in Figure 3.10.

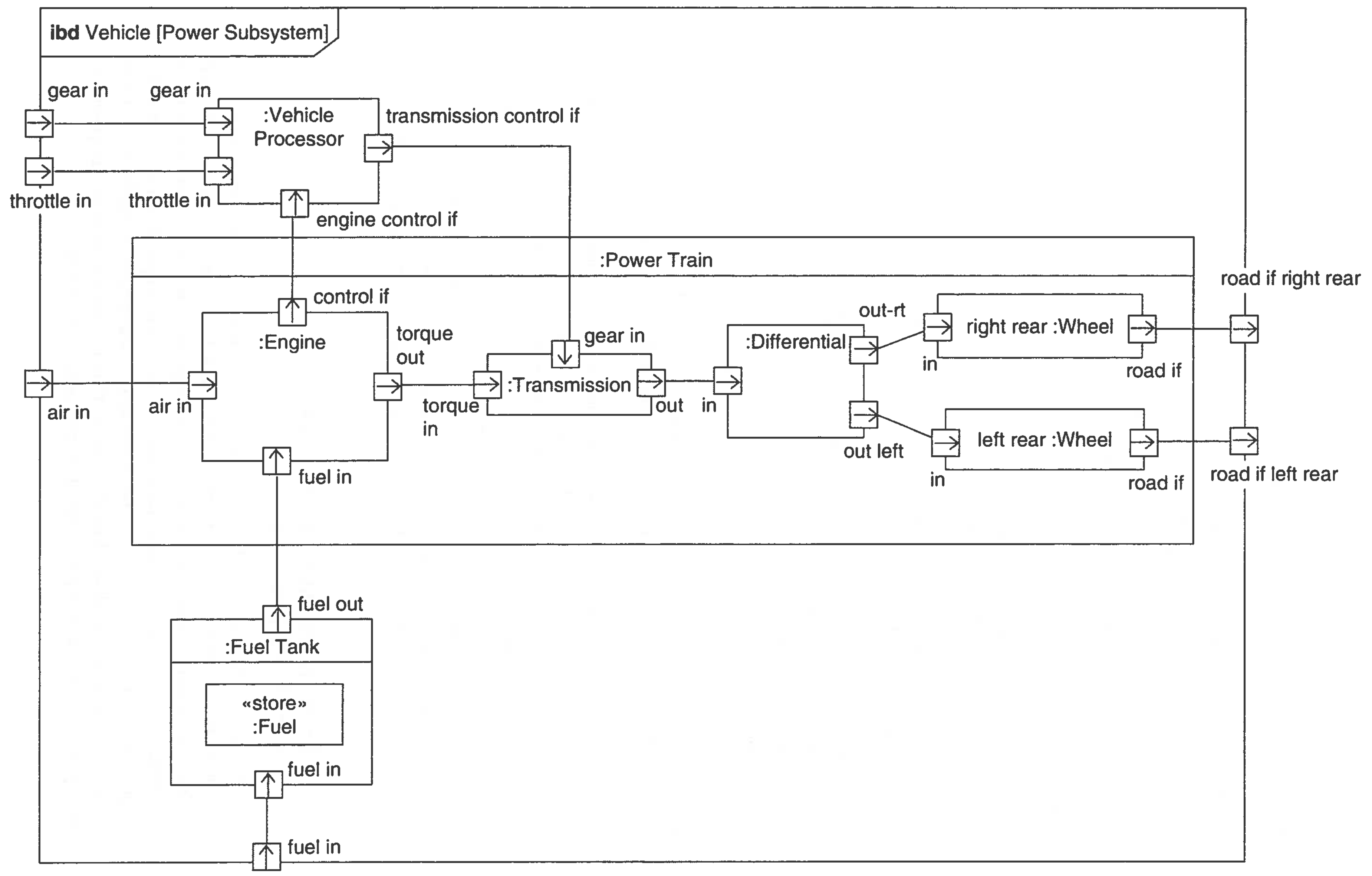

FIGURE 3.12

Internal block diagram for the *Power Subsystem* shows how the parts that *provide power* are interconnected. The parts are represented as the activity partitions in Figure 3.11.

The internal block diagram represents the *Power Subsystem* that only includes the parts of the *Vehicle* that collaborate to *provide power*. The frame of the diagram represents the *Vehicle* black box. The ports on the diagram frame correspond to the same ports shown on the *Vehicle* in the *Vehicle Context* diagram in Figure 3.9. This enables the external interfaces to be preserved as the internal structure of the *Vehicle* is further specified.

The *Engine*, *Transmission*, *Differential*, *right rear* 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». The item flows on the connectors are not shown, but represent the items that flow through the system and are allocated from the inputs and outputs on the *Provide Power* activity diagram in Figure 3.11.

Additional subsystems can be created in a similar way to realize specific functionality such as provide braking and provide steering. A composite view of all of the interconnected parts across all subsystems can also be created in a composite *Vehicle* internal block diagram. An example of this is included in the residential security example in Chapter 16.

It is appropriate to elaborate on the usage concept that was first introduced in Section 3.4.11 when discussing the *right rear* and *left rear* wheels. A part in an internal block diagram represents a particular usage of a block. The block represents the generic definition, whereas the part represents a usage of a block definition in a particular context. Thus, the right rear and left rear are different usages of the *Wheel* block in the context of the *Vehicle*. A usage (or part) is the same as a role. The parts in the diagram are indicated by the colon (:) notation. A part enables the same block to be reused in many different contexts and be uniquely identified by its usage. Each part may have unique behaviors, properties, and constraints that apply to its particular usage.

The concept of definition and usage is applied to many other SysML language constructs as well. One example is that the item flows themselves can have a definition and usage. For example, an item flow entering the fuel tank 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.

As mentioned previously, Chapter 6 provides the detailed language description for both block definition diagrams and internal block diagrams, and includes the concept of role, references, and many other key concepts for modeling blocks and parts.

3.4.14 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, such that increasing the acceleration capability can result in a design with lower fuel efficiency. Two alternative configurations, including a 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- and 6-cylinder engine alternatives are shown in the *Vehicle Hierarchy* in Figure 3.10. There are other design impacts that may result from the automobile configurations with different engines, such as the vehicle weight, body shape, and electrical power. This simplified example only considers the impact on the *Power Subsystem*. The vehicle controller is assumed to control the fuel and air mixture, and control when the gear changes the automatic transmission to optimize engine and overall performance.

The block definition diagram in Figure 3.13 introduces a new type of block called a **constraint block**. Instead of defining systems and components, the constraint block defines constraints in terms of equations and their **parameters**.

In this example, the *Analysis Context* block is composed of a series of constraint blocks to analyze the vehicle acceleration to determine whether either the 4- or 6-cylinder vehicle configuration can satisfy its requirement. The constraint blocks define generic equations for *Gravitational Force*, *Drag Force*, *Power Train*

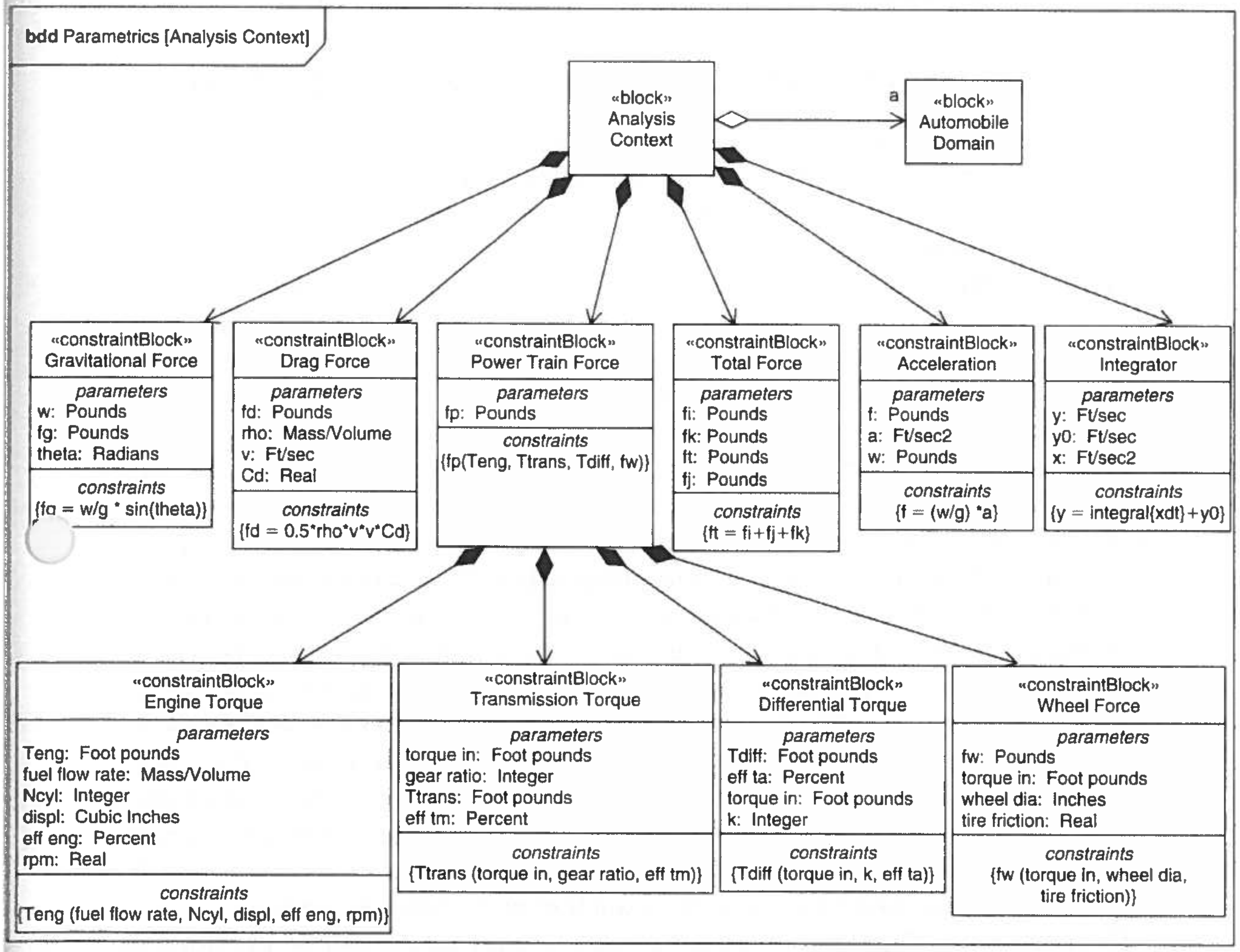

FIGURE 3.13

Block definition diagram for the *Analysis Context* that defined the equations for analyzing the vehicle acceleration requirement. The equations and their parameters are specified using constraint blocks. The *Automobile Domain* block from Figure 3.3 is referenced since it is the subject of the analysis.

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 and/or dimensions in the constraint block.

The *Power Train Force* is further decomposed into other constraint blocks that represent the equations for torque from the *Engine*, *Transmission*, *Differential*, and *Wheels*. The equations are not explicitly defined, but the critical parameters of the equations are. This is important since it may be of value to identify the critical parameters, and to defer definition of the equations until the detailed analysis is performed.

The *Analysis Context* block also references the *Automobile Domain* block that was originally shown in the block definition diagram in Figure 3.3. The intent of this diagram is to identify both the equations for the analysis and the subject of the analysis. Referencing the *Automobile Domain* enables the equations to constrain the properties of the *Vehicle*, its components, and the physical environment. The parameters of the generic equations are bound to the properties of the system and the environment that is being analyzed, as described in the next section.

3.4.15 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 3.14 shows how these equations are used to analyze the vehicle acceleration to determine the time for the *Vehicle* to accelerate from 0 to 60 mph.

The parametric diagram shows a network of constraints (equations). Each constraint is a usage of a constraint block defined in the block definition diagram in Figure 3.13. The parameters of the equation 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 that is bound to the parameter *f in* the *Acceleration* equation. This means that *ft* in the *Total Force* equation is equal to *f in* in the *Acceleration* equation.

The parameters can also be bound to **properties** of blocks to make the parameter equal to the property. The properties of blocks are shown as the rectangles in the diagram. An example is the binding of the coefficient of drag parameter *cd* in the *Drag Force* equation to the drag property called *drag*, which is a property of the vehicle *Body*. The dot notation *"a.v.b."* that precedes the drag property specifies that this is a property of the body, which is part of the vehicle that is part of the *Automobile Domain*. Another example is the binding of the road *incline* angle to the angle *theta* in the gravity force equation. This binding enables parameters of generic equations to be set equal to specific properties of the blocks. In this way, generic equations can be used to analyze many different designs.

The parametric diagram and related modeling information can be provided to the appropriate simulation and/or analysis tools to support execution. This engineering analysis is used to perform sensitivity analysis and determine which property values are required to satisfy the acceleration requirement.

Other analysis can be performed to determine the required property values for the system components (e.g., *Body*, *Chassis*, *Engine*, *Transmission*, *Differential*,

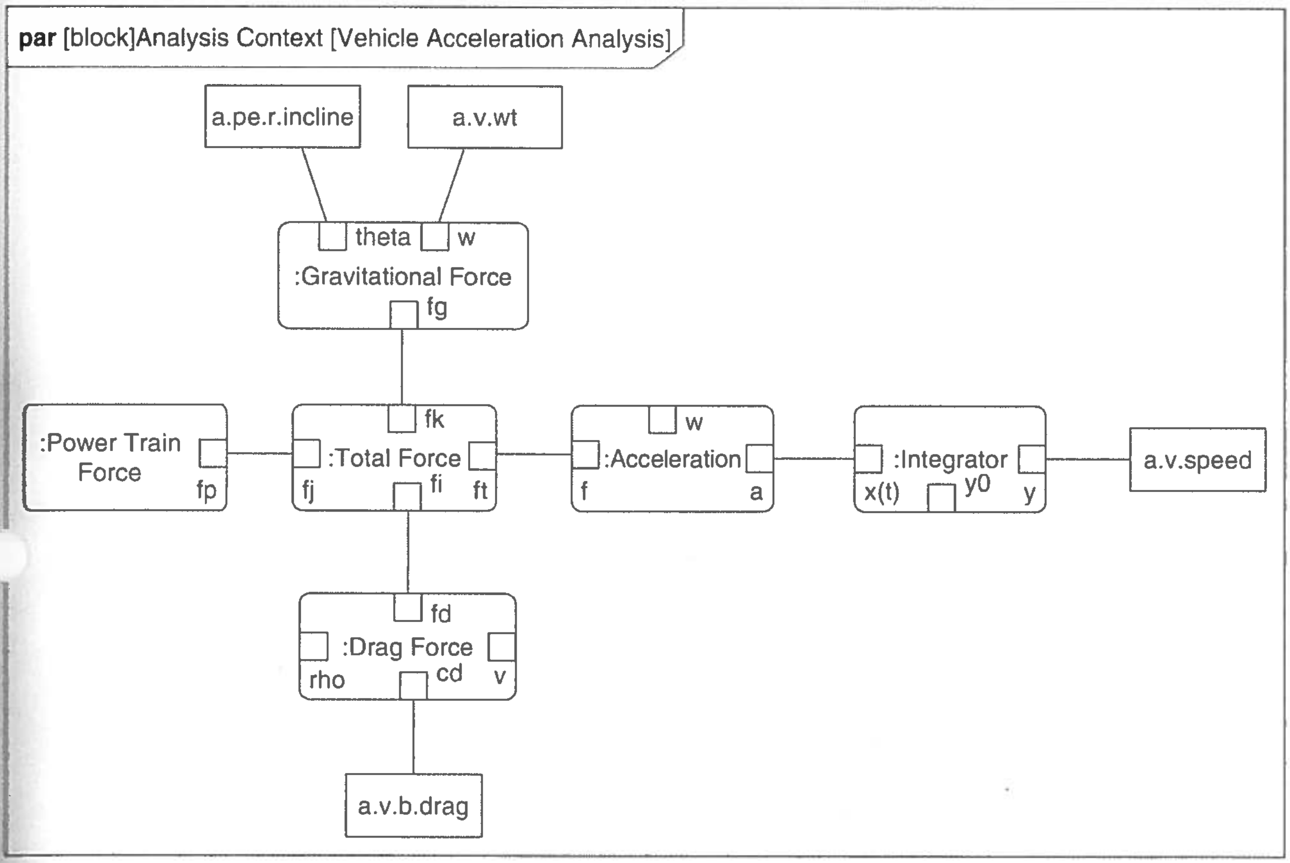

FIGURE 3.14

Parametric diagram that uses the equations defined in Figure 3.13 to analyze *Vehicle Acceleration*. The parameters of the equations are bound to parameters of other constraints and to properties of the *Vehicle* and its environment.

Brakes, Steering Assembly) to satisfy the overall system requirements. 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 so on. The parametrics enable the critical properties of the system to be identified and integrated with analysis models. Details of how to model constraint blocks and their usages in parametric diagrams are described in Chapter 7.

3.4.16 Analysis Results from Analyzing *Vehicle Acceleration*

As mentioned in the previous section, the parametric diagram is expected to be executed in an engineering analysis tool to provide the results of the analysis. This may be a separate specialized analysis tool that is not provided by the SysML modeling tool, such as a simple spreadsheet or a high-fidelity performance simulation depending on the need. The analysis results from the execution then provide values that can be incorporated back into the SysML model.

The analysis results from executing the constraints in the parametric diagram are shown in Figure 3.15. This example uses the **UML timing diagram** to display the results. Although the timing diagram is not currently one of the SysML diagram

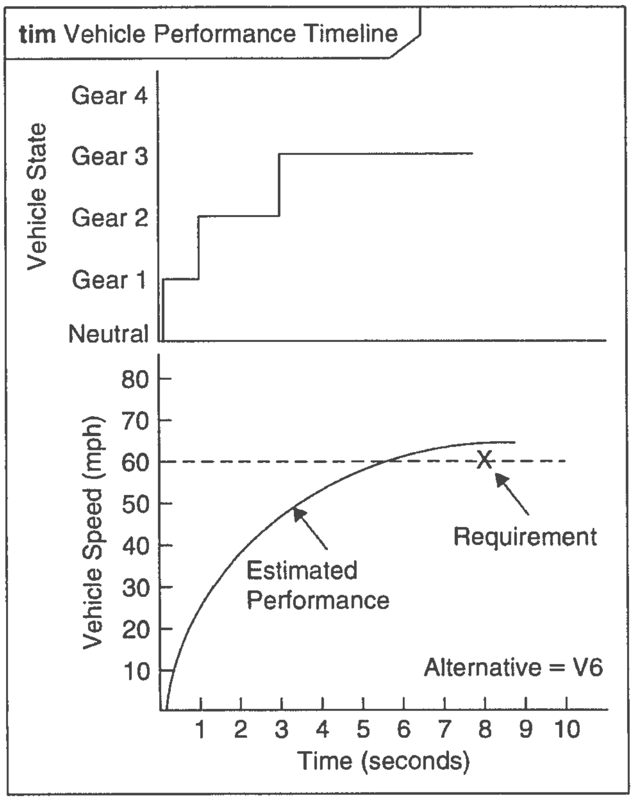

FIGURE 3.15

Analysis results from executing the constraints in the parametric diagram in Figure 3.14 showing the vehicle speed property and *Vehicle* state as a function of time. This is captured in a UML timing diagram.

types, it can be used in conjunction with SysML if it is useful for the analysis, along with other more robust visualization methods. The vehicle speed property 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 3.8. Based on the analysis performed, the 6-cylinder (V6) vehicle configuration is able to satisfy its acceleration requirement but a similar analysis showed that the 4-cylinder (V4) vehicle configuration does not satisfy the requirement.

3.4.17 Using the *Vehicle Controller* 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 for a minimum of 25 miles per gallon under the stated driving conditions as specified in the *Fuel Efficiency* requirement in Figure 3.2.

The activity diagram in Figure 3.16 is a refinement of a portion of the *Provide Power* activity diagram in Figure 3.11. In this figure, the *vehicle controller* software has been added as an activity partition to support the analysis needed to optimize

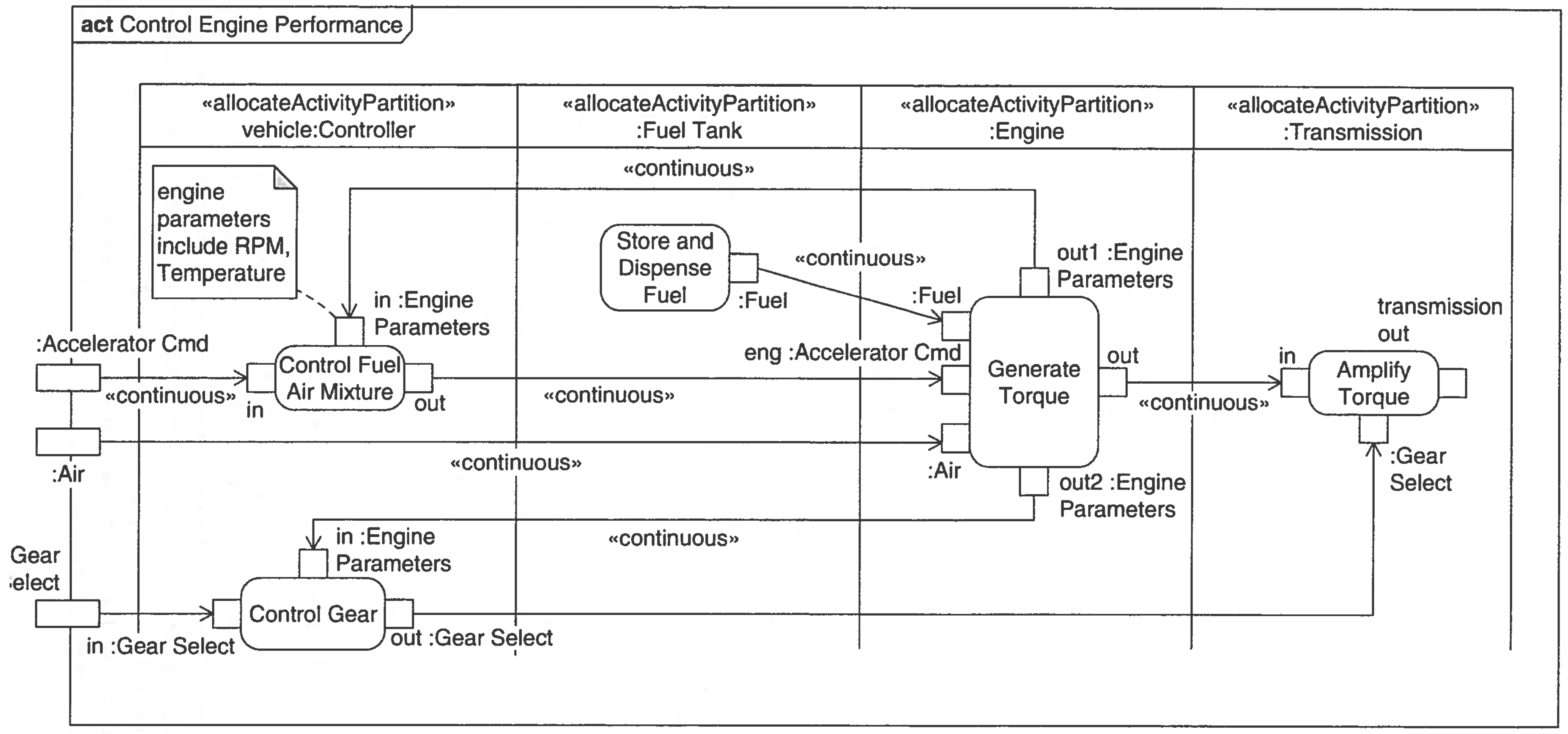

FIGURE 3.16

Activity diagram used to analyze the vehicle controller software interaction with the engine and transmission to optimize fuel efficiency and engine performance. This diagram is a refinement of a portion of the activity diagram in Figure 3.11.

fuel efficiency and engine performance. The *vehicle Controller* includes an action to *control fuel-air mixture* that in turn produces the engine accelerator command. The inputs to this action include the *Accelerator Cmd* from the *Driver* and *Engine Parameter* such as revolutions per minute (RPM) and engine temperature. 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. 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 3.8.

The specification of the algorithms to realize these actions requires further analysis. A parametric diagram can specify the required fuel and air mixture in terms of RPM and engine temperature to achieve optimum fuel efficiency, and they can be used to constrain the input and output of the actions. The algorithms must implement these constraints by controlling fuel flow rate and air intake, and perhaps other parameters. The algorithms, which consist of mathematical and logical expressions, can be captured in an activity diagram or directly in code. Based on the previous engineering analysis—the details of which are omitted here—the V6 engine is able to satisfy the fuel efficiency requirements and is selected as the preferred vehicle system configuration.

3.4.18 Specifying the *Vehicle* and Its Components

The block definition diagram in Figure 3.10 defined the blocks for the *Vehicle* and its components. The preceding analysis is used to specify the features of the blocks 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 for state-based behavior and definitions of items that are stored by the block, such as fuel.

A simple example is the specification of the *Engine* block shown in Figure 3.17. This block was originally shown in the *Vehicle Hierarchy* block definition diagram in Figure 3.10. In this example, the *Engine* hardware element performs a function called *generate torque*, with ports that specify its interfaces to *air in*, *fuel in*, *control if*, and *out torque*. Selected properties are shown that represent performance and physical properties including its *displacement*, *combustion efficiency*, *max power*, and *weight* along with their **units**. The property values may also be represented as either a single value or a distributed value. Other blocks are specified in a similar way.

3.4.19 Requirements Traceability

The *Automobile System Requirements* were shown in Figure 3.2. Capturing the text-based requirements in the SysML model provides the means to establish traceability between the text-based requirements and other parts of the model.

The requirements traceability for the *Maximum Acceleration* requirement is shown in Figure 3.18. The requirement is **satisfied** by the *Power Subsystem*. The **rationale** refers to the engineering analysis based on the *Vehicle Acceleration Analysis* parametric diagram in Figure 3.14. The *Max Acceleration* **test case** is

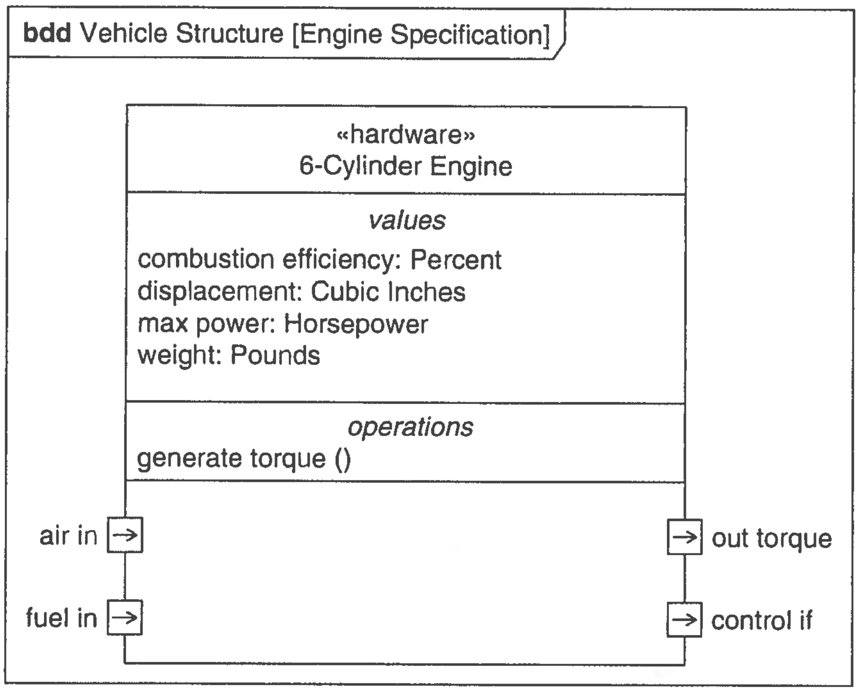

FIGURE 3.17

The block definition diagram 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 3.10.

also shown as the method to verify that the requirement is satisfied. In addition, the *Engine Power* requirement is derived from the *Max Acceleration* requirement and contained in the *Engine Specification*. The *Engine* block refines the *Engine Specification* by restating the text requirements in the model. In this way, the system requirements can be traced to the system design and test cases, along with rationale.

The direction of the arrows points from the *Power Subsystem* design, *Max Acceleration* test case, and *Engine Power* requirement to the *Max Acceleration* as the source requirement. This is in the opposite direction that is often used to represent requirements flow-down. The direction represents a dependency from the design, test case, and derived requirement to the source requirement, such that if the source requirement changes, the design, test case, and derived requirement should also change.

As stated previously, there are other requirements relationships for trace and copy. The requirements are supported by multiple notation options including a tabular representation. Details of how SysML requirements and their relationships are modeled are described in Chapter 12.

3.4.20 Package Diagram for Organizing the Model

The concept of an integrated system model is a foundational concept for MBSE as described in Chapter 2. The **model** contains all of the **model elements**. The

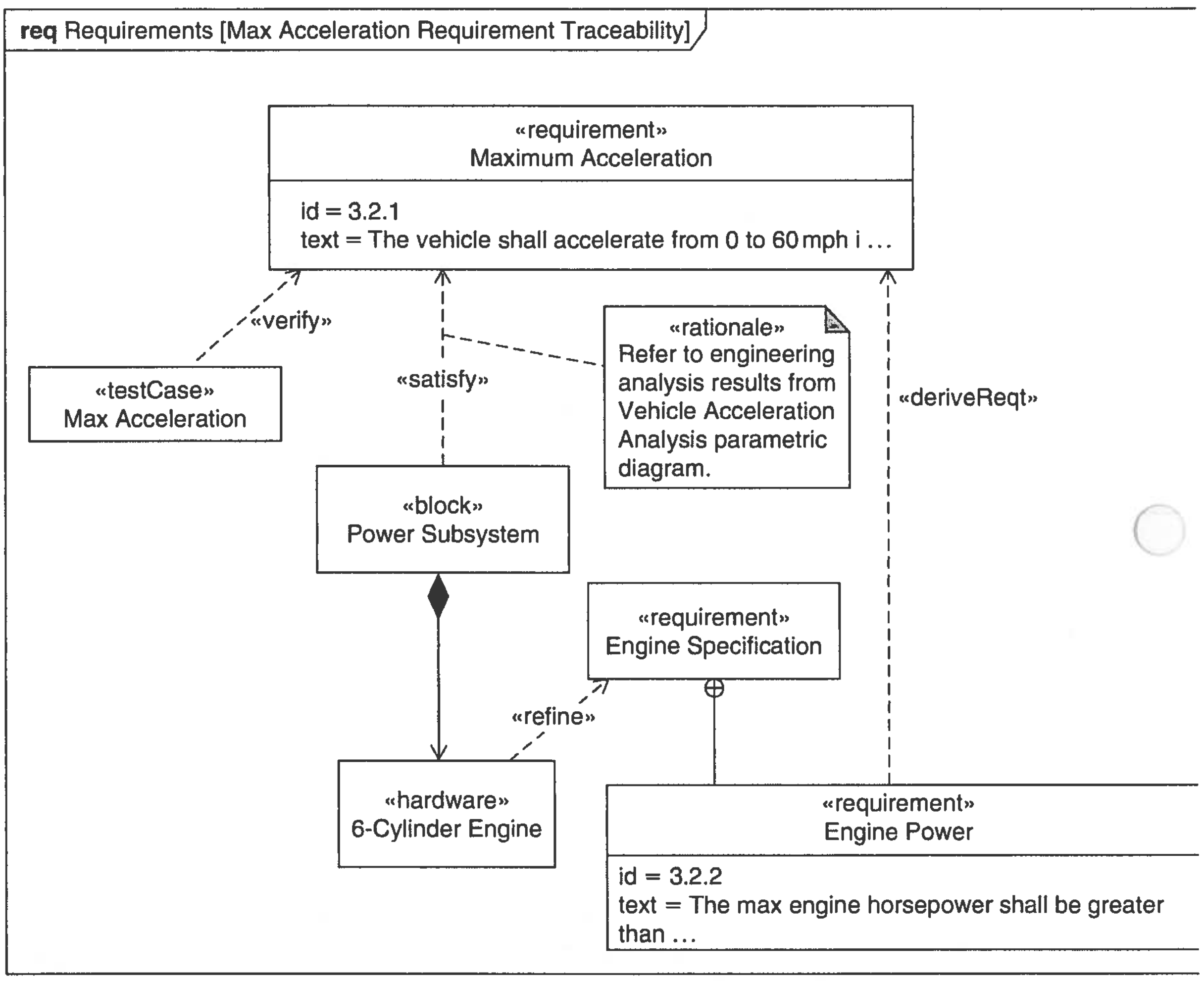
req Requirements [Max Acceleration Requirement Traceability]
«requirement» Maximum Acceleration
id = 3.2.1
text = The vehicle shall accelerate from 0 to 60 mph i ...
«verify»
«testCase» Max Acceleration
«satisfy»
«rationale» Refer to engineering analysis results from Vehicle Acceleration Analysis parametric diagram.
«deriveReqt»
«block» Power Subsystem
«requirement» Engine Specification
«refine»
«hardware» 6-Cylinder Engine
«requirement» Engine Power
id = 3.2.2
text = The max engine horsepower shall be greater than ...

FIGURE 3.18

Requirement diagram showing the traceability of the *Max Acceleration* requirement that was shown in the *Automobile Specification* in Figure 3.2. The traceability to a requirement includes the design elements that satisfy it, other requirements derived from it, and test cases to verify it. Rationale for the traceability relationships is also shown.

model elements and their relationships are captured in a model repository and can be displayed on diagrams. The model elements are integrated such that a model element that appears on one diagram may have relationships to model elements that appear on other diagrams. An example is the *Road* property, such as the *incline* angle that appears as a property of *Road* in the block definition diagram in Figure 3.3, and also is bound to a parameter of a constraint in the parametric diagram in Figure 3.14. The diagrams represent a view into this model.

A model organization is essential to managing the model. A well-organized model is akin 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. This facilitates understandability, access control, and change management of the model.

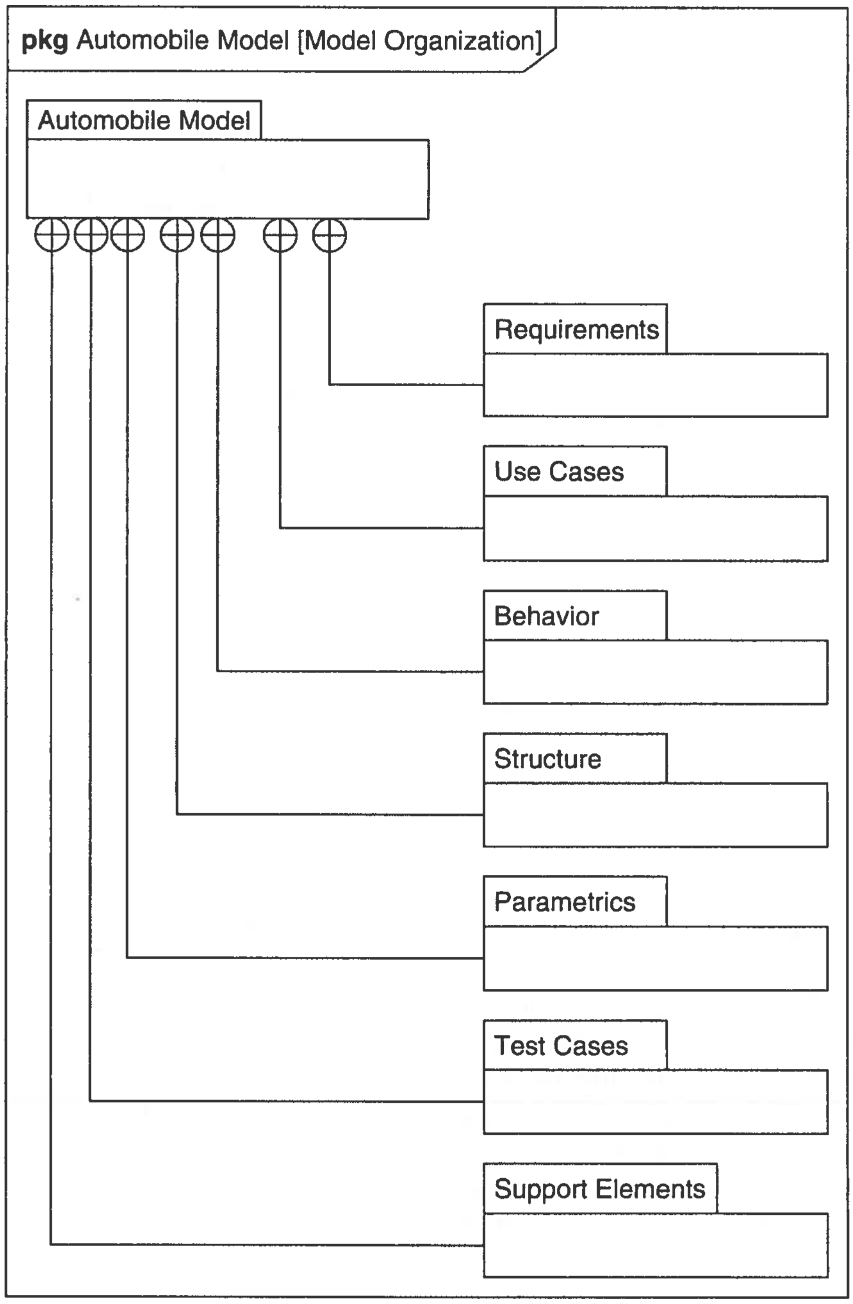

FIGURE 3.19

Package diagram showing how the model is organized into packages that contain model elements that comprise the *Automobile Domain*. Model elements in packages are displayed on diagrams. Model elements in one package can be related to model elements in another package.

The **package diagram** in Figure 3.19 shows how the model elements for this example are organized into **packages**. Each package **contains** a set of model elements. Model elements in one package can be related to model elements in another package. However, model organization enables each model element to be uniquely identified by the package that contains it. The model organization is generally similar to the view that is shown in the tool browser. Details on how to organize the model with packages are given in Chapter 5.

3.4.21 Model Interchange

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. An example may be the ability to export selected parts of the SysML model to another UML tool to support software development of the 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 interchange capability may be limited by the quality of the model and how the tool implements the standard, but this capability continues to improve. A description of XMI is included in Chapter 17.

3.5 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 semantics of the language enable a modeler to develop an integrated model where model elements on one diagram can be related to model elements on other diagrams. The diagrams enable capturing and viewing the information in the model repository to help specify, design, analyze, and verify systems. The repository information can be imported and exported to exchange model data via the XMI standard and other exchange mechanisms.

The SysML language is a critical enabler of MBSE and can be used with a variety of processes and methods. However, effective use of the language requires a well-defined MBSE method. The automobile example illustrated the use of one such method. Other examples are included in Part III.

3.6 Questions

1. What are some of the aspects of a system that SysML can represent?
2. What is a requirement diagram used for?
3. What is an activity diagram used for?
4. What is a sequence diagram used for?
5. What is a state machine diagram used for?
6. What is a use case diagram used for?
7. What is the primary unit of structure in SysML?
8. What is the block definition diagram used for?
9. What is an internal block diagram used for?
10. What is a parametric diagram used for?
11. What is a package diagram used for?