Modeling with SysML

Sanford Friedenthal, Lockheed Martin, Corp

Joseph A Wolfrom

Abstract

The OMG Systems Modeling Language (OMG SysML™) is a general-purpose, graphical, modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, it provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametric equations that can integrate with a broad range of engineering analysis. SysML represents a subset of UML 2.0 with extensions to satisfy the requirements of the UML™ for Systems Engineering RFP. The OMG SysML™ Specification was adopted in May 2006. For more information on SysML, including articles, tool vendor and related links, and the specification, go to http://www.omgsysml.org/.

This tutorial provides an introduction to how SysML can address your systems engineering needs, including: background and motivation, an overview of the SysML diagram types and language concepts,, a demonstration of how the language can be used throughout a Model-based Systems Engineering (MBSE) process. A class exercise is included to help solidify the student's understanding of the language. The course will also include a brief introduction to a typical SysML tool.

Attendees should gain appreciation of the value of model-based Systems Engineering versus legacy, document-centric methods; awareness of the graphical notation; and a high level understanding of when and how a Systems Engineer can exploit the various diagrams and models as part of their systems engineering process.

Modeling with SysML

Sanford Friedenthal, Lockheed Martin, Corp
Joseph A Wolfrom

Biographies

Sanford A. Friedenthal is a Principal Systems Engineer at Lockheed Martin leading an effort to enable model based systems development (MBSD) across the corporation. Mr. Friedenthal's experience includes the application of systems engineering throughout the system life cycle from conceptual design, through development and production on a broad range of systems in aerospace and defense. He has been a systems engineering department manager, and a lead developer of advanced systems engineering processes and methods including the Lockheed Martin Integrated Enterprise Process and the Object-Oriented Systems Engineering Method (OOSEM). Mr. Friedenthal was a leader of the Industry Standards effort through the Object Management Group (OMG) and INCOSE to develop the Systems Modeling Language (OMG SysML ™) that was adopted by the OMG in 2006. He also co-authored the book, "A Practical Guide to SysML".

Joseph A. Wolfrom is a Senior Professional Staff member of the Aerospace Systems Analysis Group in the Global Engagement Department of the Johns Hopkins University Applied Physics Laboratory (JHU/APL). Mr. Wolfrom serves as APL's lead system architect for the U.S Air Force Airborne Electronic Attack System of Systems architecture and numerous other Department of Defense programs. Mr. Wolfrom recently developed an APL-internal course entitled "Model-Based Systems Engineering with the Systems Modeling Language" and taught the course to 18 members of APL's technical staff.

Modeling with SysML

Instructors:

Sanford Friedenthal

sanford.friedenthal@lmco.com

Joseph Wolfrom

joe.wolfrom@jhuapl.edu

Content Developer

OBJECT MANAGEMENT GROUP • CERTIFICATION PROGRAM • SM

APL

The Johns Hopkins University

APPLIED PHYSICS LABORATORY

OMG SysML™ Specification

- Specification status
  - Adopted by OMG in May ’06
  - Available Specification v1.0 in Sept ’07
  - Available Specification v1.1 in Nov ‘08
  - Available Specification for v1.2 in March ‘10
  - Revision Task Force for v1.3 in process
- Multiple vendor implementations available
- This tutorial is based on:
  - OMG SysML available specification (formal/2007-09-01) and
  - OMG/INCOSE tutorial by Friedenthal, Moore, and Steiner
  - “A Practical Guide to SysML” by Friedenthal, Moore, and Steiner
  - Tutorial Material from JHU/APL Course developed by Joe Wolfrom
- This OMG tutorial, specifications, papers, and vendor info can be found on the OMG SysML Website at http://www.omgsysml.org/

Agenda

- Introduction
- SysML Diagram Overview
- Introduction to a Modeling Tool
- Language Concepts and Constructs
- Class Exercise
- Process Summary
- Tools Overview
- Wrap-up

Objectives & Intended Audience

At the end of this tutorial, you should have an awareness of:

- **Motivation of model-based systems engineering approach**
- **SysML diagrams and basic language concepts**
- **How SysML is used as part of an MBSE process**

This course is not intended to make you a systems modeler!

You must use the language.

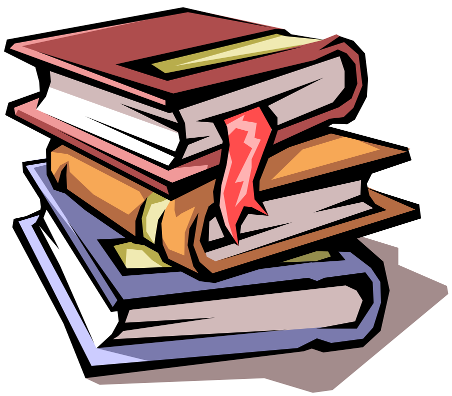
Intended Audience:

- **Practicing Systems Engineers interested in system modeling**
- **Software Engineers who want to better understand how to integrate software and system models**
- **Familiarity with UML is not required, but it helps**

INTRODUCTION

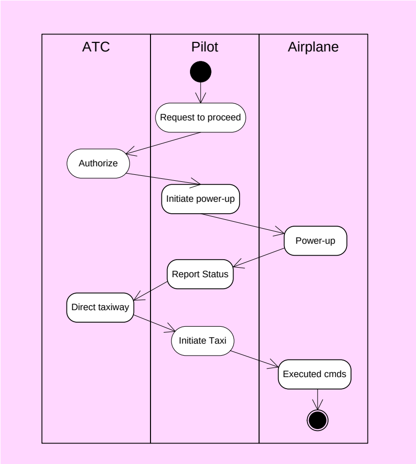
SE Practices for Describing Systems

Past



- Specifications
- Interface requirements
- System design
- Analysis & Trade-off
- Test plans

Future

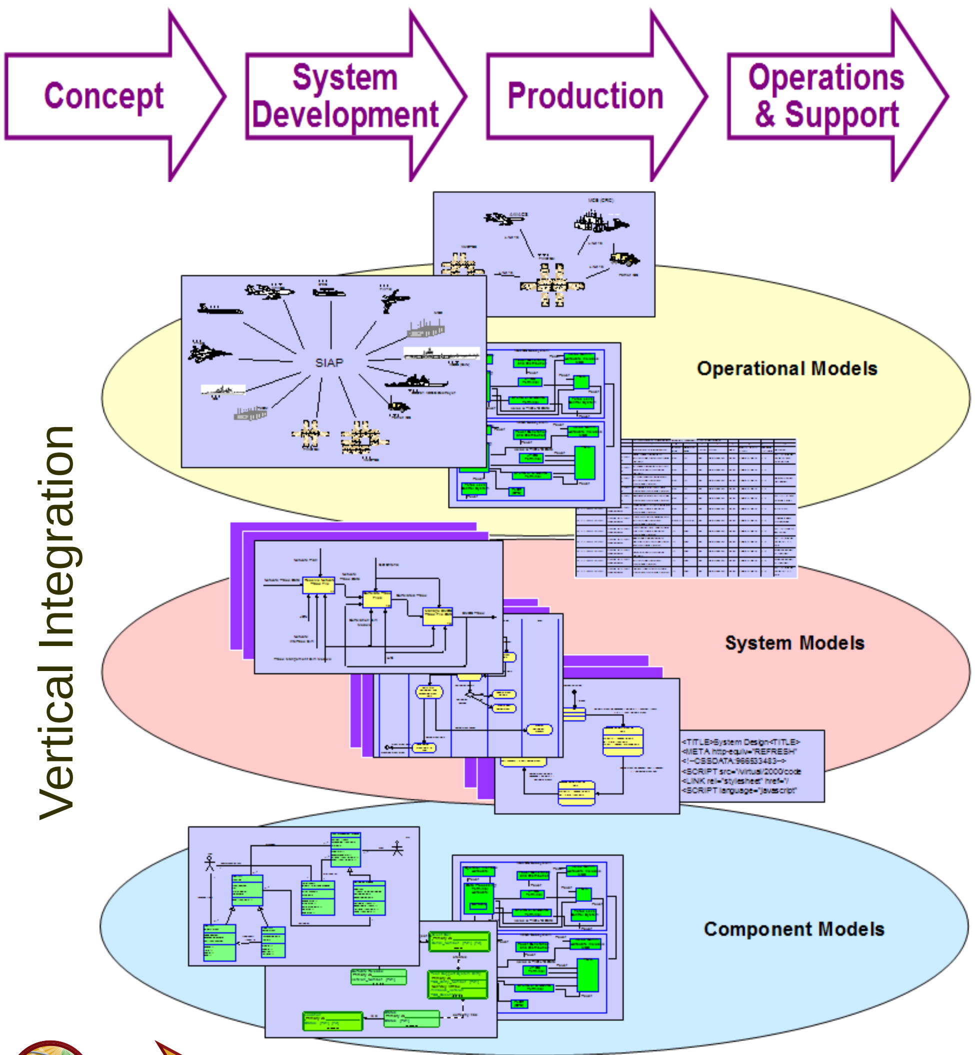


Moving from Document centric to Model centric

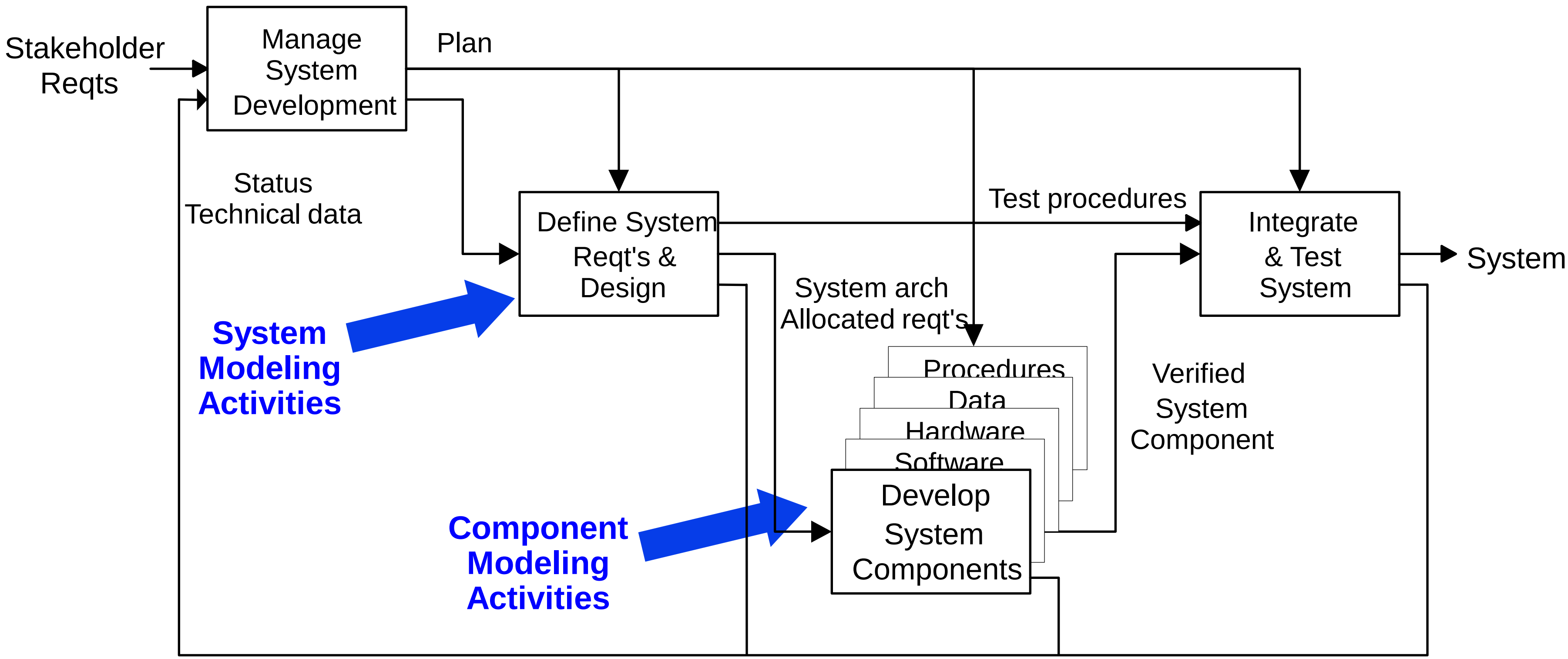
Model-based Systems Engineering (MBSE)

- Formalizes the practice of systems development through use of models
- Broad in scope
  - Integrates with multiple modeling domains across life cycle from system of systems to component
- Results in quality/productivity improvements & lower risk
  - Rigor and precision
  - Communications among system/project stakeholders
  - Management of complexity

Life Cycle Support



System Development Process



Integrated Product Development (IPD) is essential to improve communications

A Recursive V process that can be applied to multiple levels of the system hierarchy

System Modeling Activities – OOSEM

Integrating MBSE into the SE Process

Major SE Development Activities

Analyze Needs
- Causal analysis
- Mission use cases/scenarios
- Enterprise model

Define System Requirements
- System use cases/scenarios
- Elaborated context

Define Logical Architecture
- Logical decomposition
- Logical scenarios
- Logical subsystems

Synthesize Allocated Architecture
- Node diagram
- HW, SW, Data arch
- System deployment

Optimize & Evaluate Alternatives
- Parametric Diag
- Trade study

Manage Requirements
- Reqt's Diagram & tables

Support Validation & Verification
- Test cases
- Test procedures

Common Subactivities

SysML Diagram Types

SysML includes nine diagrams as shown in this diagram:

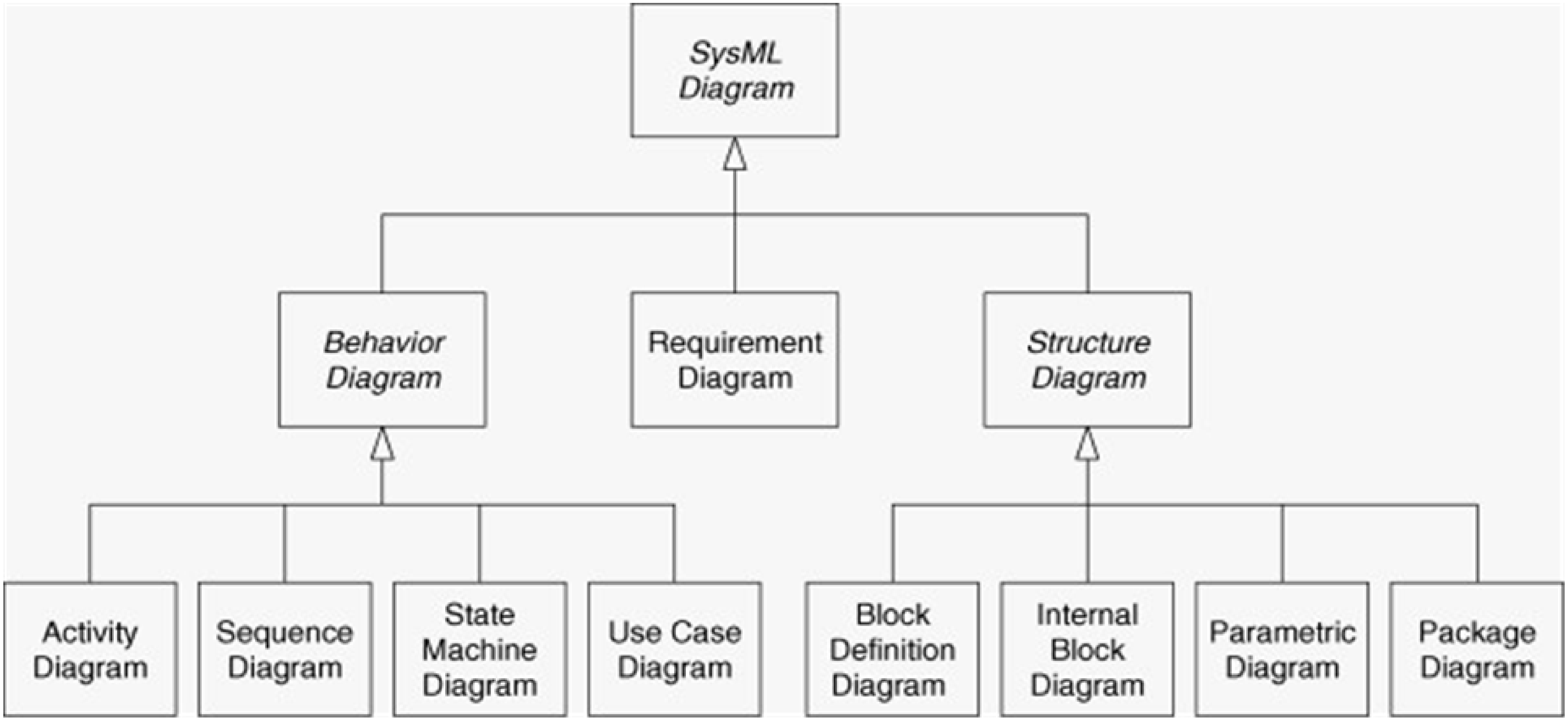


FIGURE 3.1

4 Pillars of SysML – ABS Example

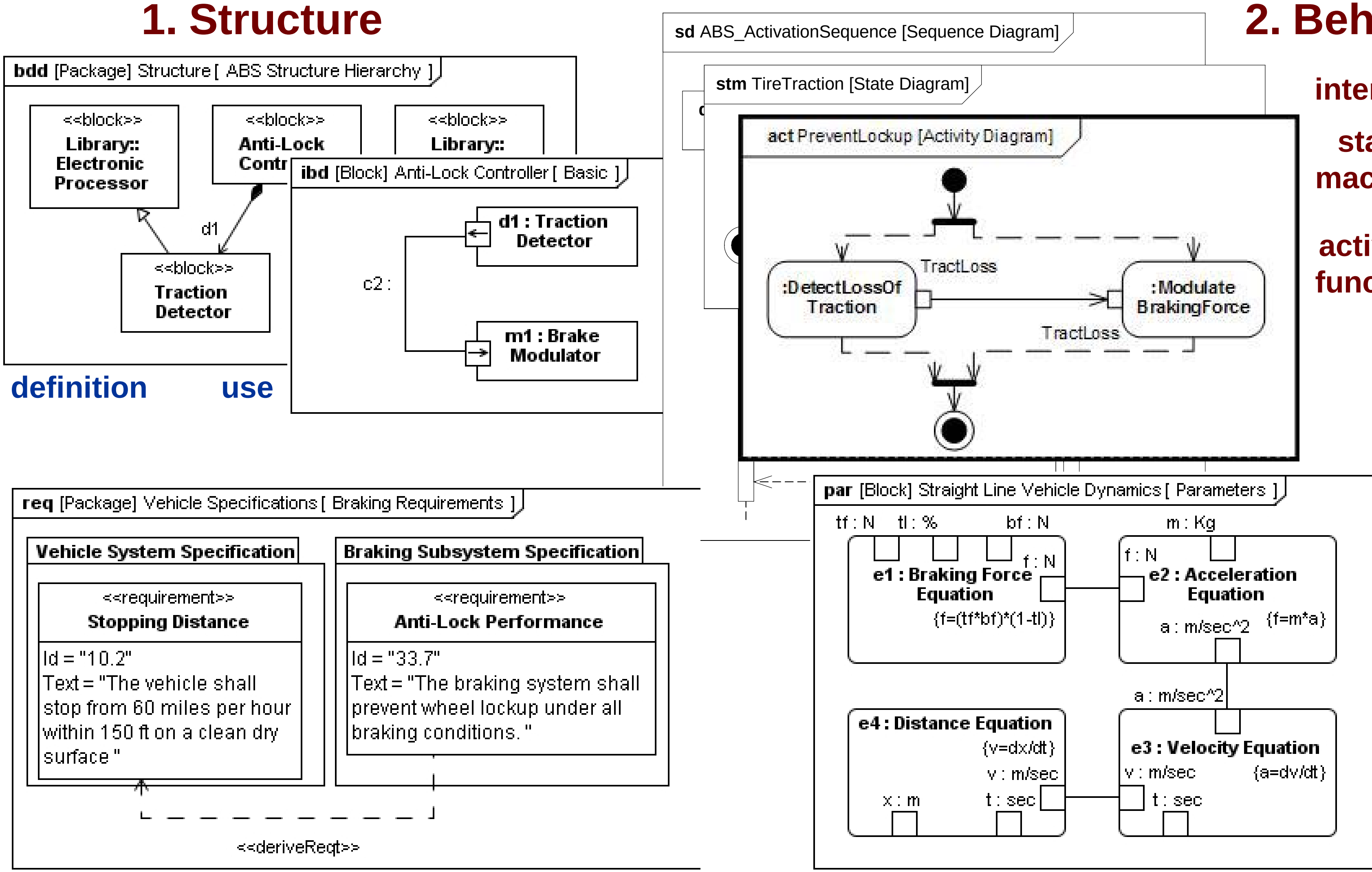
1. Structure

2. Behavior

interaction

state machine

activity/ function



3. Requirements

•4. Parameretrics

APL

SYSML DIAGRAM OVERVIEW

SysML Diagram Frames

- **Each SysML Diagram must have a diagram frame**
- **Each SysML diagram frame represents a model element**
- **Diagram context is indicated in the header:**
  - Diagram kind (act, bdd, ibd, sd, etc.)
  - Model element type (package, block, activity, etc.)
  - Model element name
  - User defined diagram name or view name
- **A separate diagram description block is used to indicate if the diagram is complete, or has elements elided**

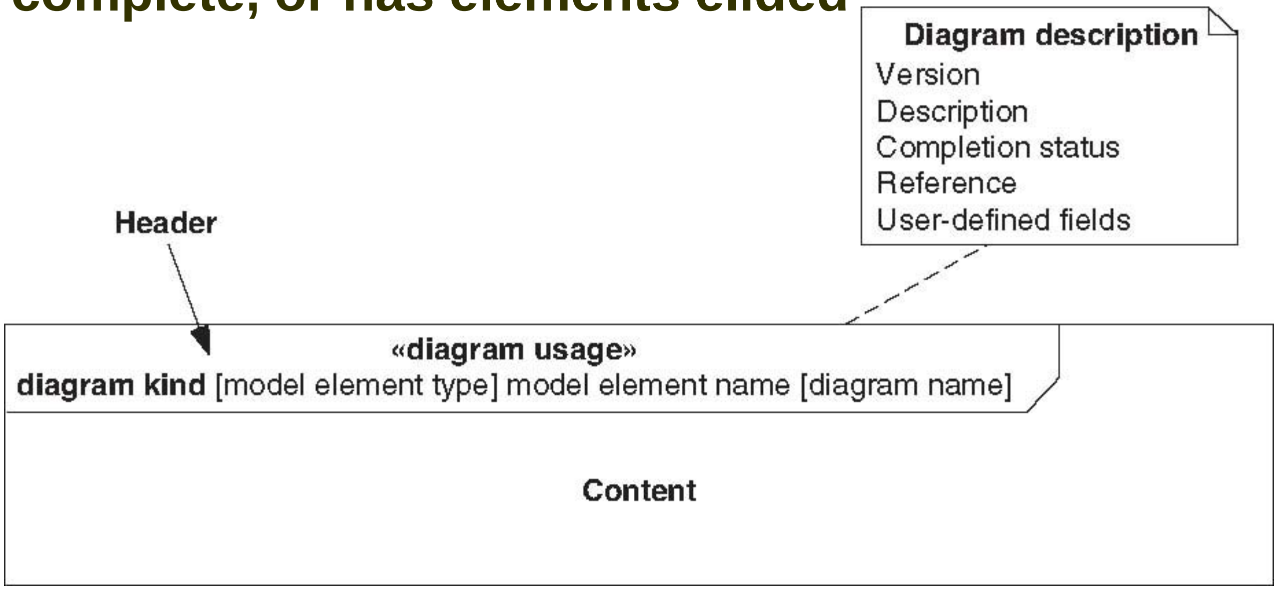
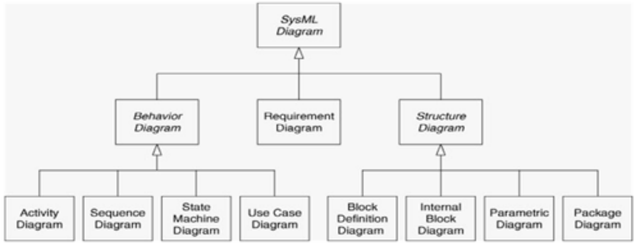


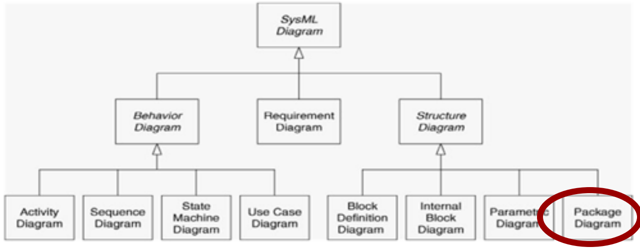
FIGURE 4.8

SysML Diagrams



- Package diagram
- Requirement diagram
- Use Case diagram
- Block Definition diagram
- Internal Block diagram
- Activity diagram
- Sequence diagram
- State Machine diagram
- Parametric diagram

Package Diagram



- **Represents the organization of a model in terms of packages that contain model elements**

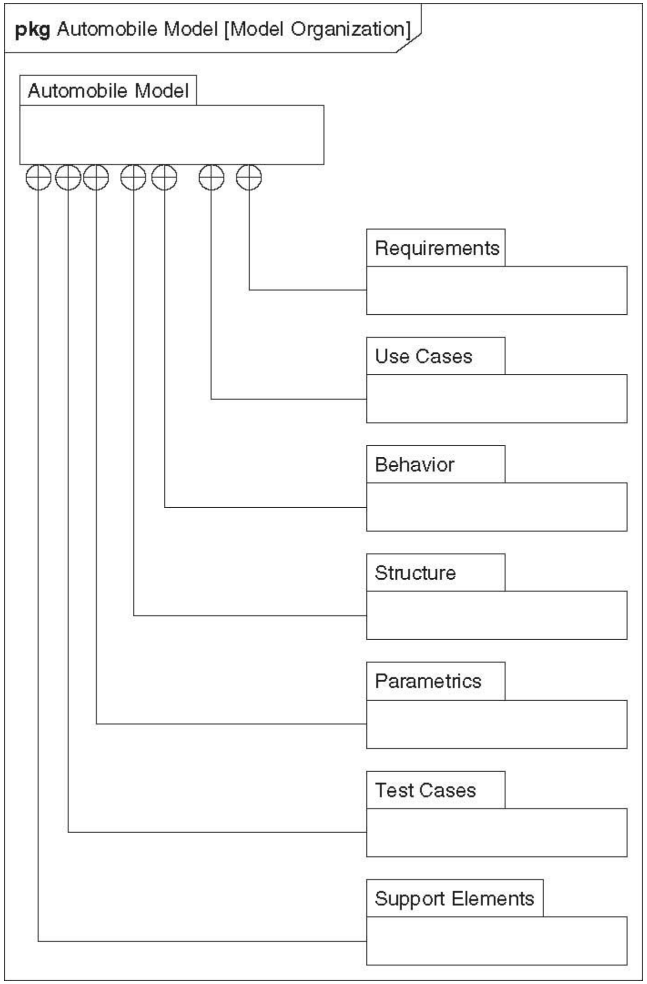
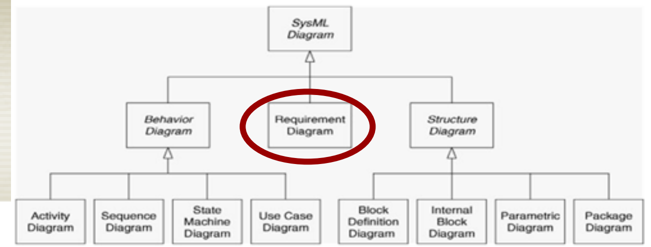


FIGURE 3.19

Requirement Diagram



- **Represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability**

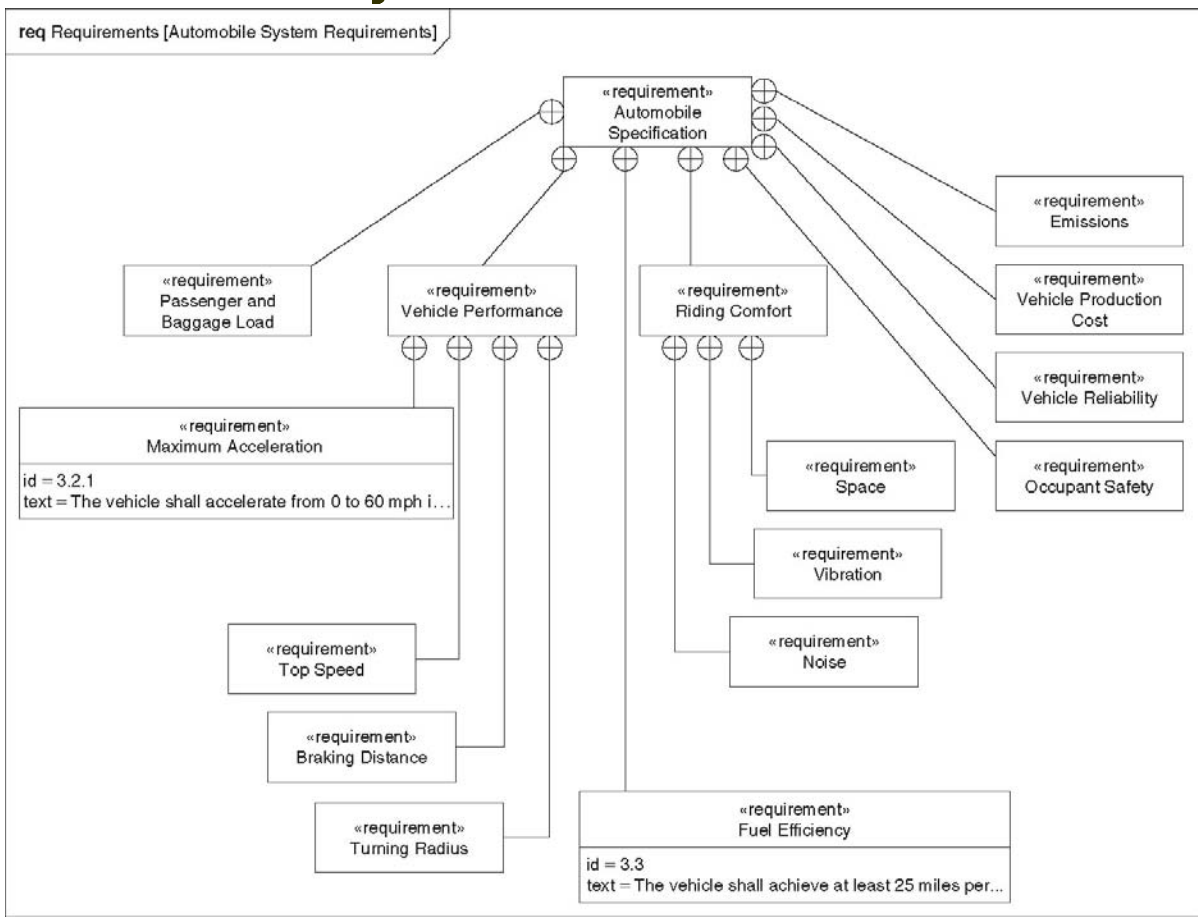
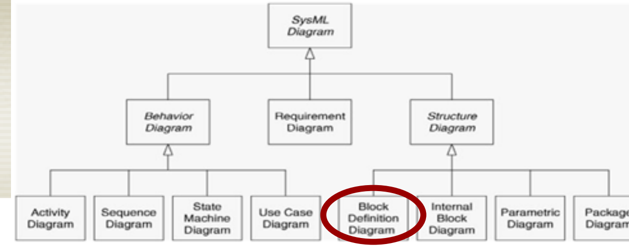


FIGURE 3.2

© 2008 Elsevier, Inc.: A Practical Guide to SysML

Block Definition Diagram



- **Represents structural elements called blocks, and their composition and classification**

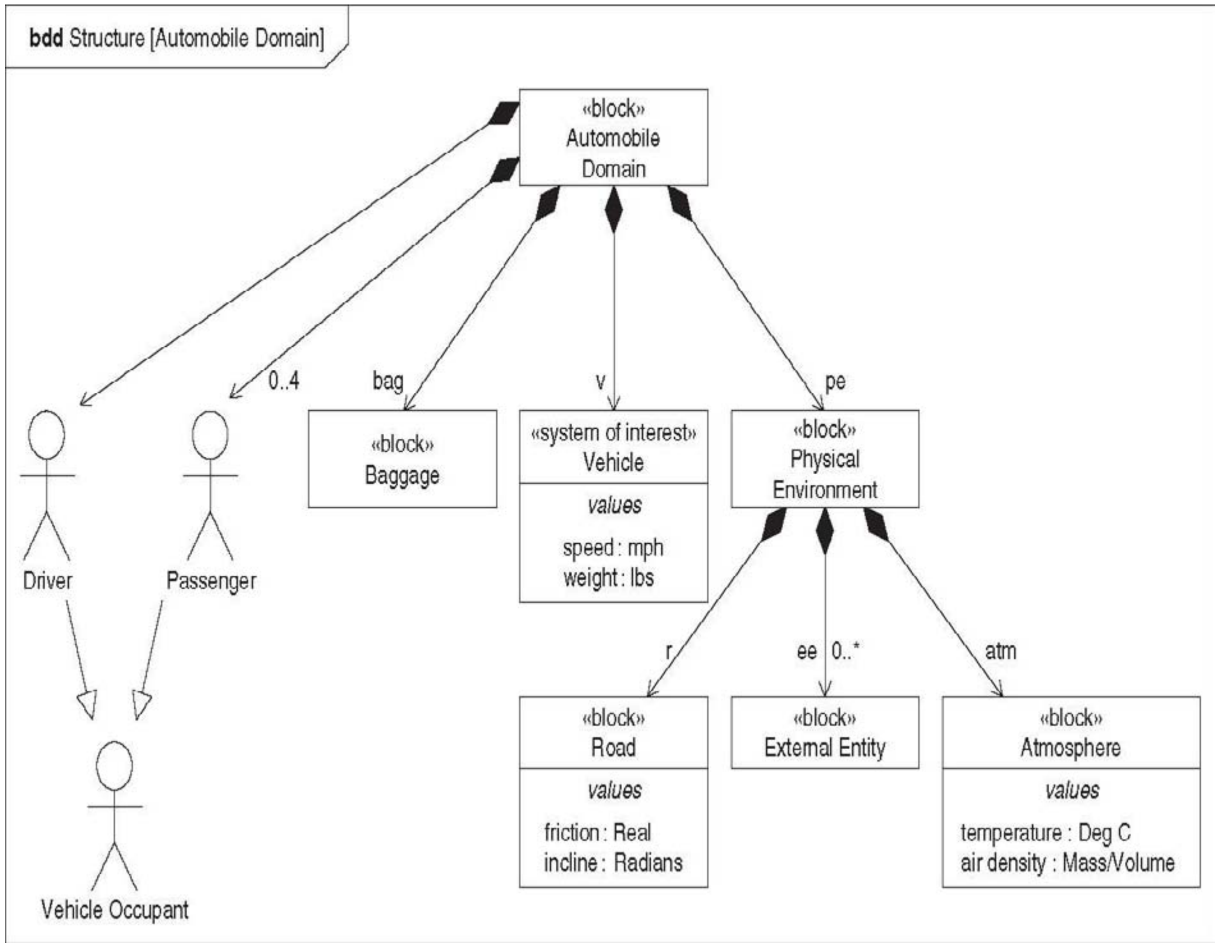
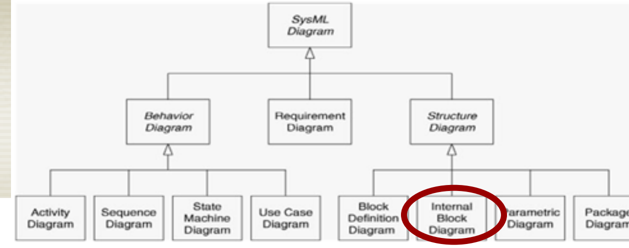


FIGURE 3.3

Internal Block Diagram



- **Represents interconnection and interfaces between the parts of a block**

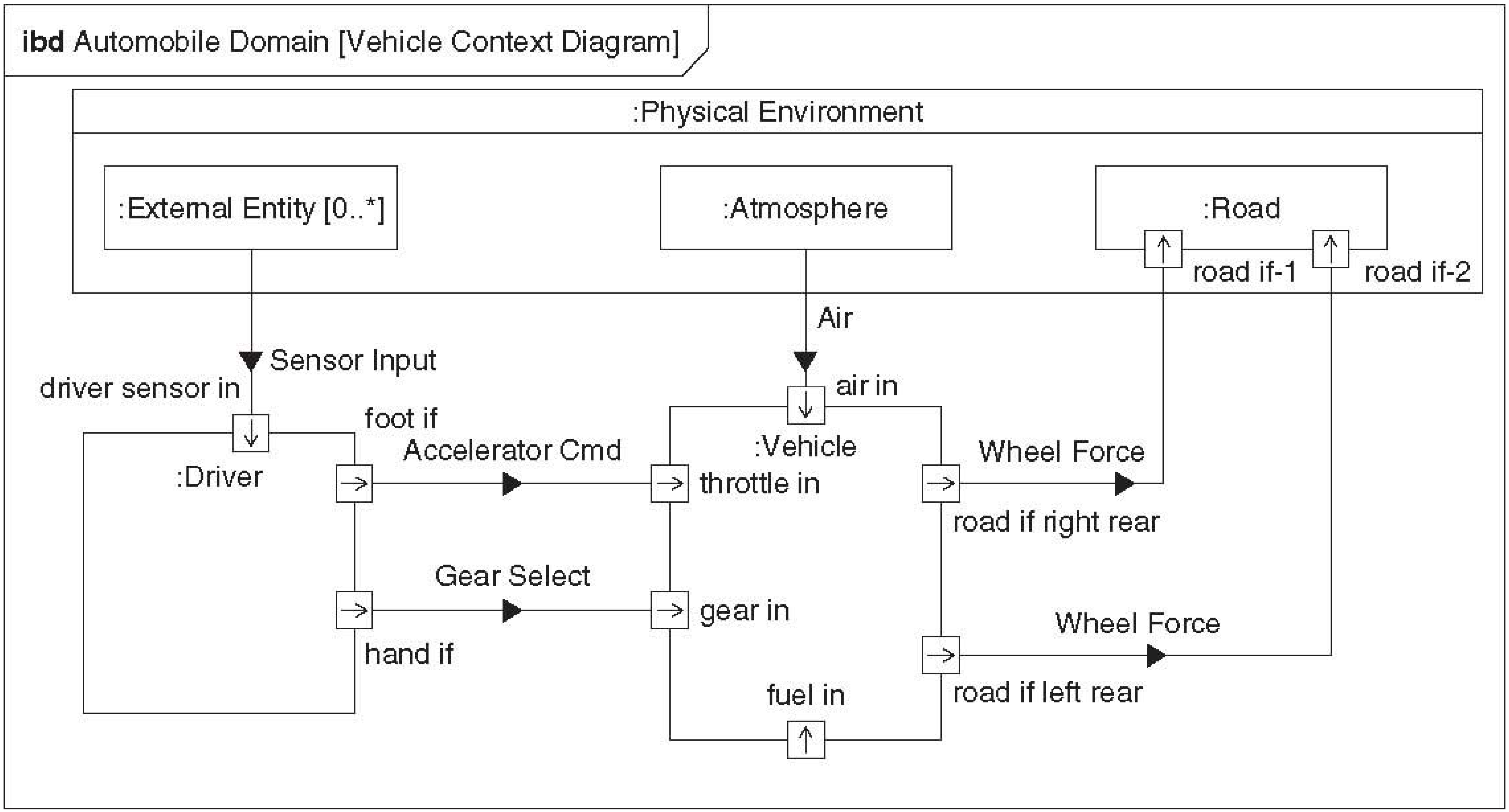
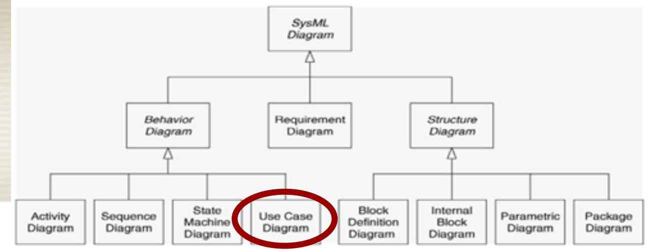


FIGURE 3.9

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**

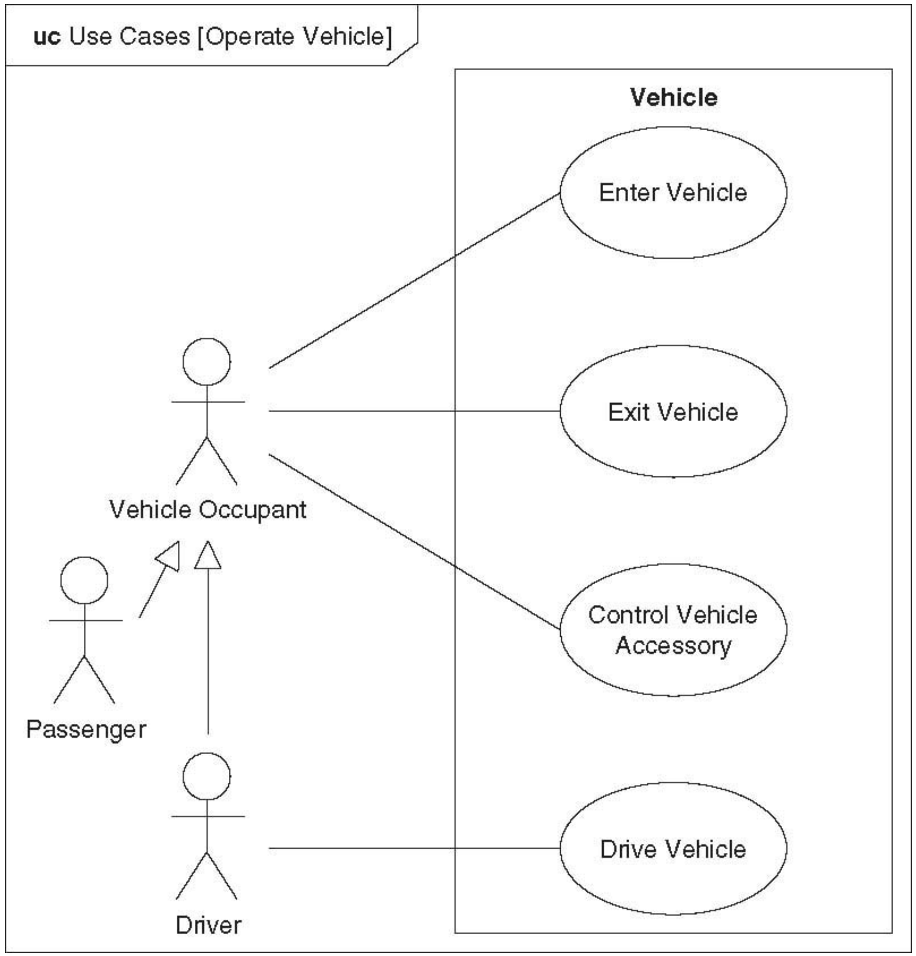
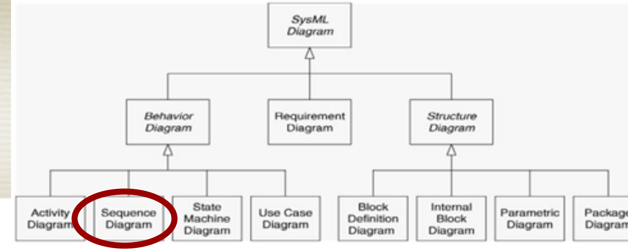


FIGURE 3.4

Drive Vehicle Sequence Diagram



- Represents behavior in terms of a sequence of messages exchanged between parts

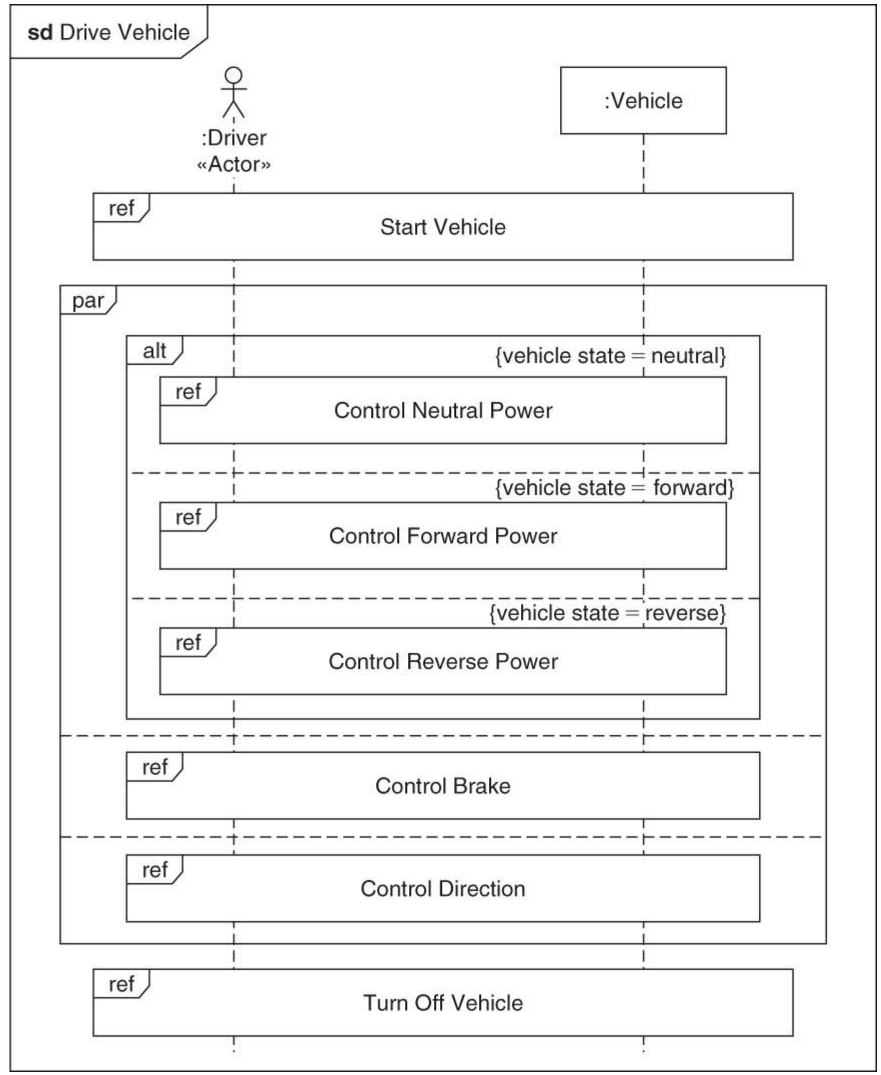
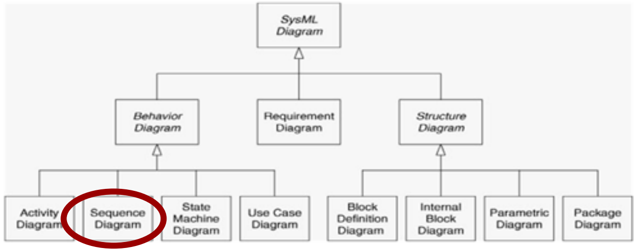


FIGURE 3.5

© 2008 Elsevier, Inc.: A Practical Guide to SysML

Start Vehicle Sequence Diagram



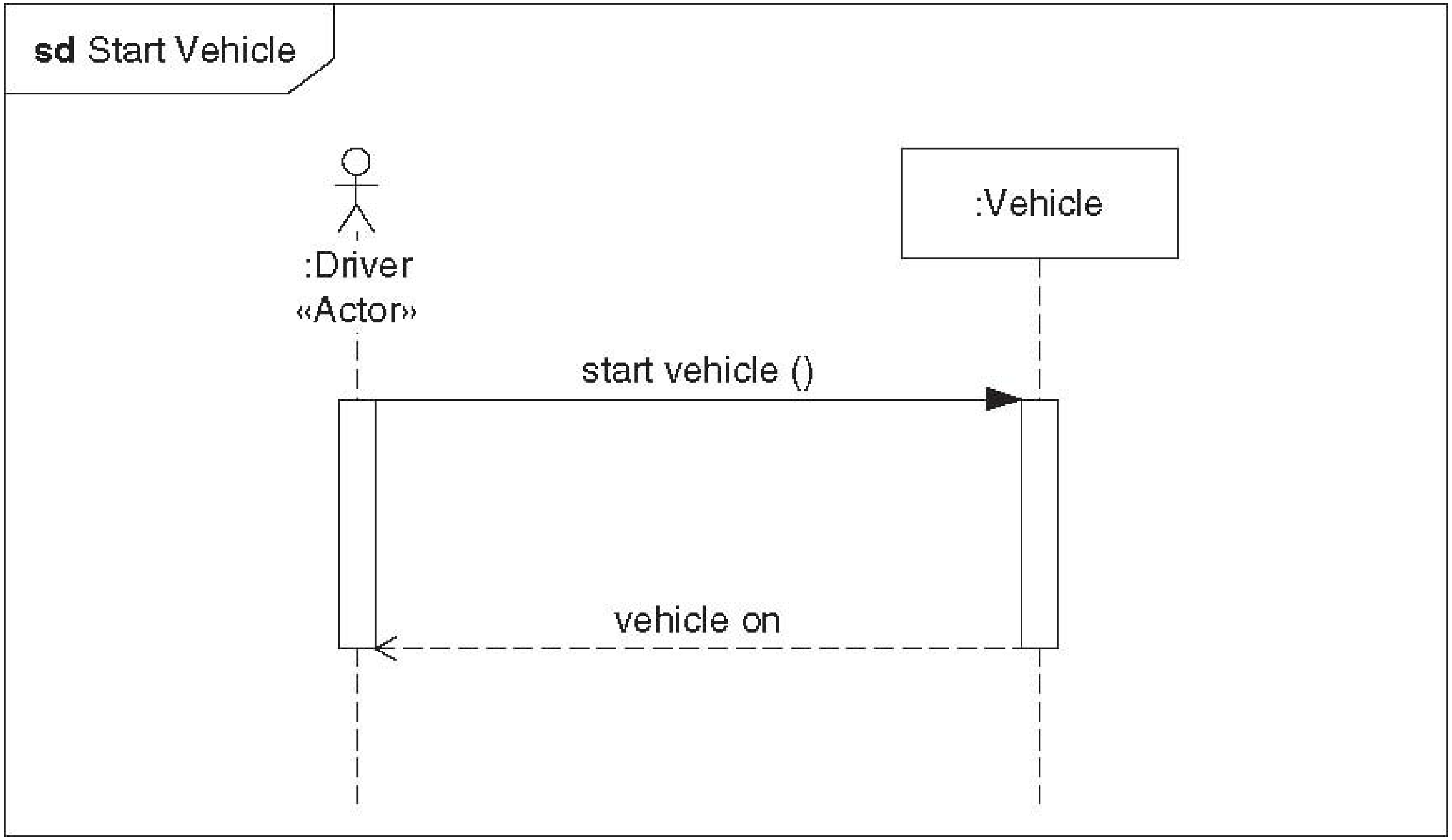
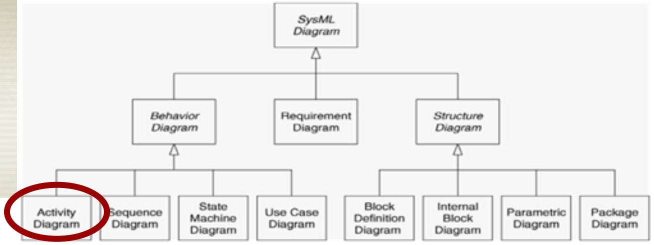


FIGURE 3.6

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**

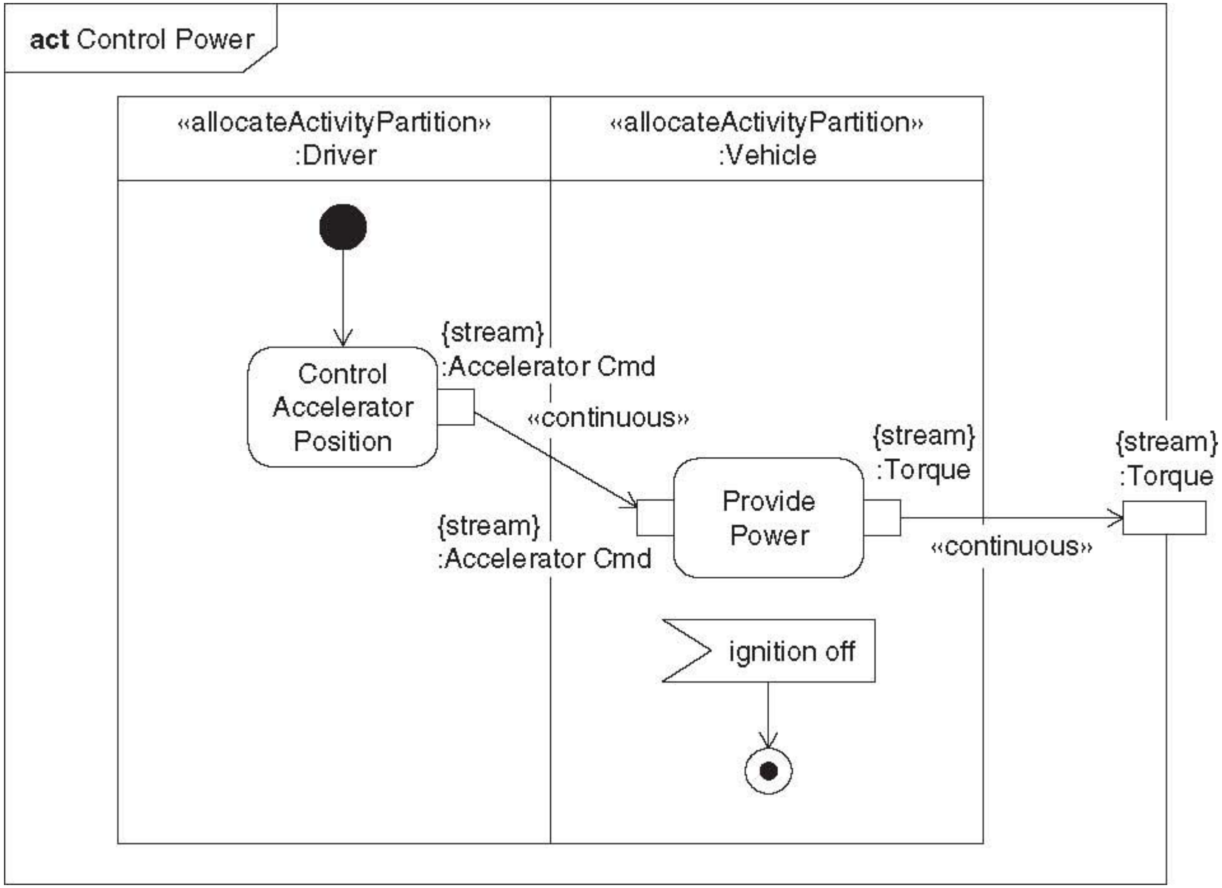
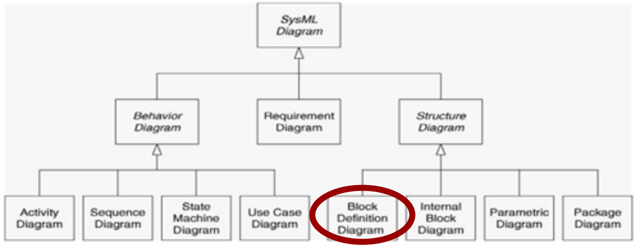


FIGURE 3.7

Vehicle System Hierarchy Block Definition Diagram



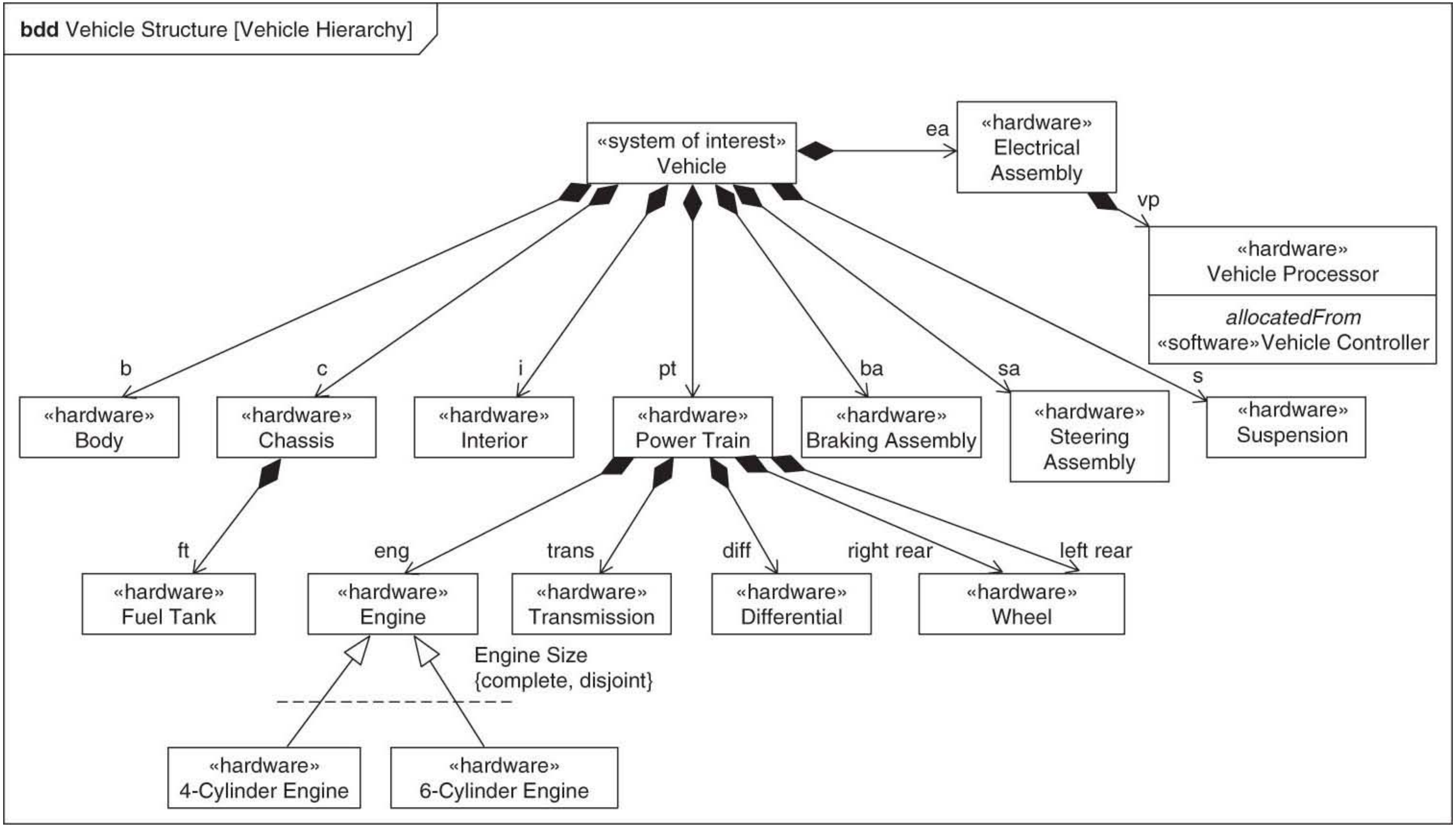
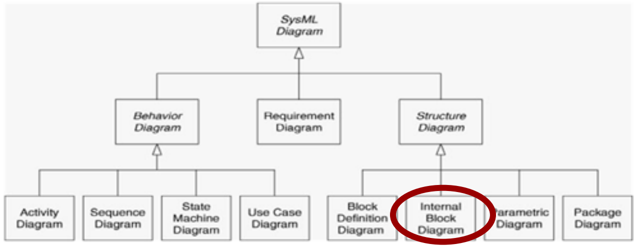


FIGURE 3.10

Power Subsystem Internal Block Diagram



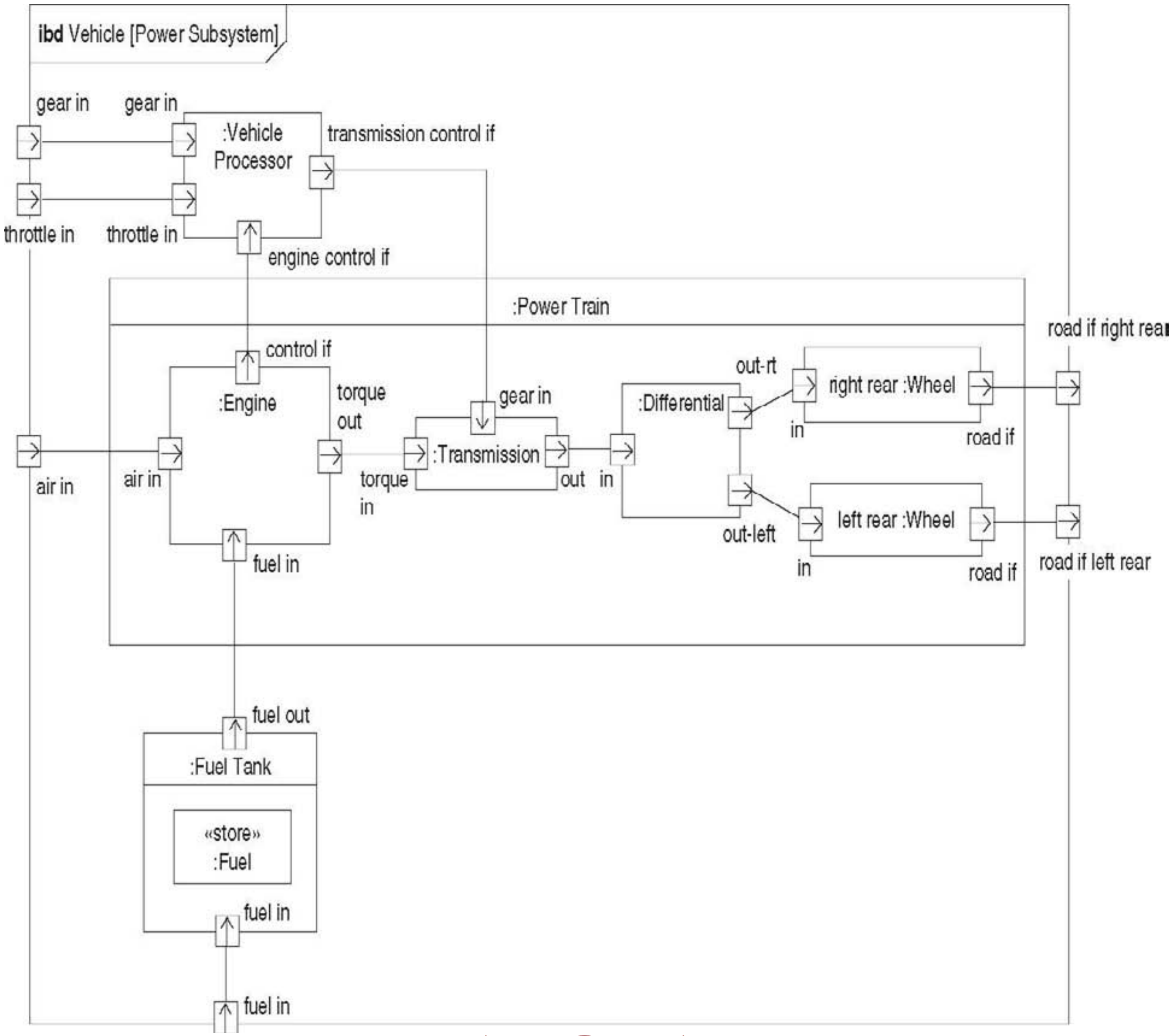
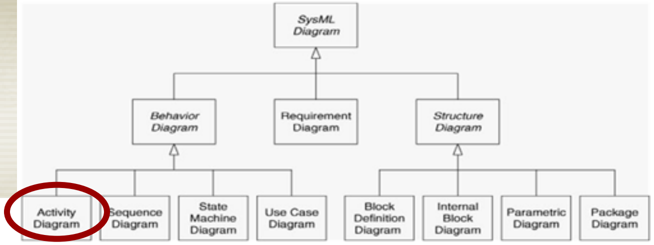
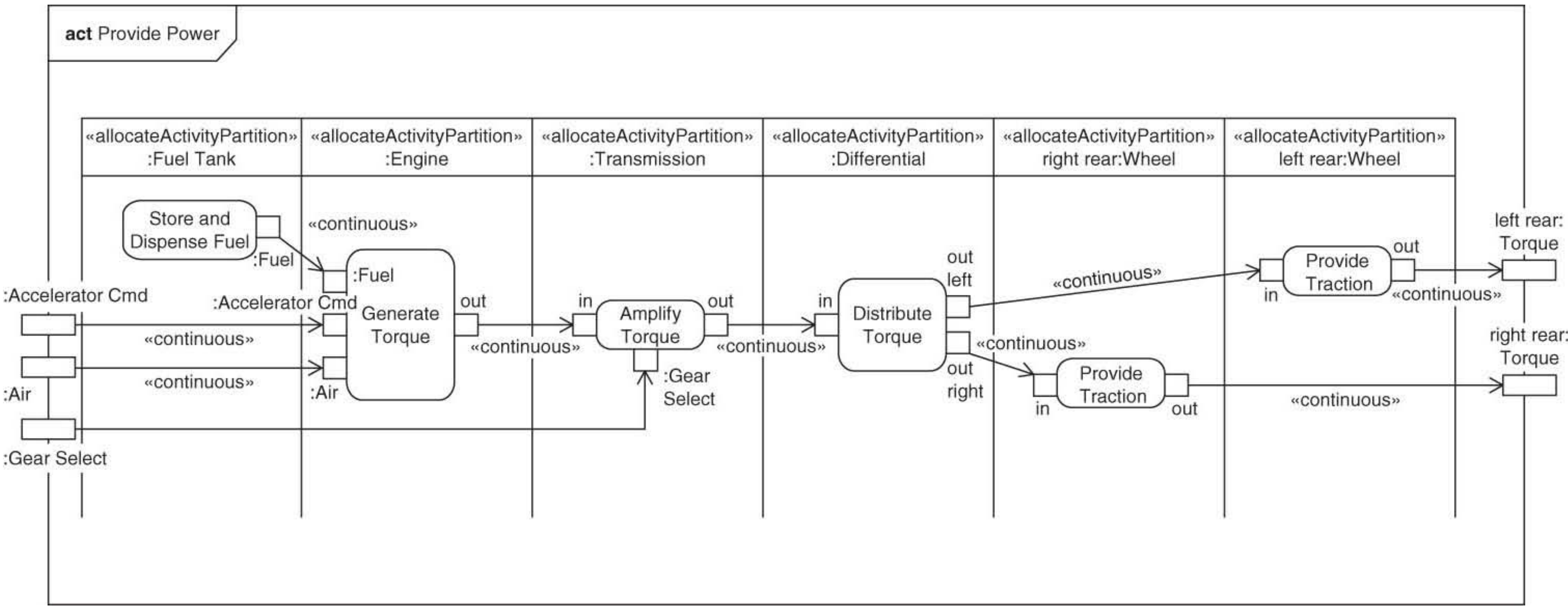


FIGURE 3.12

Provide Power Activity Diagram

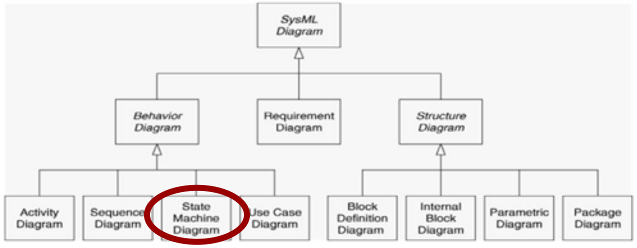




© 2008 Elsevier, Inc.: A Practical Guide to SysML

FIGURE 3.11

State Machine Diagram



- **Represents behavior of an entity in terms of its transitions between states triggered by events**

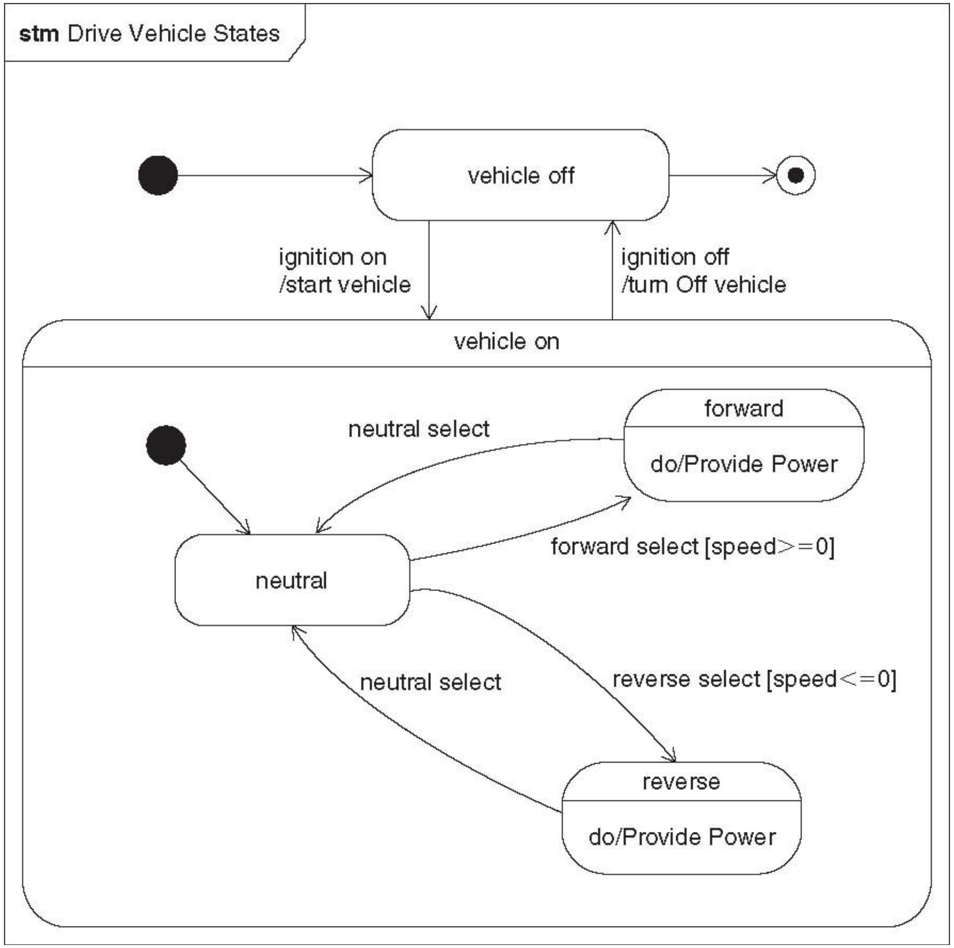
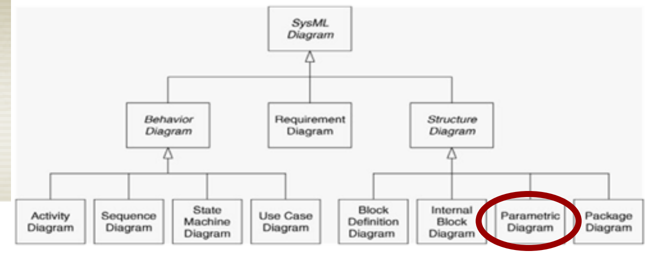


FIGURE 3.8

Parametric Diagram



- Represents constraints on property values, such as F=m*a, used to support engineering analysis

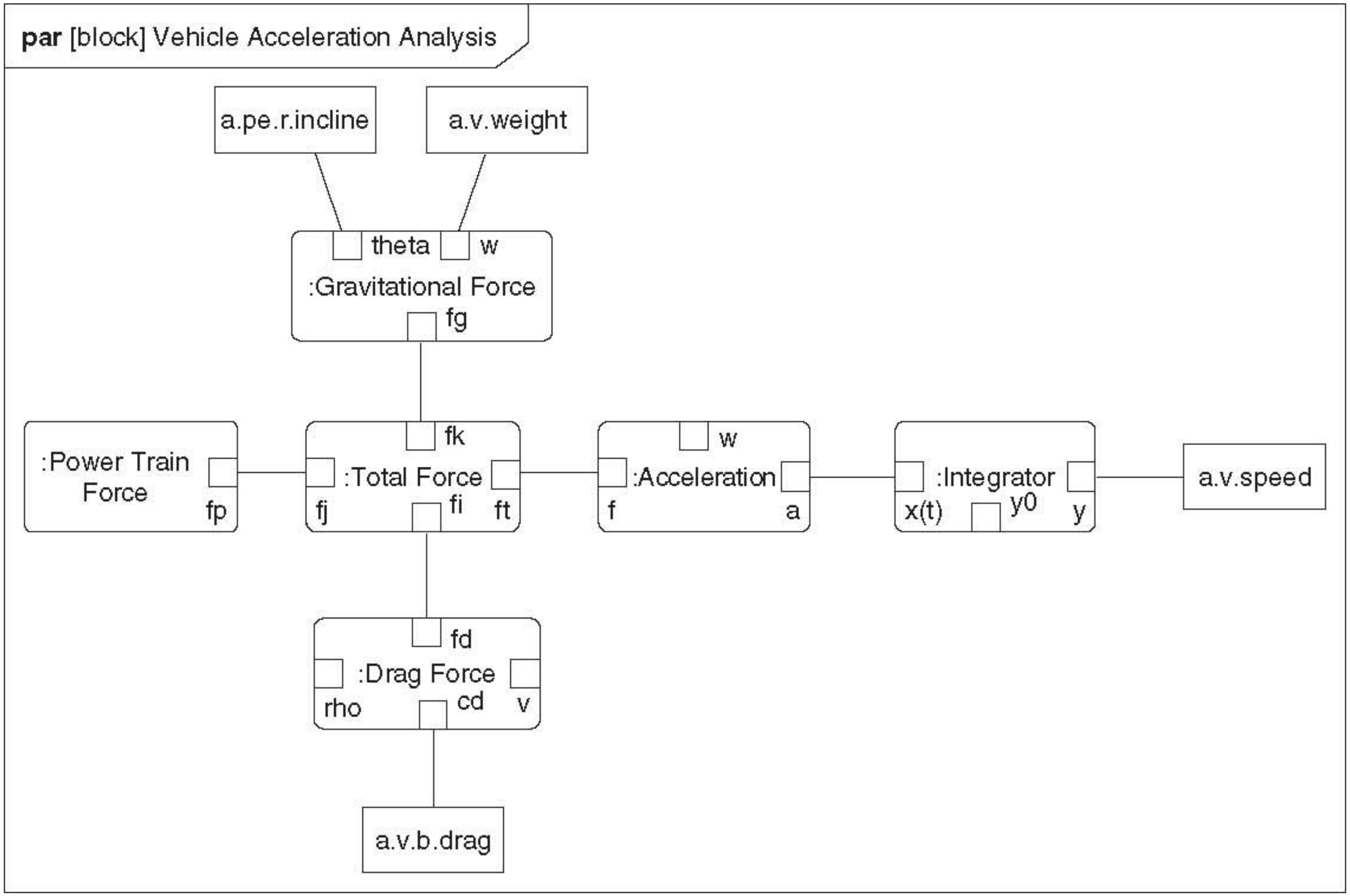
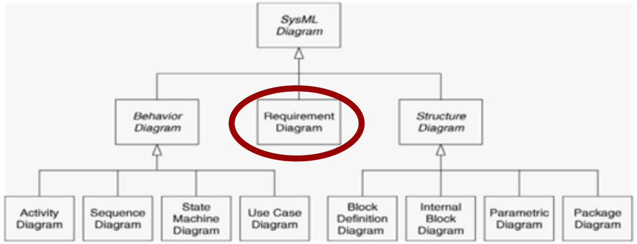


FIGURE 3.14

Requirements Traceability



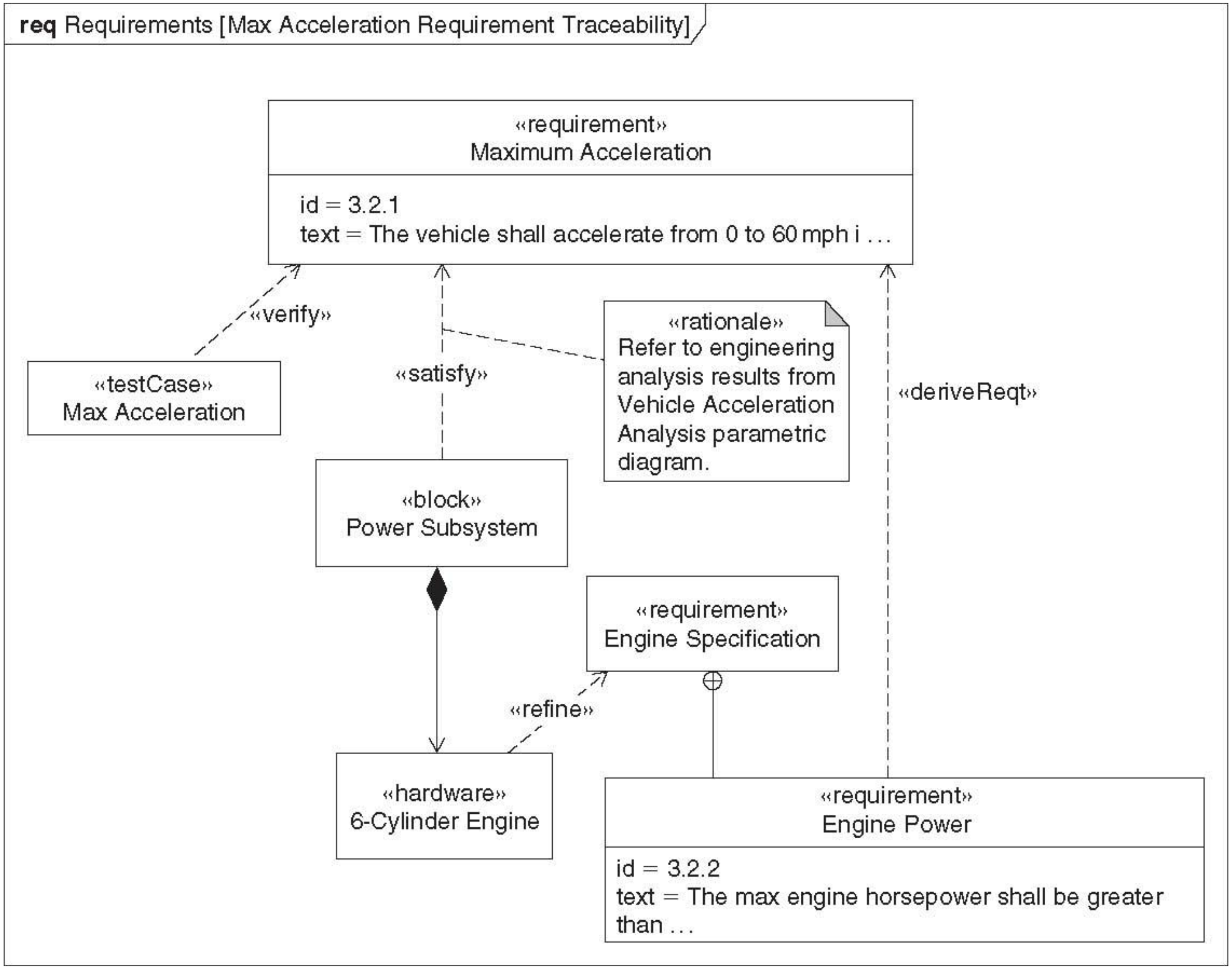
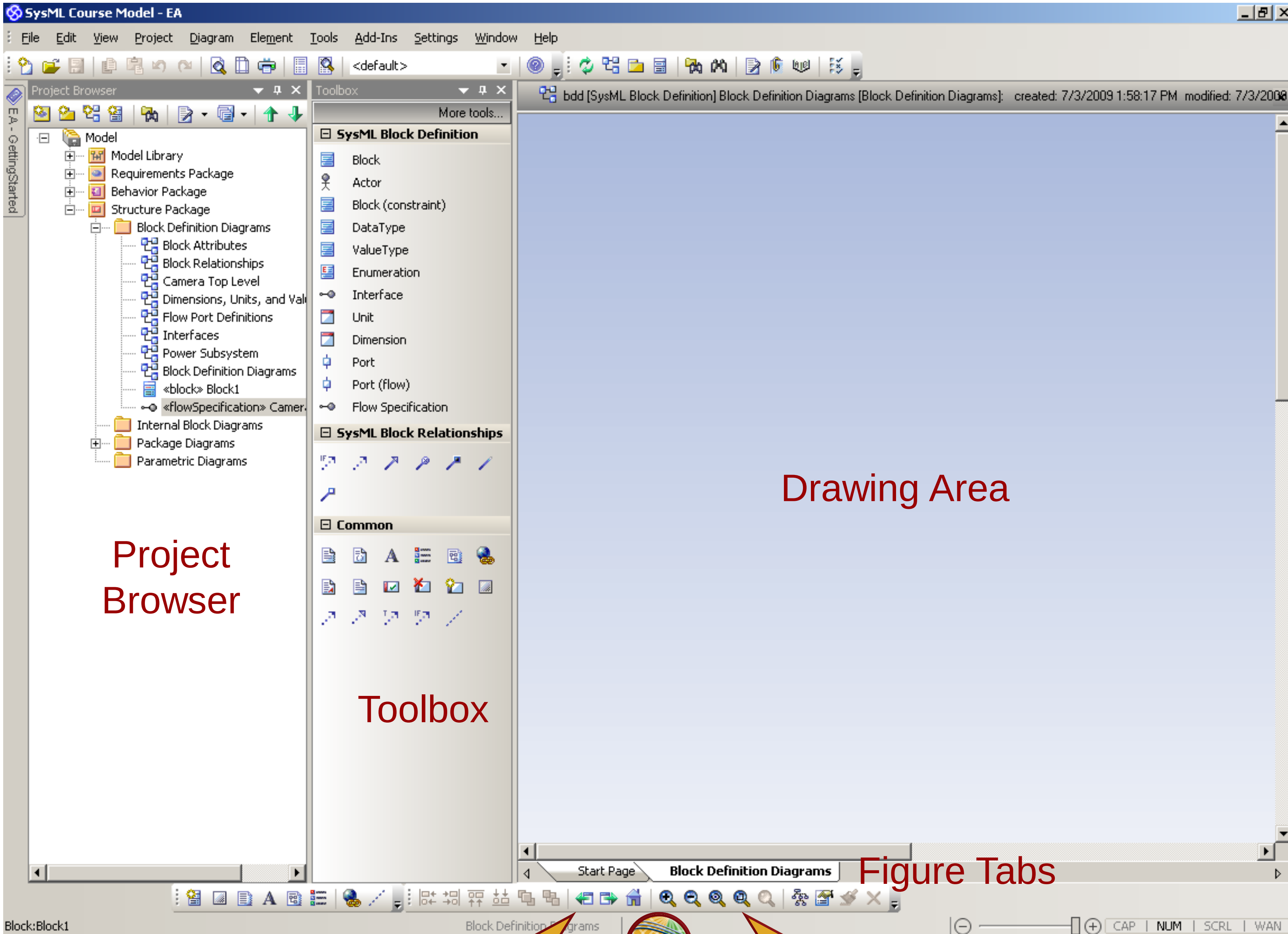


FIGURE 3.18

INTRODUCTION TO A MODELING TOOL

Typical Work Area Components



LANGUAGE CONCEPTS AND CONSTRUCTS

Agenda

- **Language Concepts and Constructs**
  - Organizing the Model with Packages
  - Capturing Text-Based Requirements in the Model
  - Modeling High Level Functionality with Use Cases
  - Modeling Structure With Blocks
    - Modeling Blocks and Their Relationships on a BDD
    - Modeling Part Interconnection on an IBD
  - Modeling Behavior
    - Flow-based Behavior with Activities
    - Message-based Behavior with Interactions
    - Event-based Behavior with State Machines
  - Modeling Constraints with Parametrics
  - Modeling Cross Cutting Relationships with Allocations

ORGANIZING THE MODEL WITH PACKAGES

Packages

- Packages are used to organize the model
  - Groups model elements into a name space
  - Often represented in tool browser
  - Supports model configuration management (check-in/out)
- Model can be organized in multiple ways
  - By System hierarchy (e.g., enterprise, system, component)
  - By diagram kine (e.g., requirements, use cases, behavior)
  - Use viewpoints to augment model organization
- Package Diagrams provide a graphical depiction of the model organization and/or package content

Package Diagram for Automobile Model

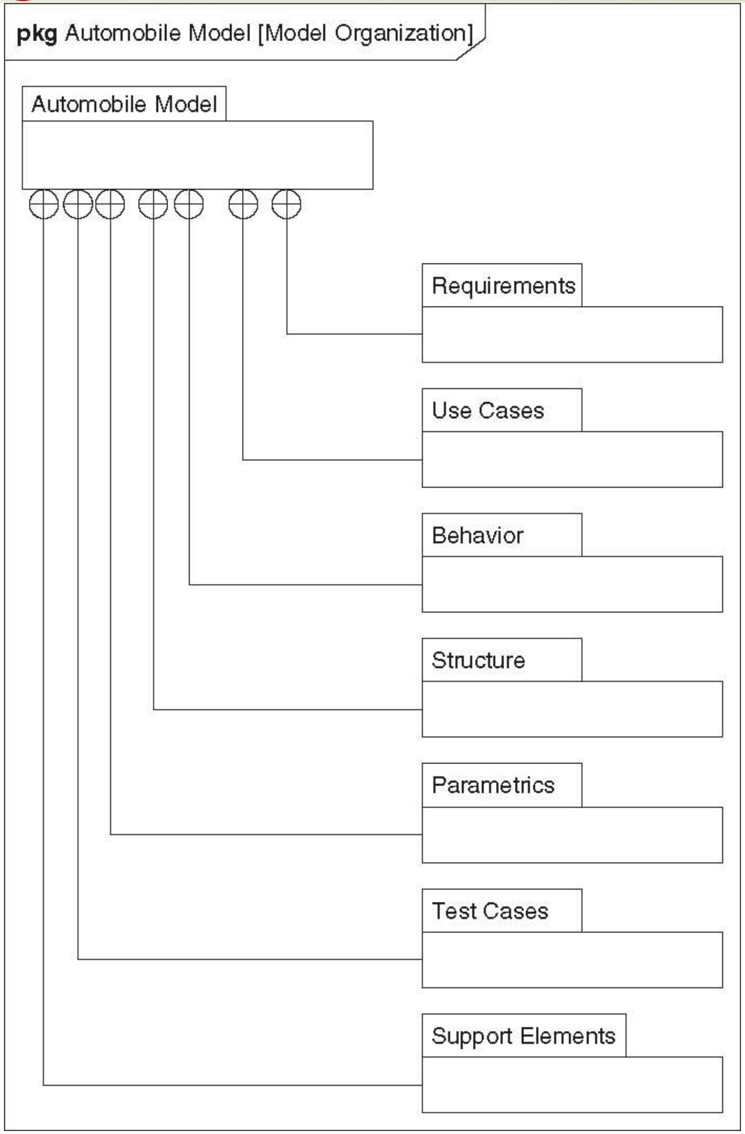


FIGURE 3.19

Package Diagram Containment Relationship

- Depicts Package Hierarchy
- Three techniques (displayed below)
  - Packages contained within ‘frame’ of parent package
  - Packages contained within a package
  - Crosshair pointing to the parent package

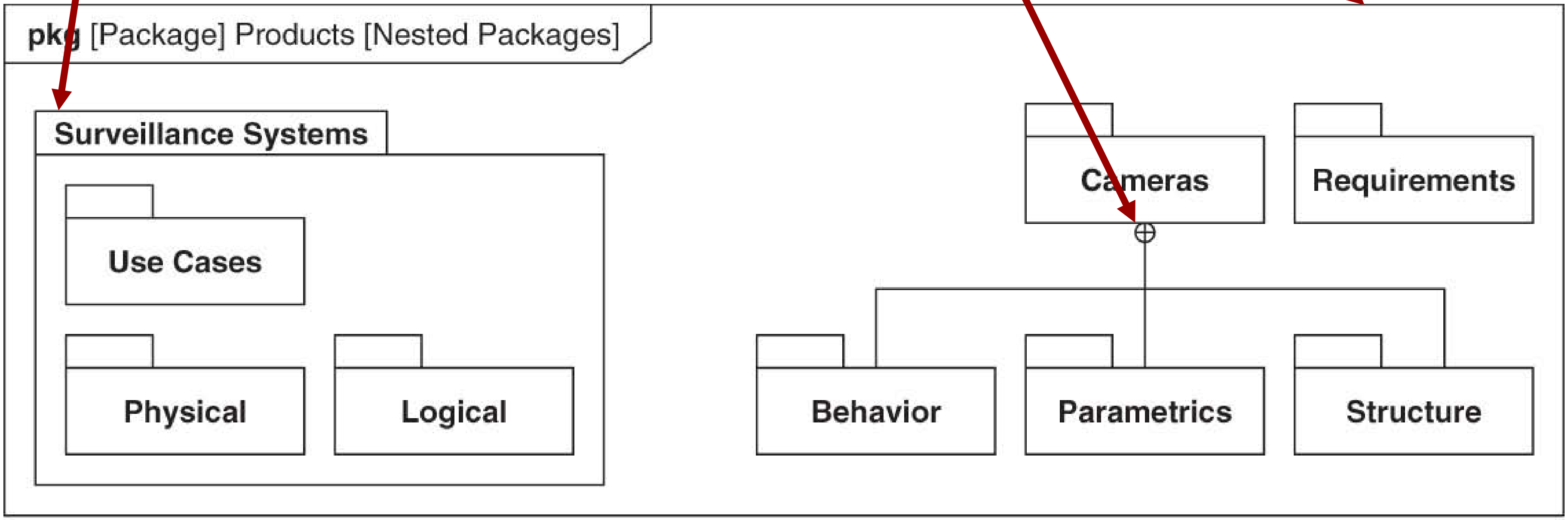
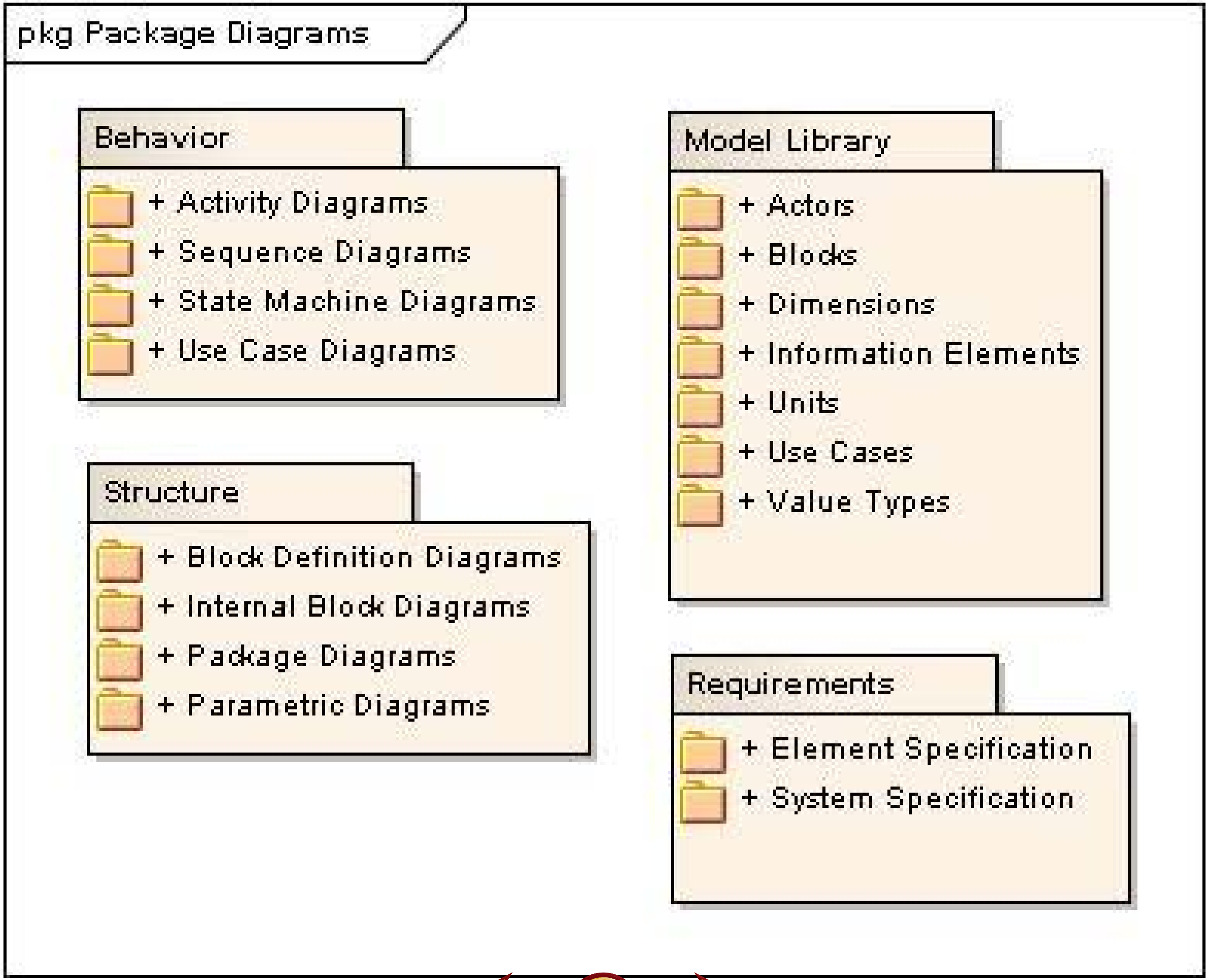


FIGURE 5.1

Package Organization for Parking Garage Gate



Summary

- Packages are used for Model Organization
- Package Diagrams are used to depict how the model is organized
- Packages can contain:
  - Other packages
  - Model elements
- Models may be organized using a variety of methods

CAPTURING TEXT-BASED REQUIREMENTS IN THE MODEL

Requirements

- The «requirement» stereotype represents a text based requirement
  - Includes id and text properties
  - Can add user defined properties such as verification method
  - Can add user defined requirements categories (e.g., functional, interface, performance)
- Requirements hierarchy describes requirements contained in a specification
- Requirements relationships include Containment, DeriveReqt, Satisfy, Verify, Refine, Trace, Copy
  - SysML provides a graphical depiction of these relationships
  - SysML also provides a means to capture rationale for a specific requirement or relationship

Automobile Specification Requirements Diagram

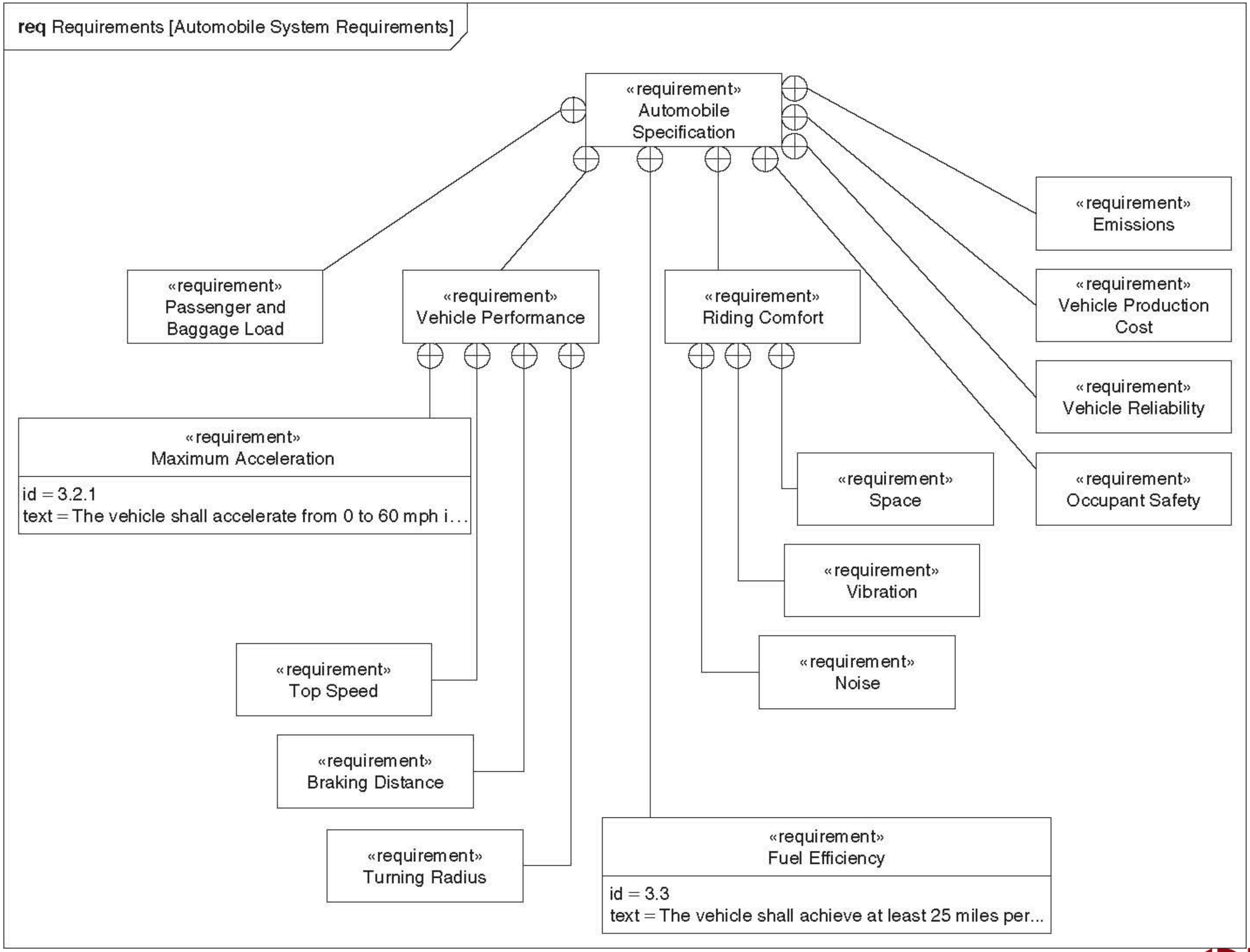


FIGURE 3.2

Requirements Traceability

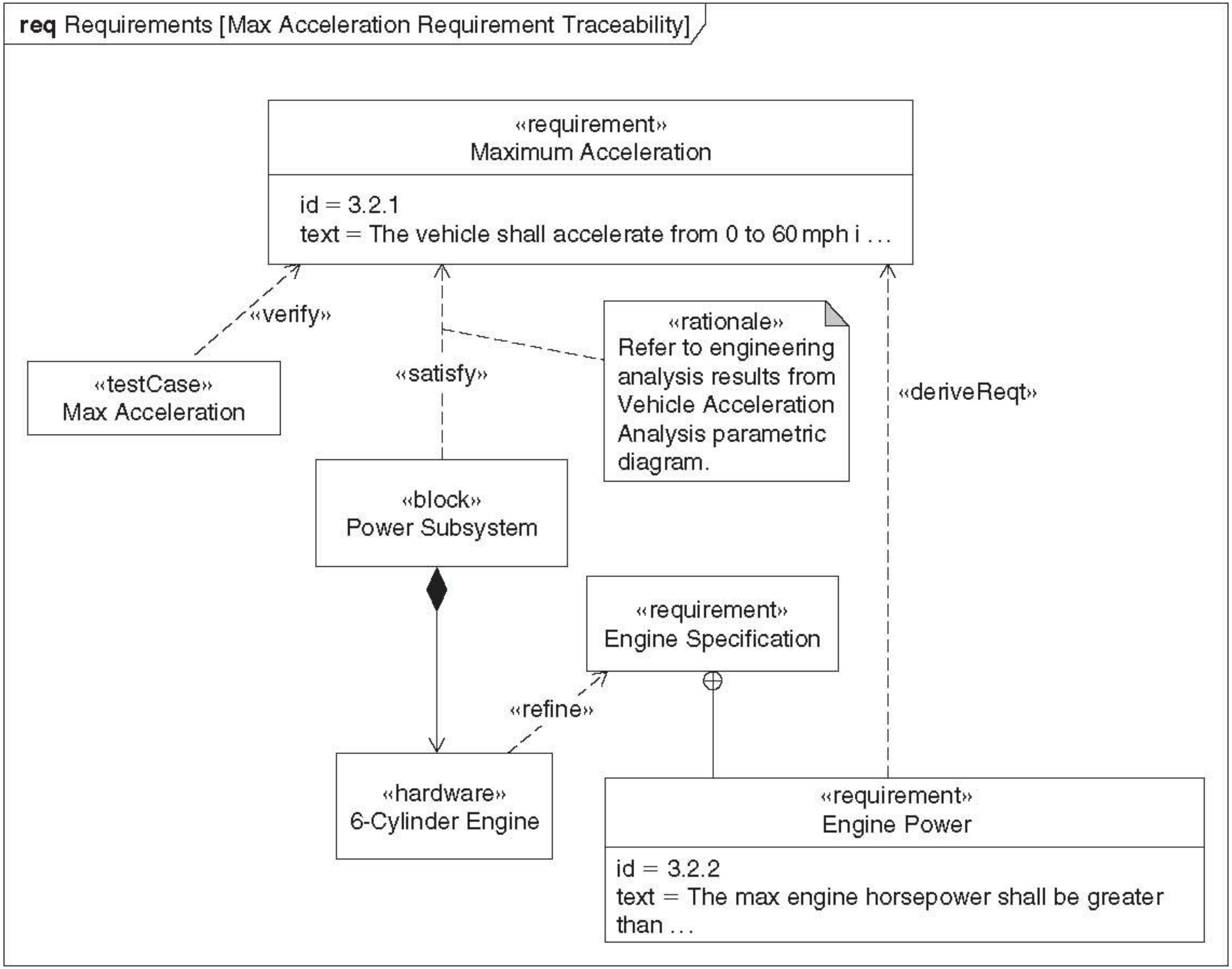


FIGURE 3.18

Representing Relationships

- Three ways to depict requirement relationships in SysML:
  - Direct
  - Compartment
  - Callout

Direct Notation

- Used when the requirement and the related model element appear on the same diagram
- Establishes dependency of model element to requirement in model
- Read figure below as: "The camera satisfies the Sensor Decision requirement".

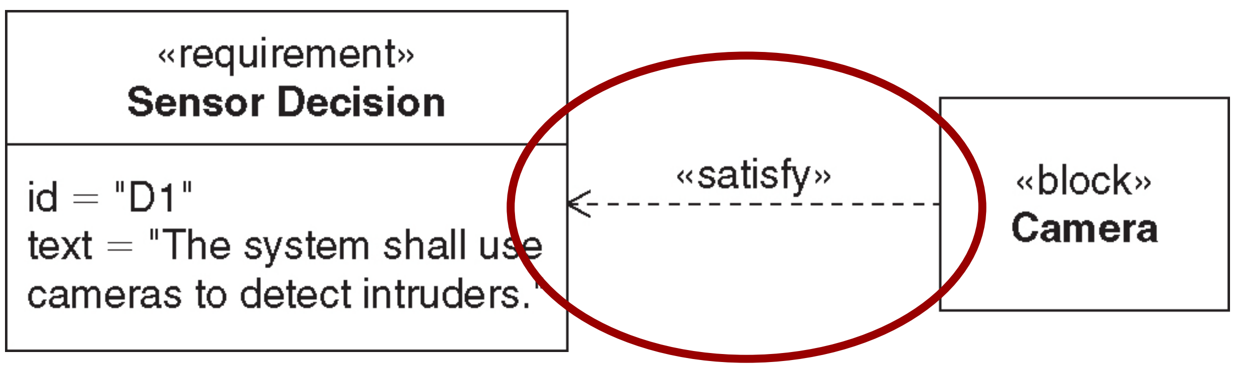


FIGURE 12.3

Compartment Notation

- Used when the requirement and model element do not appear on the same diagram.
- Used for model elements such as blocks or requirements that support compartments.

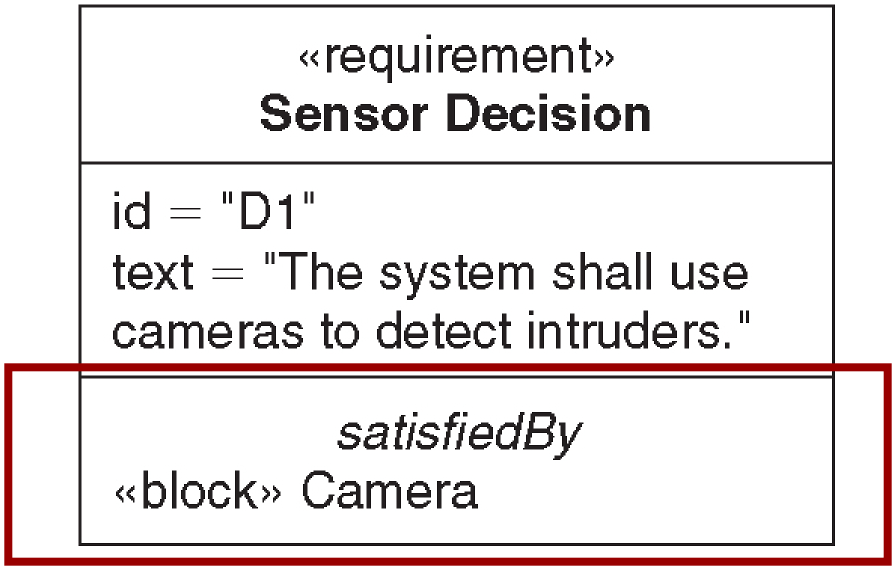
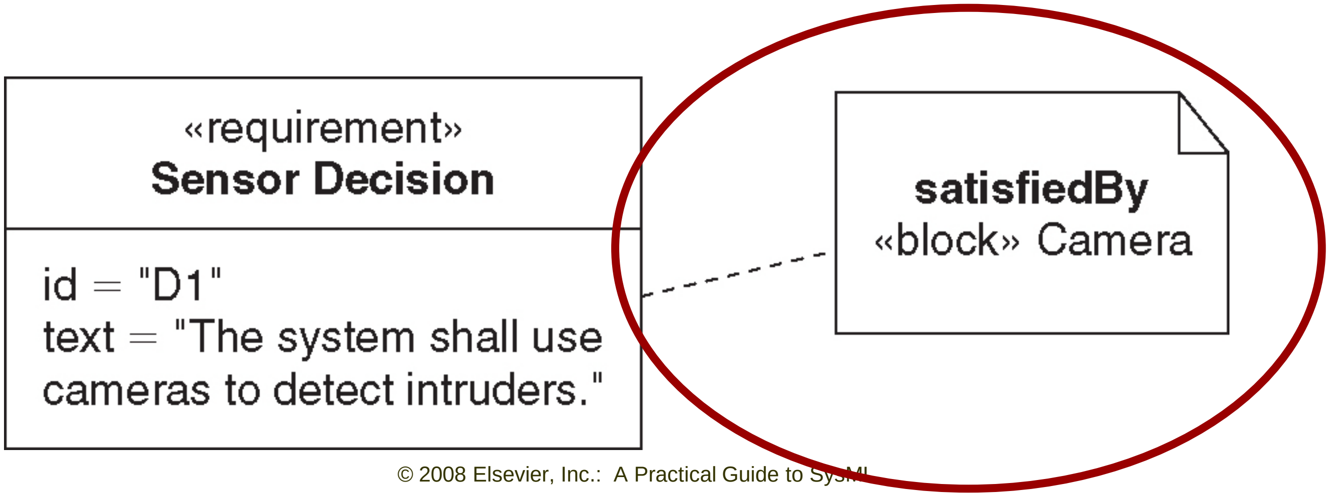


FIGURE 12.4

Callout Notation

- Used when the requirement and model element do not appear on the same diagram
  - Uses 'Note' box, rather than model element
- Can be used when the model element or tool does not support compartments



© 2008 Elsevier, Inc.: A Practical Guide to SysML

FIGURE 12.5

Depicting Rationale

- **Used to explain or justify a requirement or a requirement relationship**

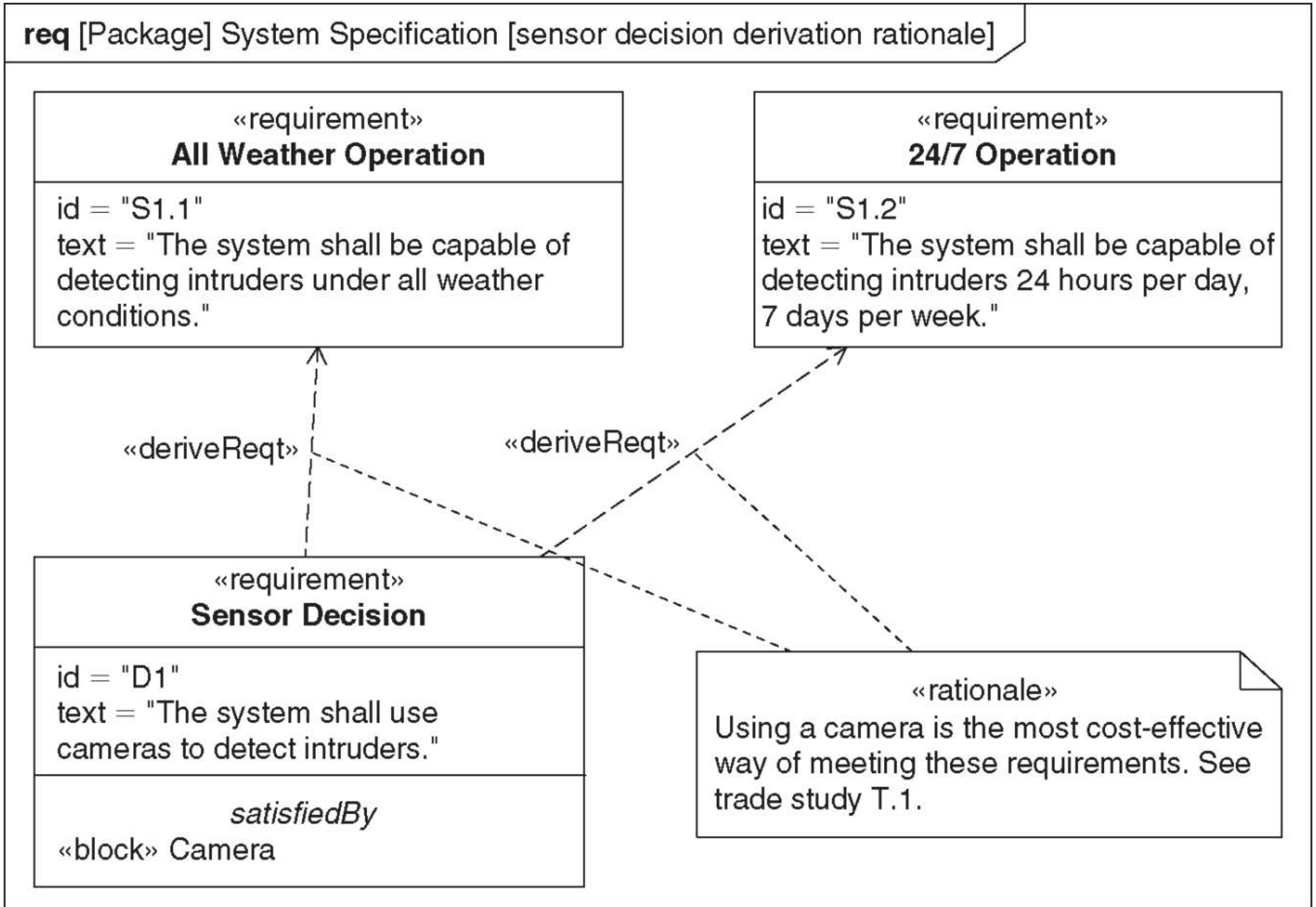
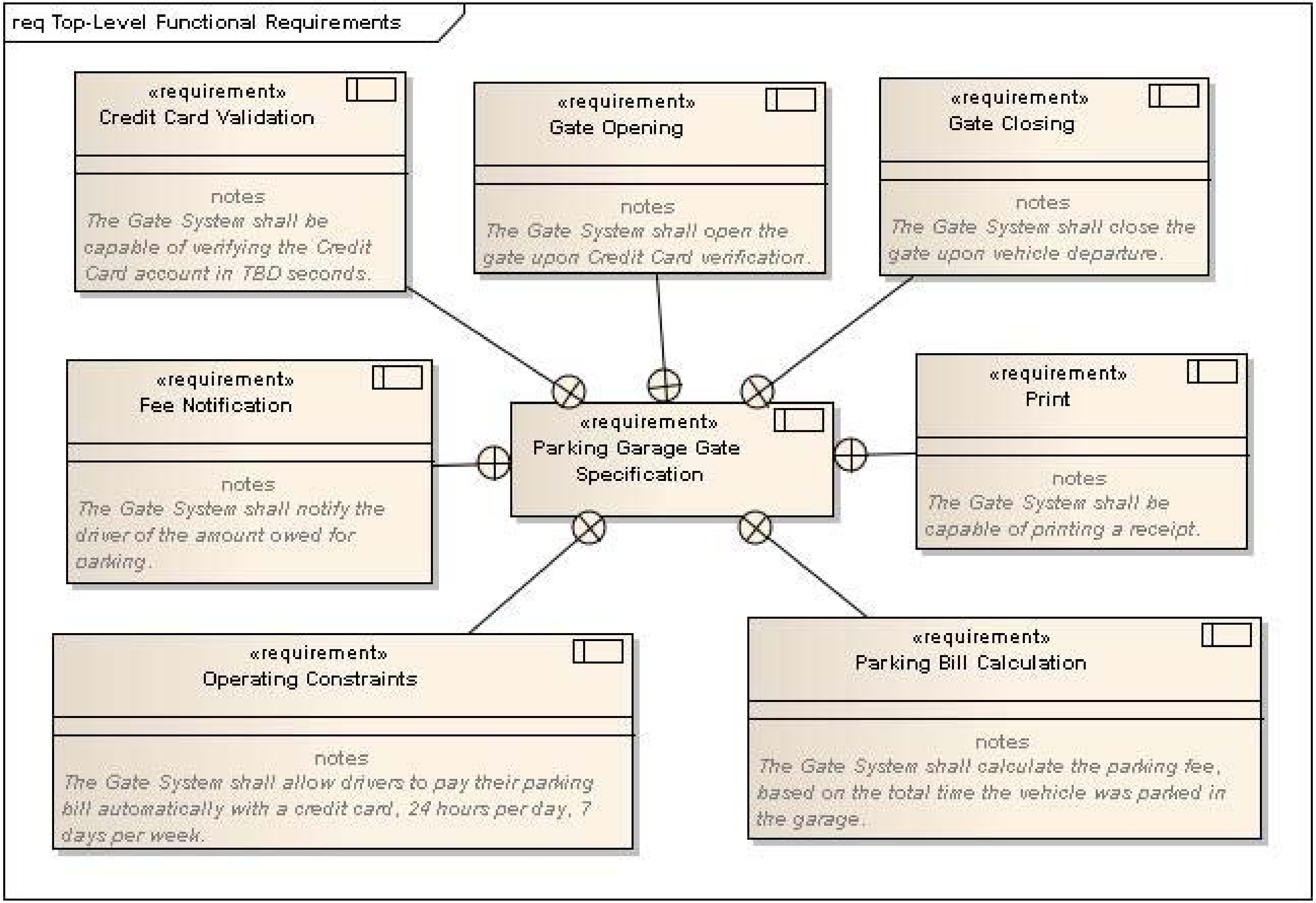


FIGURE 12.14

Parking Garage Requirements Model



Summary

- Requirement modeling graphically depicts:
  - Hierarchy between requirements
  - Traceability between requirements and the rest of the model elements
- There are three types of notation used to depict requirement relationships: Direct, Compartment, and Callout
- There are seven types of requirement relationships in SysML:
  - Containment
  - Satisfy
  - Verify
  - Derive
  - Refine
  - Trace
  - Copy

MODELING HIGH LEVEL FUNCTIONALITY WITH USE CASES

Use Cases

- Provide means for describing basic functionality in terms of usages/goals of the system by actors
  - Use is methodology dependent
  - Often accompanied by use case descriptions
- Common functionality can be factored out via «include» and «extend» relationships
- Elaborated via other behavioral representations to describe detailed scenarios
- No change to UML

Operate Vehicle Use Case Diagram

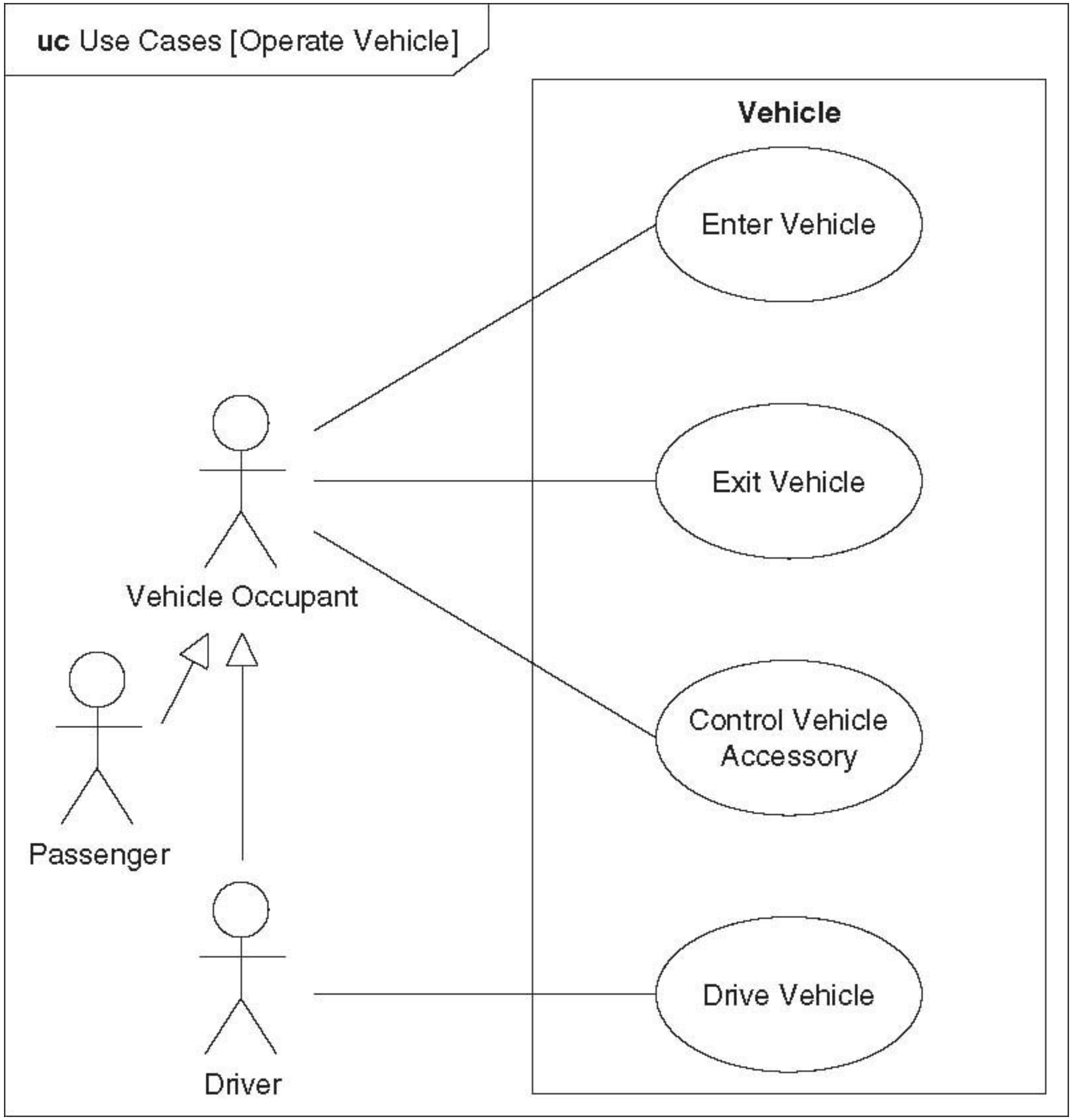


FIGURE 3.4

Use Case Diagram Components

- Use Case diagrams are comprised of the following:
  - Subject
  - Actors
  - Use Cases
  - Relationships

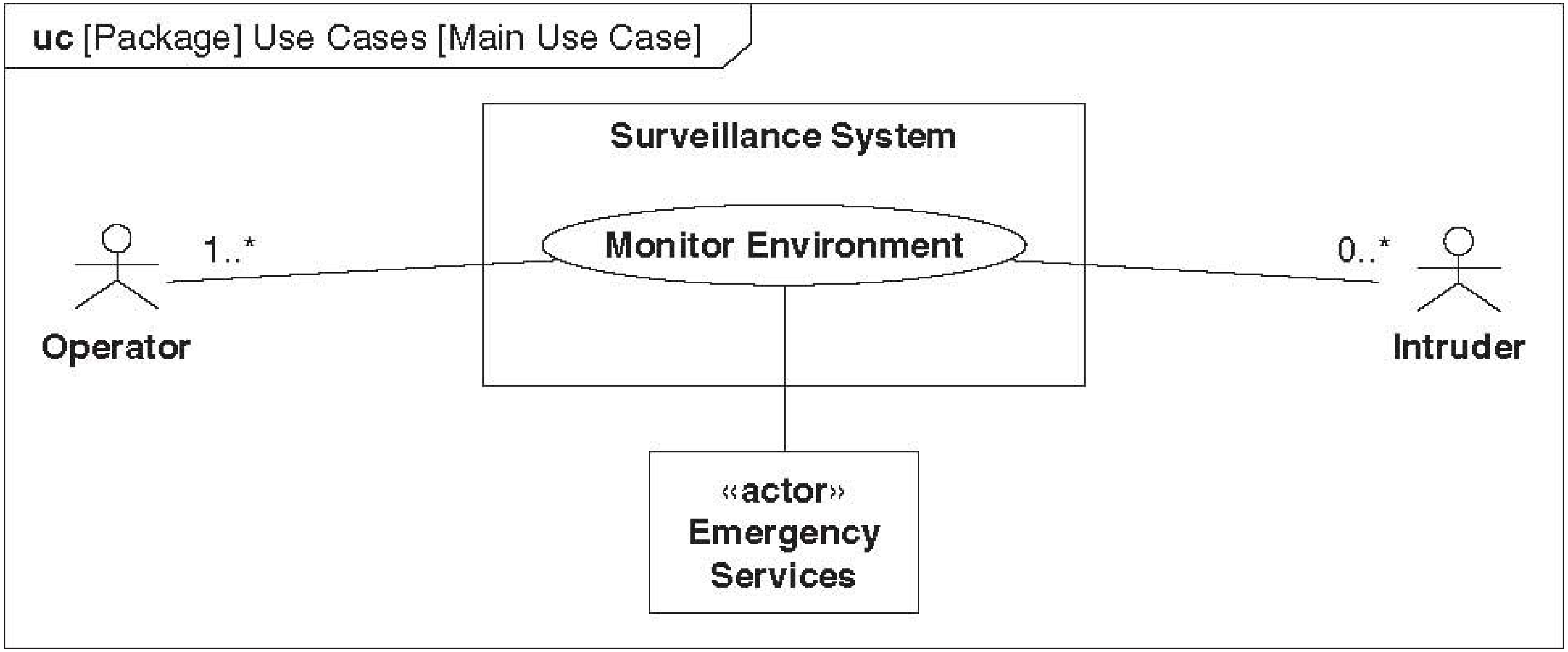


FIGURE 11.1

© 2008 Elsevier, Inc.: A Practical Guide to SysML

Subject

- Provides the functionality in support of the use cases
- Represents a system being developed
- Also called the ‘system under consideration’
- Represented by a rectangle on the use case diagram

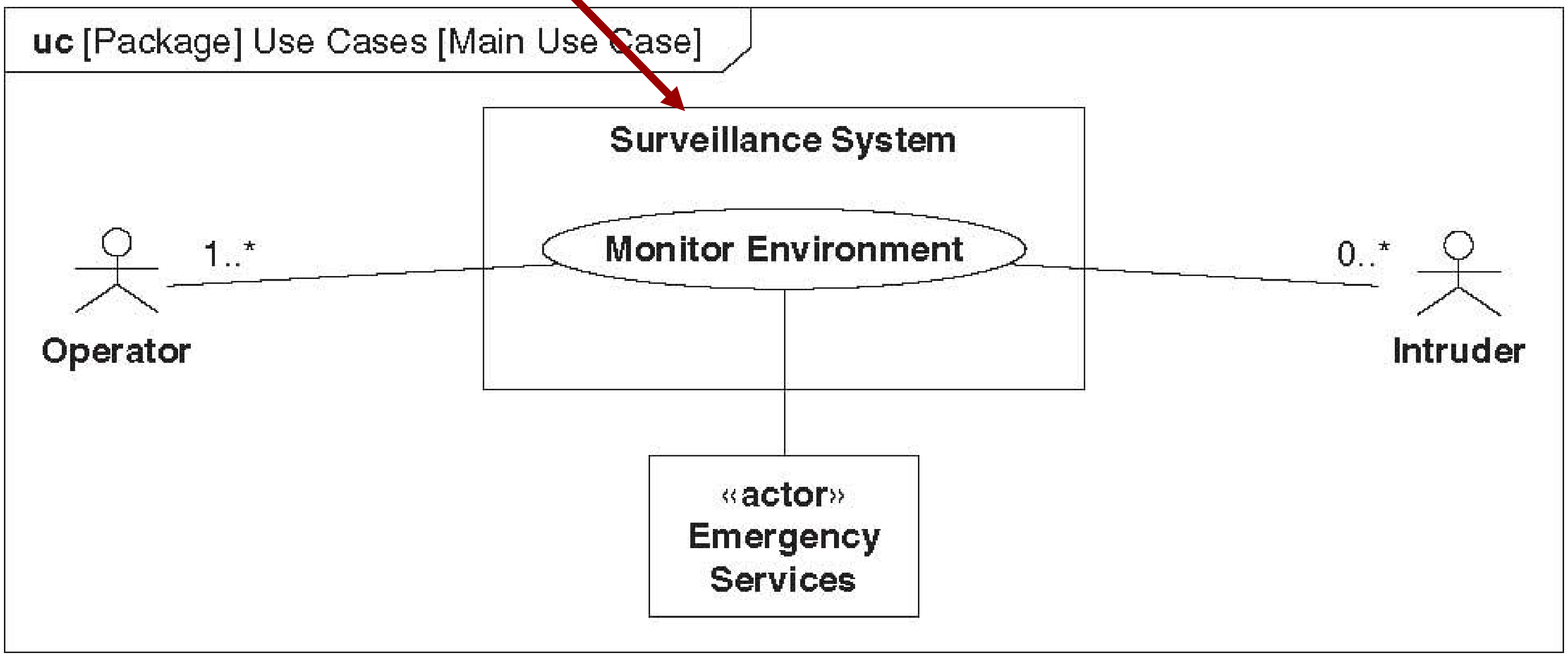


FIGURE 11.1

Actors

- Used to represent something that uses the system
  - Not ‘part’ of the system
    - Depicted outside of the system ‘box’
  - Actors interface with the system
- Can be a person or another system
- Usually depicted by a stick figure and/or block with <<actor>> label
- Name the Actors based on the **role** they perform as a user of the system (e.g. Operator, Customer, etc)

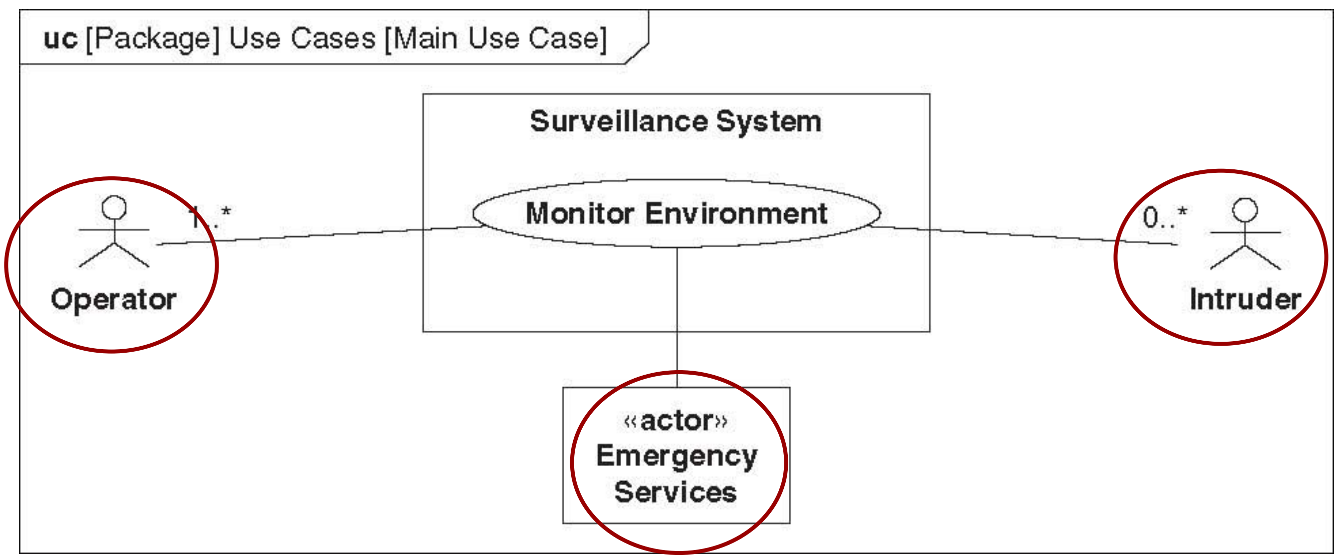


FIGURE 11.1

Use Cases

- Represent the goals that a system will support
- Depicted by an oval with the Use Case name inside
- Name should consist of a verb and a noun that describe the functionality of the system (e.g. Record Grades, Monitor Environment)

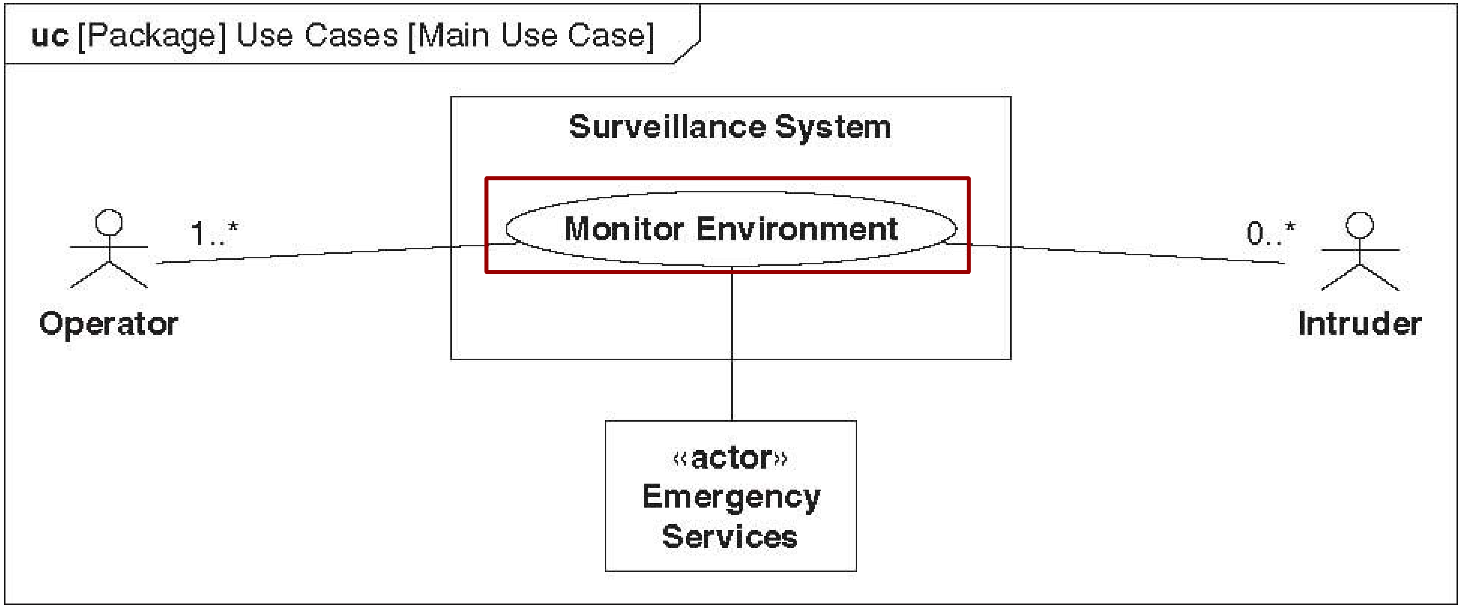


FIGURE 11.1

Relationships on a Use Case Diagram

- Relationships between Actors and Use Cases
- Relationships between Use Cases
  - Include
  - Extend
  - Generalize/Specialize

Relationships Between Use Cases (Include)

- Uses UML ‘dependency’ relationship
- Depicts shared (or re-used) functionality
- The included Use Case is always performed by the base Use Case

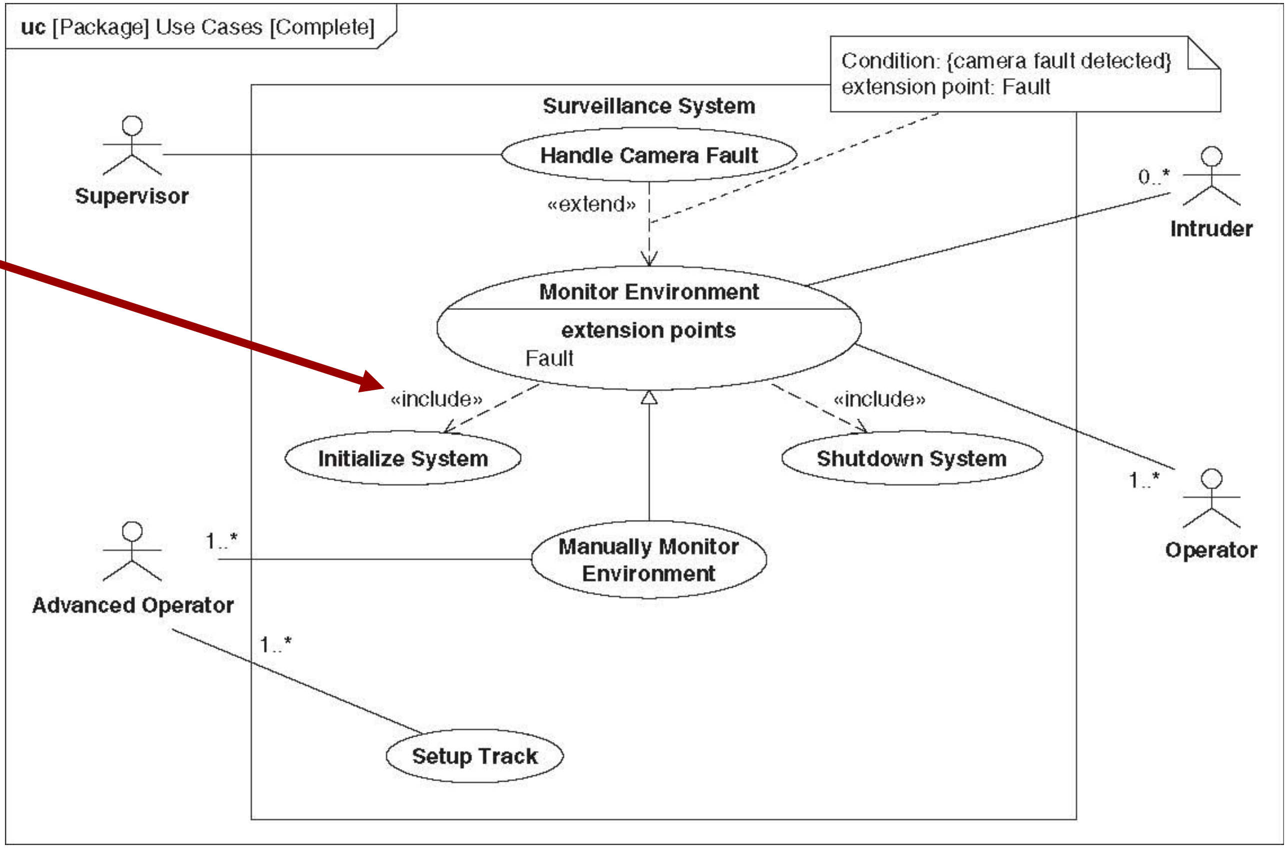
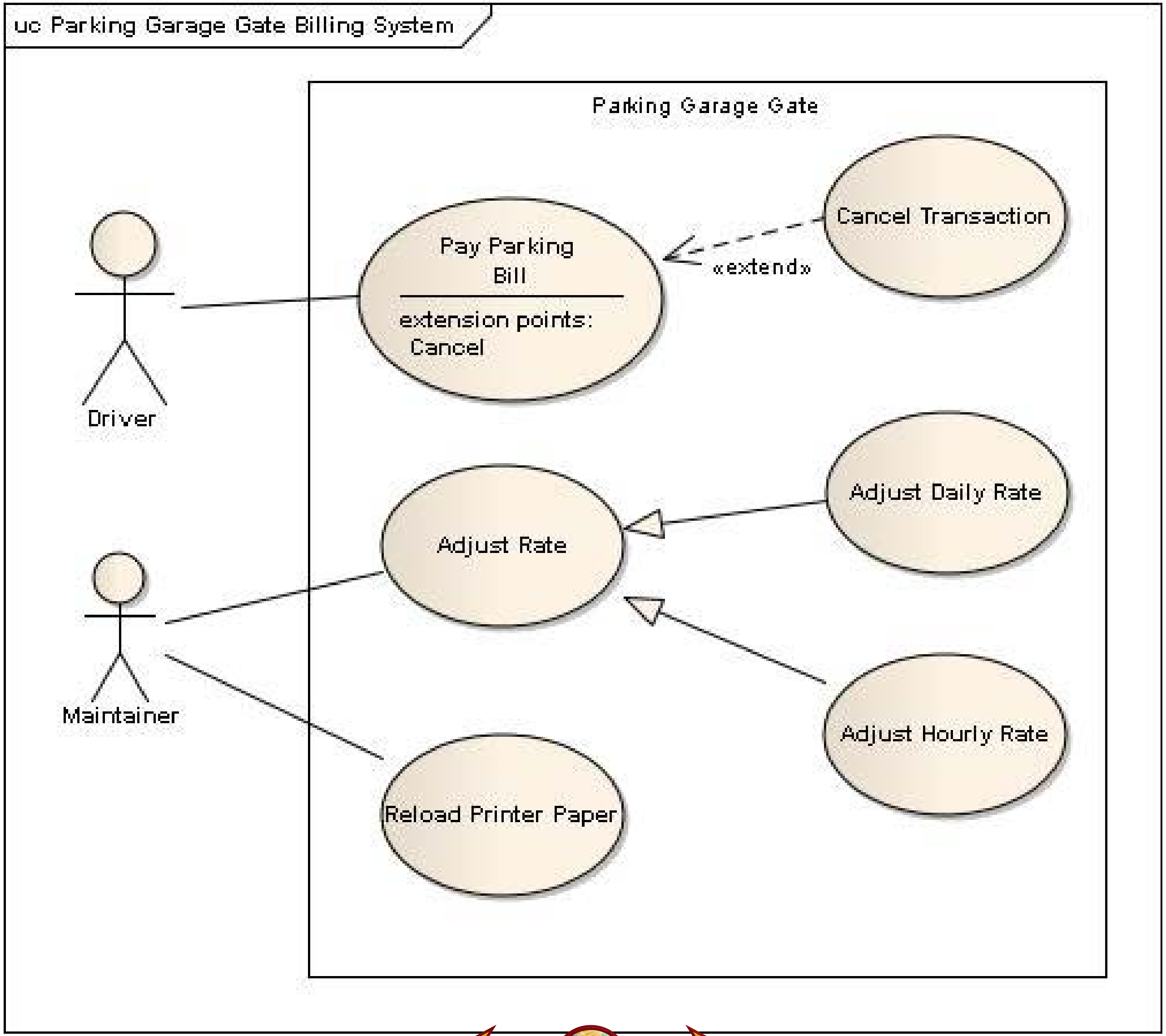


FIGURE 11.4 © 2008 Elsevier, Inc.: A Practical Guide to SysML

Use Case Model for Parking Garage Gate



Summary

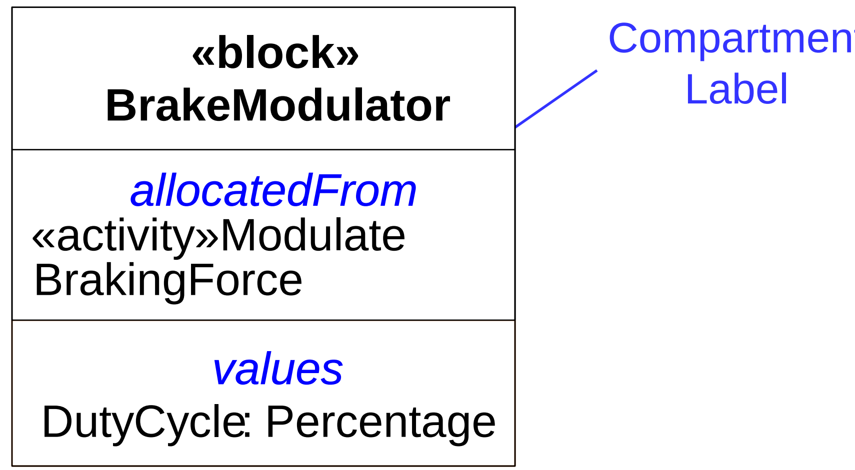
- Use Cases capture the functionality a system must provide to achieve user goals
- Use Case diagrams are made up of:
  - Subject
  - Actors
  - Use Cases
  - Relationships
- Use Case can be elaborated through:
  - Activity diagrams
  - Sequence diagrams
  - State machine diagrams

MODELING STRUCTURE WITH BLOCKS

MODELING BLOCKS AND THEIR RELATIONSHIPS ON A BDD

Blocks are Basic Structural Elements

- **Provides a unifying concept for describing the structure of an entity**
  - **System**
  - **Hardware**
  - **Software**
  - **Data**
  - **Procedure**
  - **Facility**
  - **Person**



- **Multiple standard compartments can describe the block characteristics**
  - **Properties (parts, references, values, ports)**
  - **Operations**
  - **Constraints**
  - **Allocations from/to other model elements (e.g. activities)**
  - **Requirements the block satisfies**
  - **User defined compartments**

Top Level Block Definition Diagram

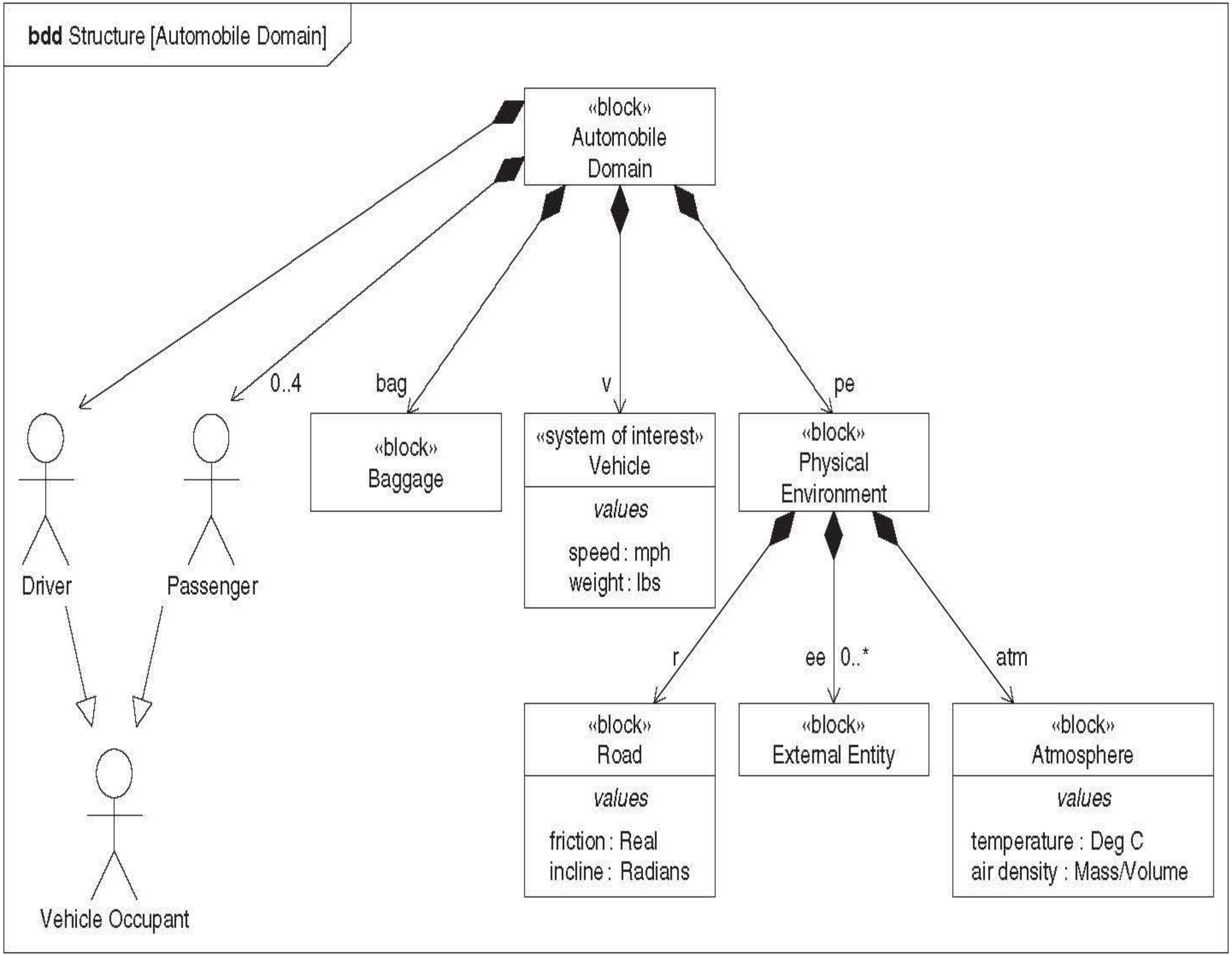


FIGURE 3.3

© 2008 Elsevier, Inc.: A Practical Guide to SysML

Vehicle System Hierarchy

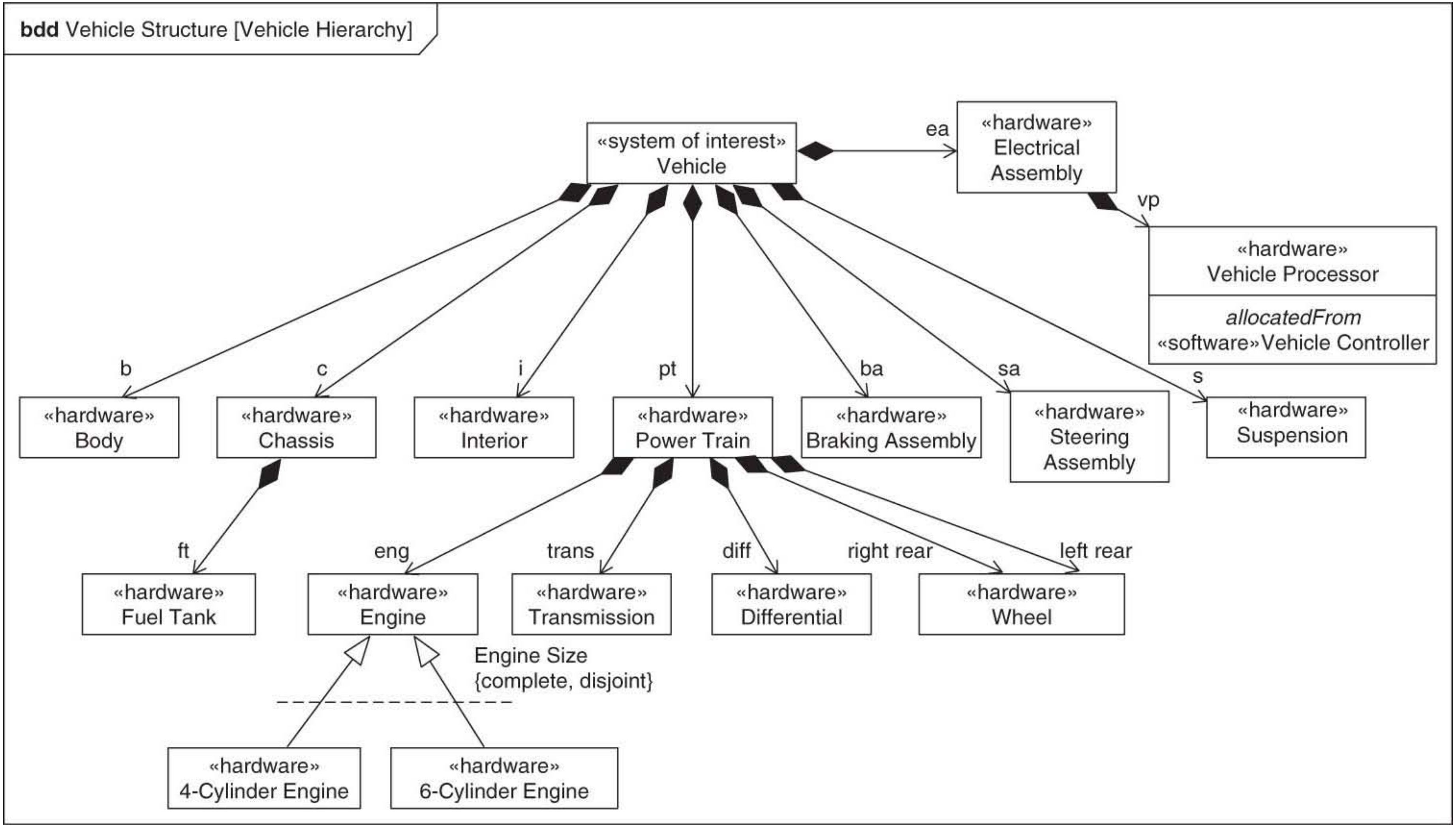


FIGURE 3.10

Purpose of Block Definition Diagrams

- Depicting Relationships between Blocks
  - Part Associations
  - Generalizations
- Depicting Structural Characteristics of Blocks
  - Value Properties
  - Flow Ports
    - Atomic Ports
    - Non-atomic Ports and Flow Specifications
- Depicting Behavioral Characteristics of Blocks
    - Operations
    - Receptions

Block Composition

- Part Associations depict parts that make up the Whole
  - Black diamond on the Whole end
  - Role names can appear on the part end

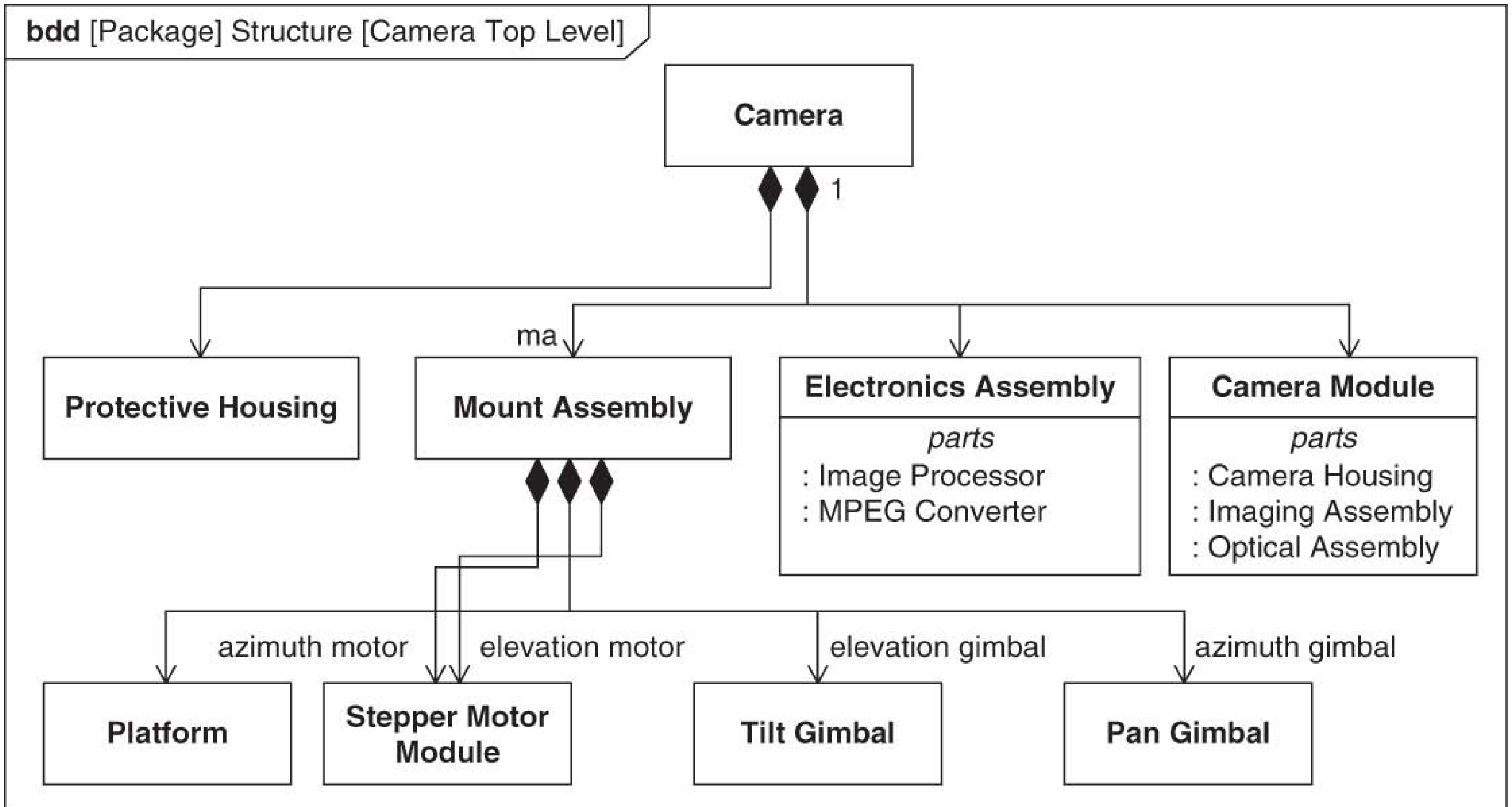
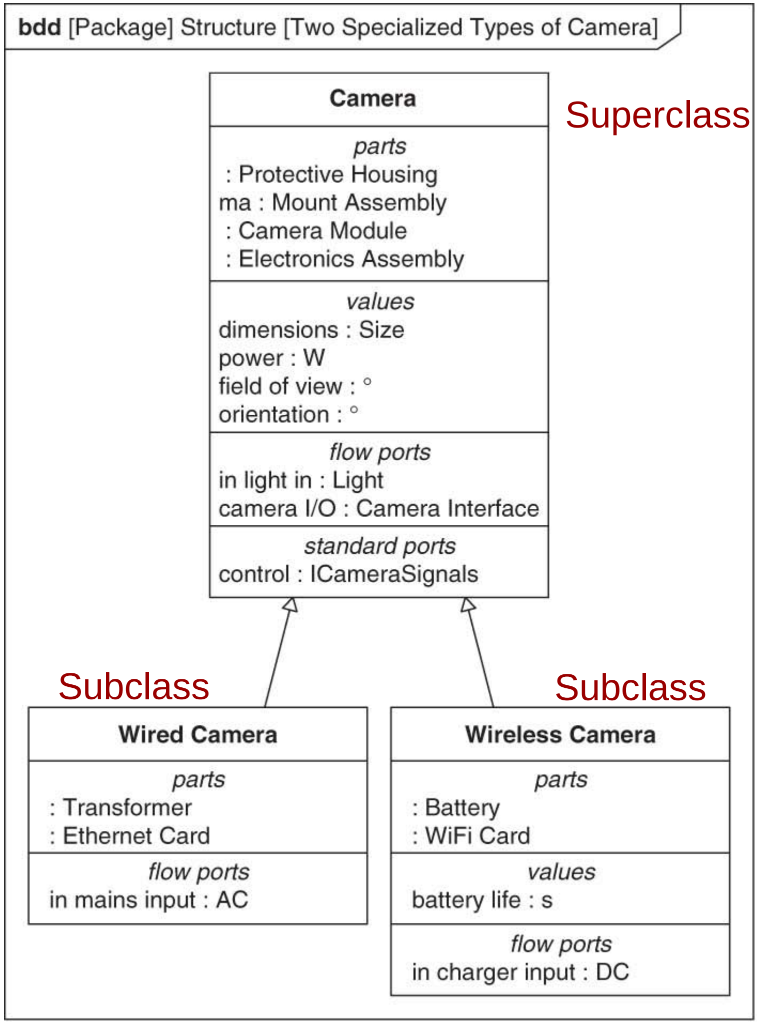


FIGURE 6.5

Generalizations

- Block Definition Diagrams can be used to depict generalization and specialization relationships
- Facilitates reuse
  - The specialized block (subclass) reuses (inherits) the features of a generalized block (superclass), and adds its own features
- Depicts an ‘is-a’ relationship
- Depicted with a closed arrowhead pointing toward the generalized block

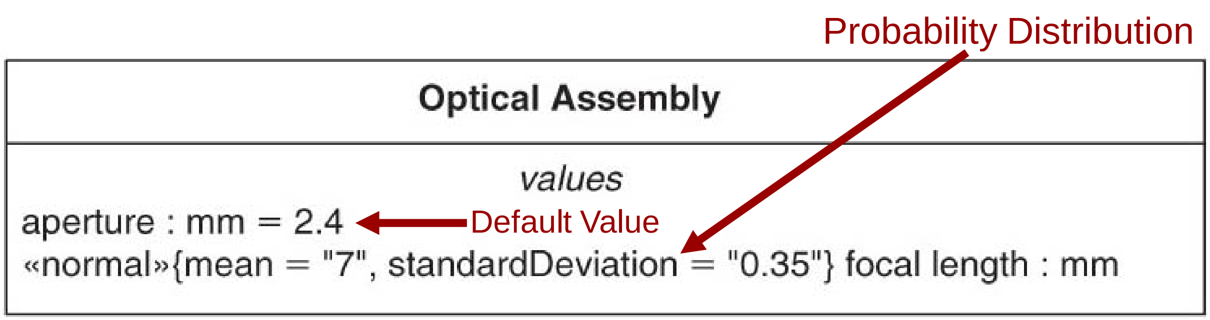


© 2008 Elsevier, Inc.: A Practical Guide to SysML

FIGURE 6.35

Value Properties

- Used to model quantifiable block characteristics or attributes
- Based on a Value Type, which describe the values for quantities
- Listed in compartments using the following syntax:
  - value property name: value type name
- Value Properties:
  - can have default values
  - can also define a probability distribution for their values



© 2008 Elsevier, Inc.: A Practical Guide to SysML

FIGURE 6.22

Flow Ports

- **Flow Ports – used to describe an interaction point for items flowing in or out of a block**
- **Two types:**
  - **Atomic Ports**
  - **Non-atomic Ports**
- **Can be depicted as a box on the block border or in a block compartment**

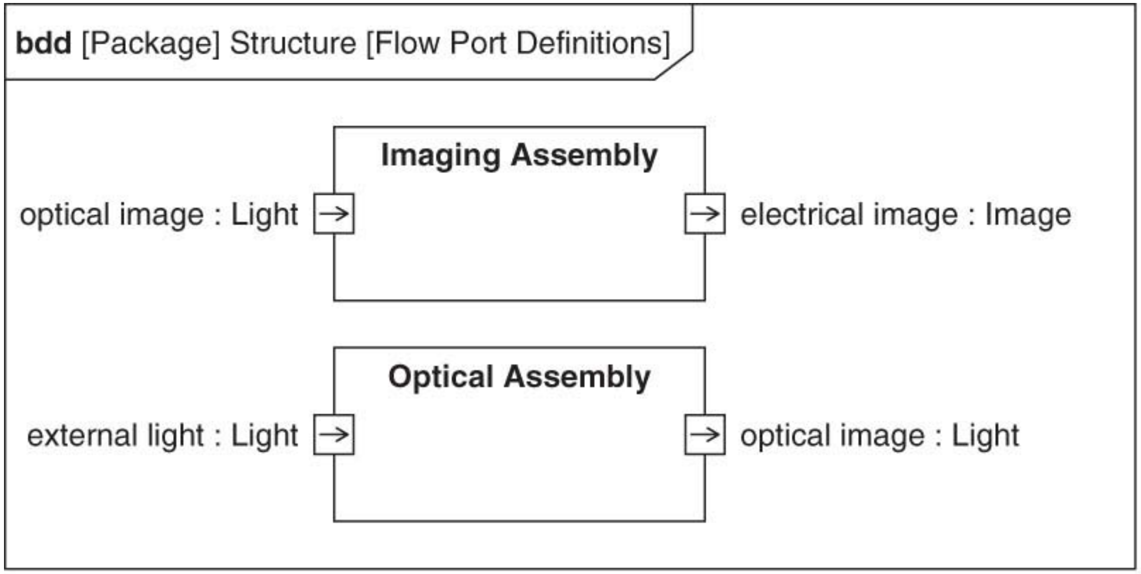


FIGURE 6.25

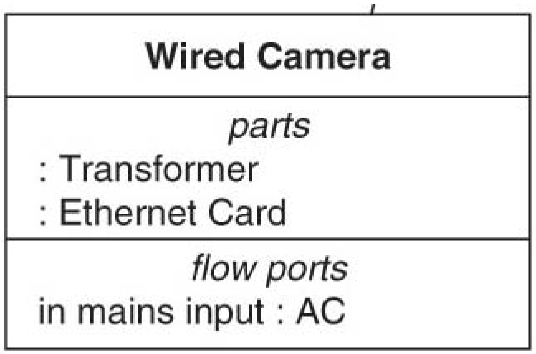


FIGURE 6.35

Non-Atomic Ports and Flow Specifications

- **Non-Atomic Ports - Used when multiple items flow in and/or out of a block to another block**
  - **Depicted with <> inside the flow port**
- **Flow specification - lists the flow properties, that define the items flowing and their directions, listed as follows:**
  - **direction property name: item name**
- **Conjugate port - indicates that the direction of all flow properties is reversed**

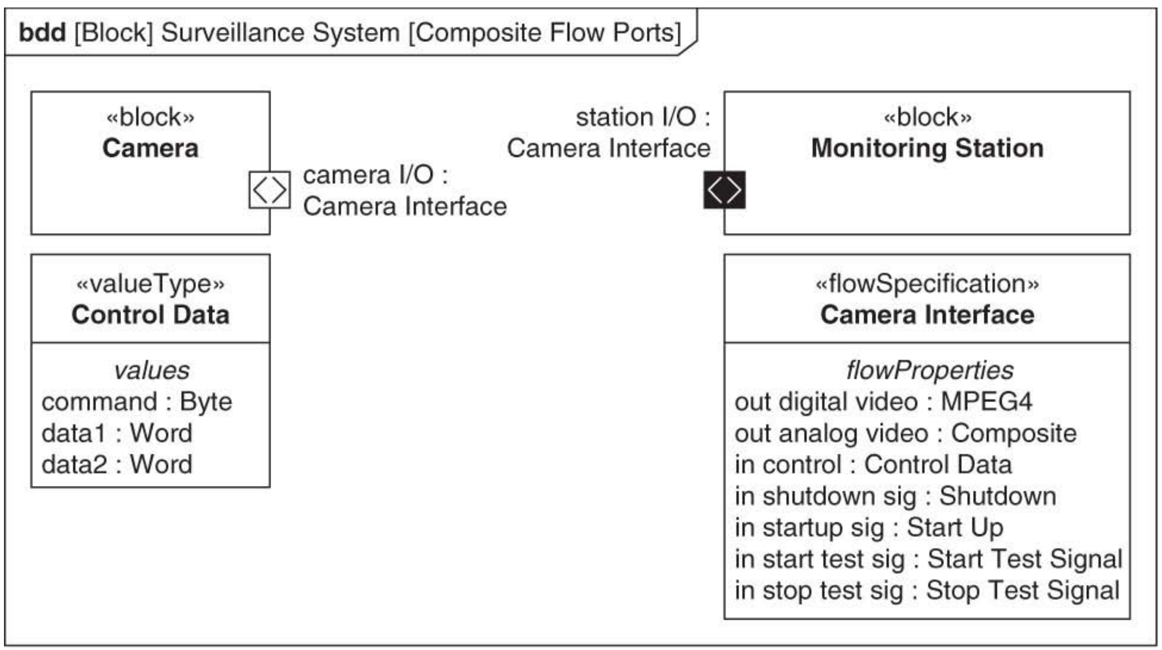


FIGURE 6.26

Operations

- **Operations describe something that a block can do**
- **Operations can have parameters that are passed into or out of the operation**
- **Operations are typically synchronous, (i.e. requestor waits for a response)**
- **Operations are listed in the 'operations' compartment of a block, as follows:**
  - **operation name (parameter list): return type**

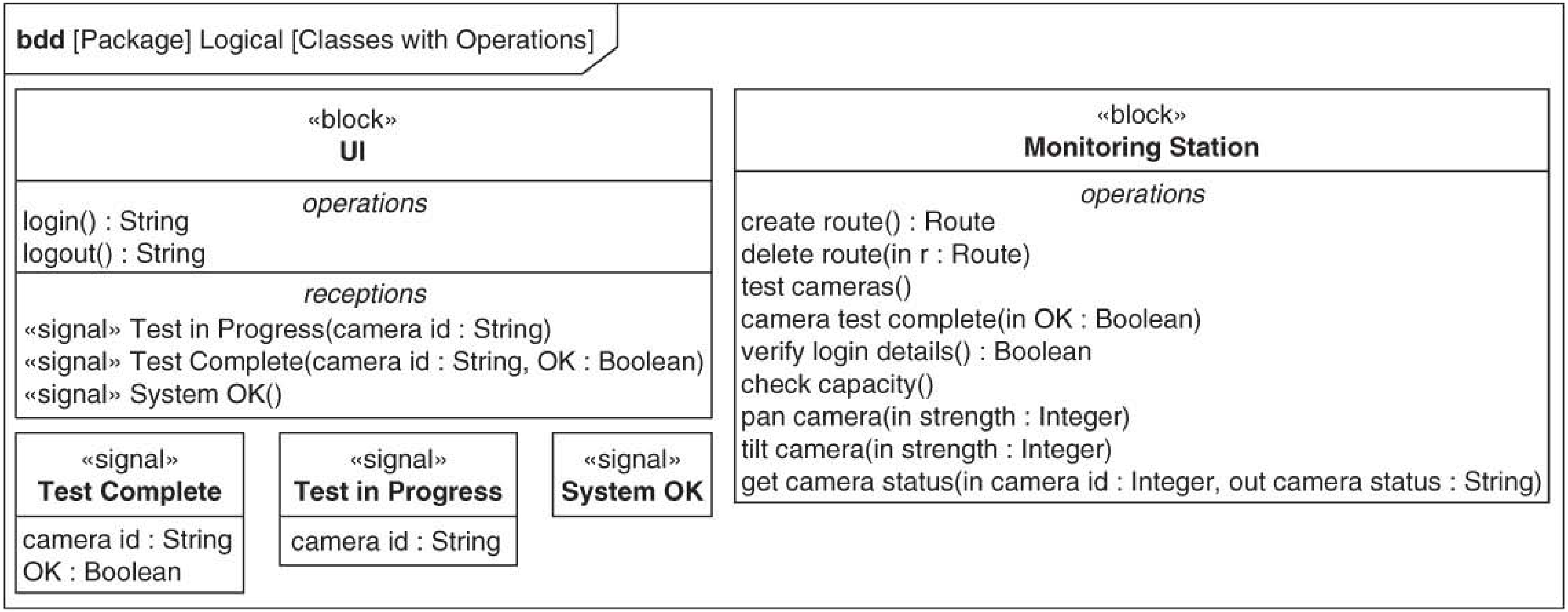


FIGURE 6.31

Interfaces (and Standard Ports)

- Standard Ports – depict interfaces that specify the behavioral features (services) that a block either provides or requires
- Interface symbols have operation and reception compartments like block symbols
- Provided Interface – specifies operations that a block provides
  - Depicted by a ‘ball’ or a realization dependency
- Required Interface – specifies operations required by the block
  - Depicted by a ‘socket’ or a uses dependency (not shown)

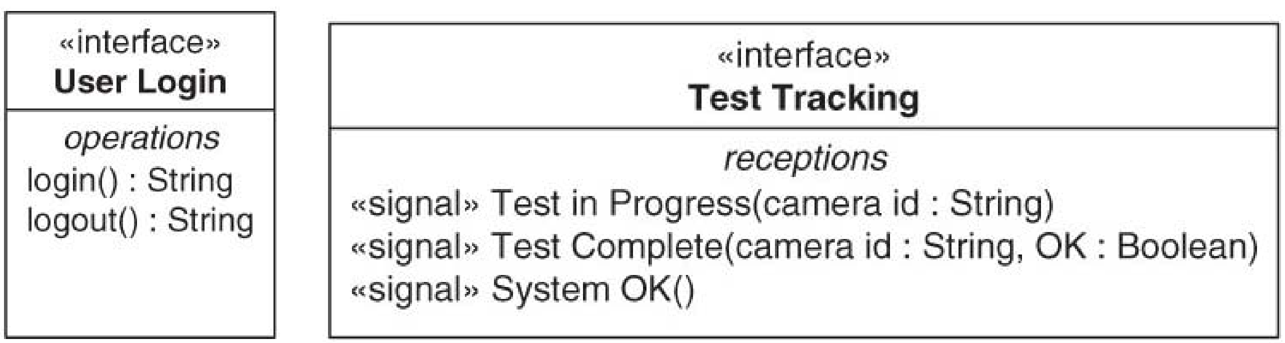


FIGURE 6.32

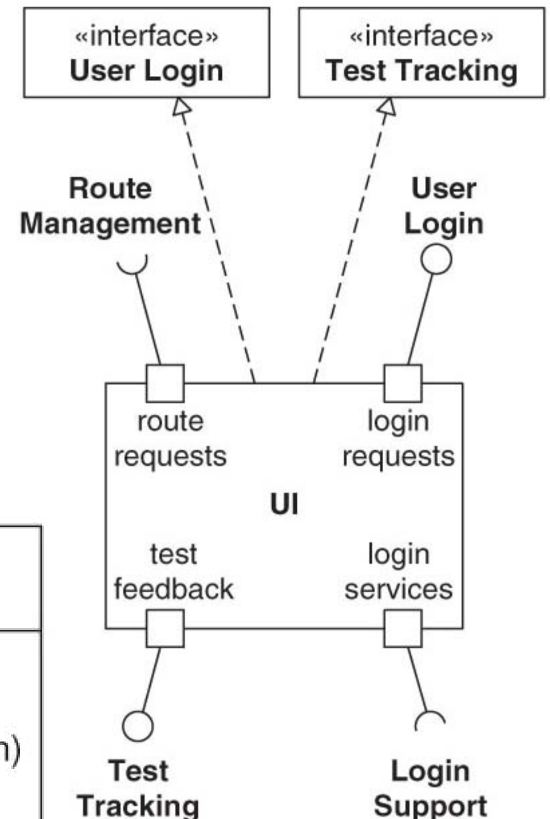
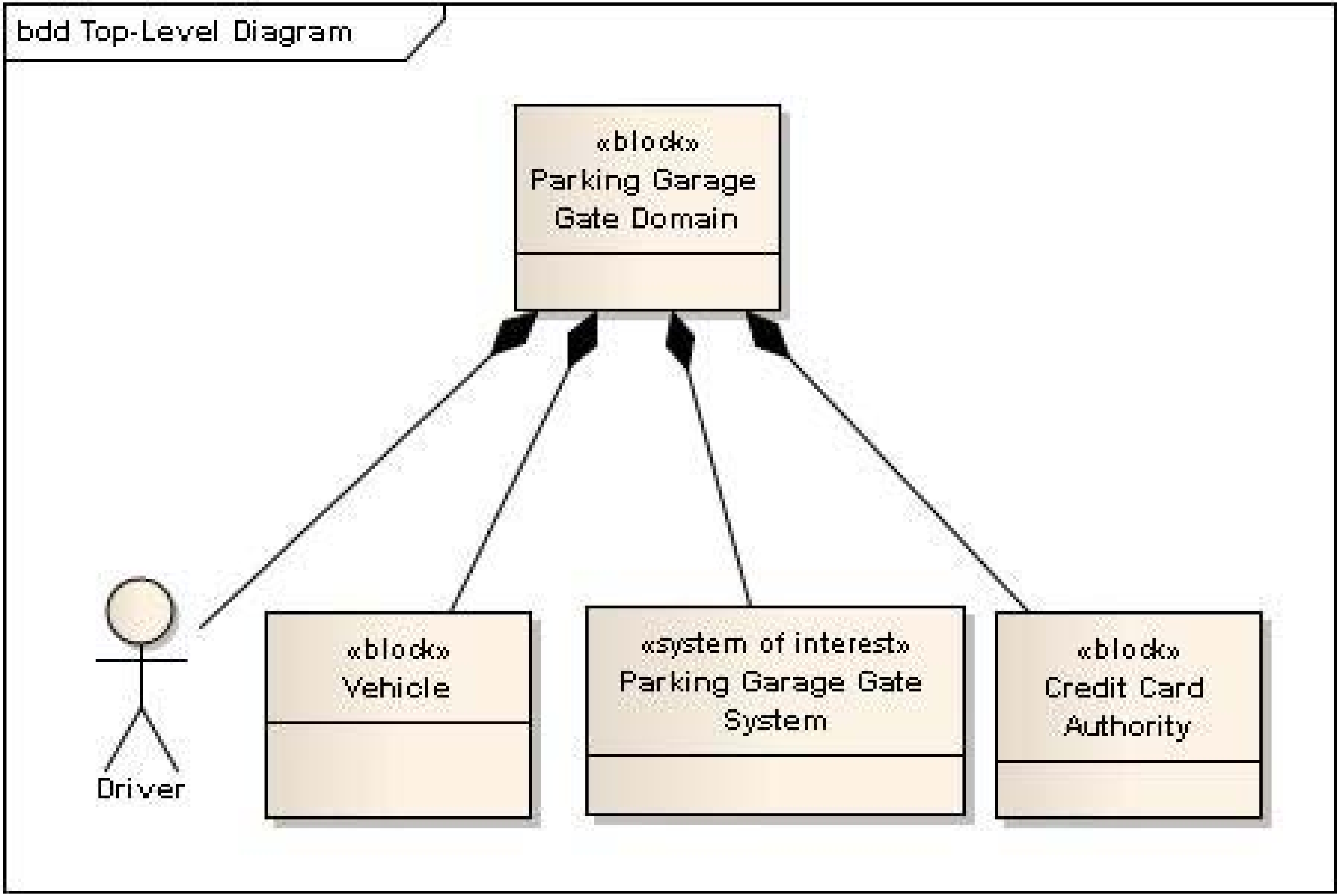
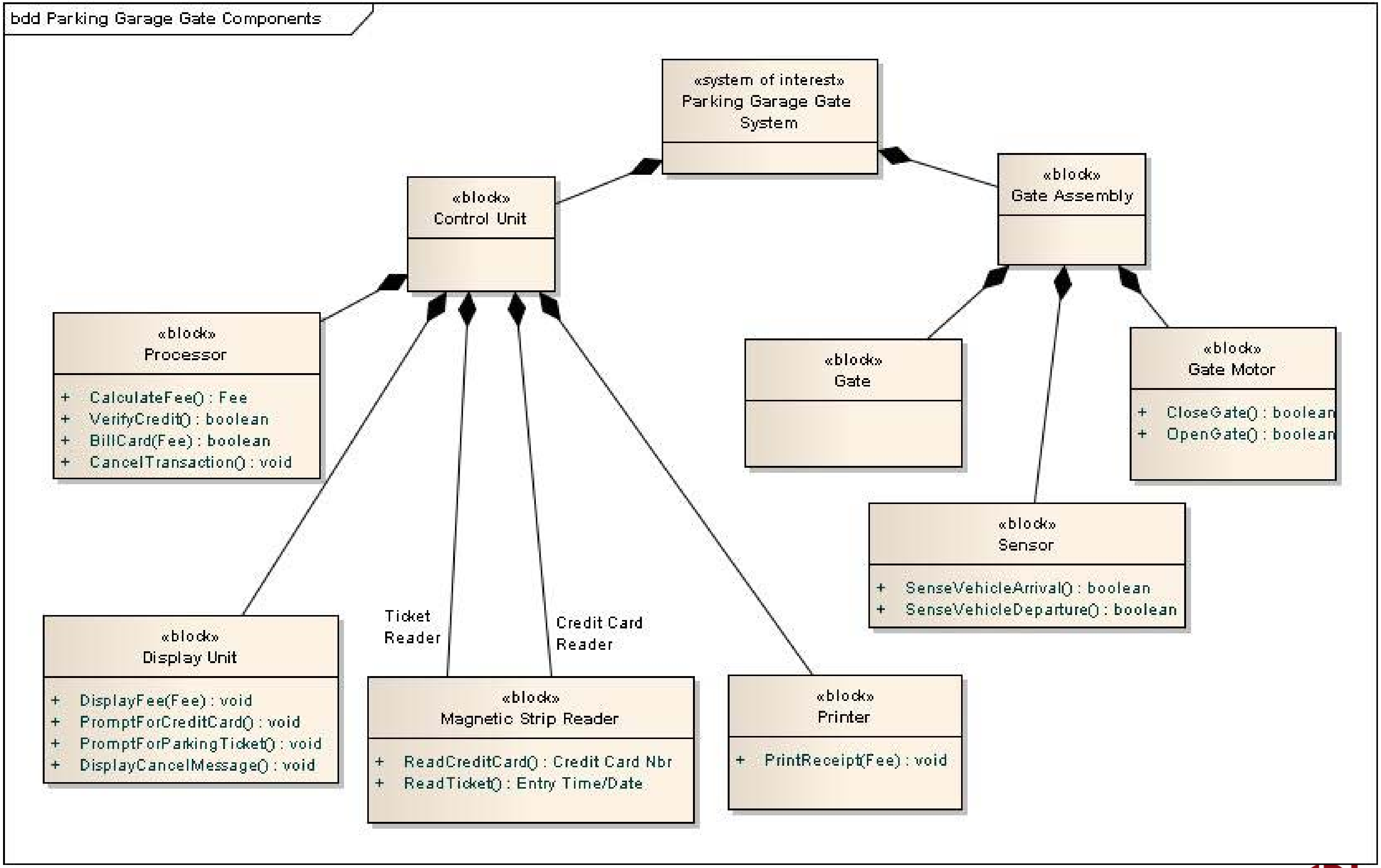


FIGURE 6.33

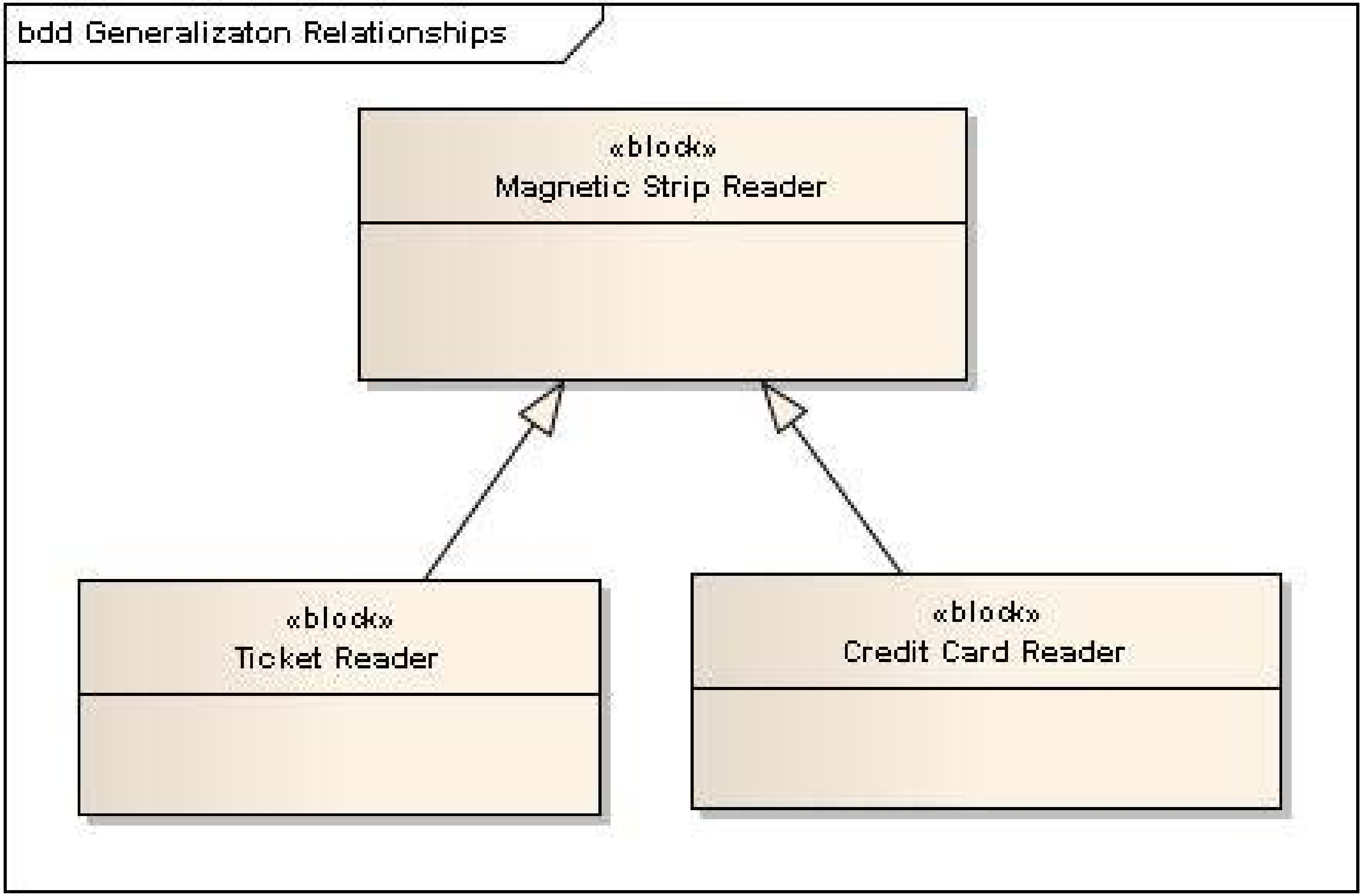
Top Level Block Definition Diagram for Parking Garage Gate Domain



Block Definition Diagram for Gate System



Generalization/Specialization Relationship



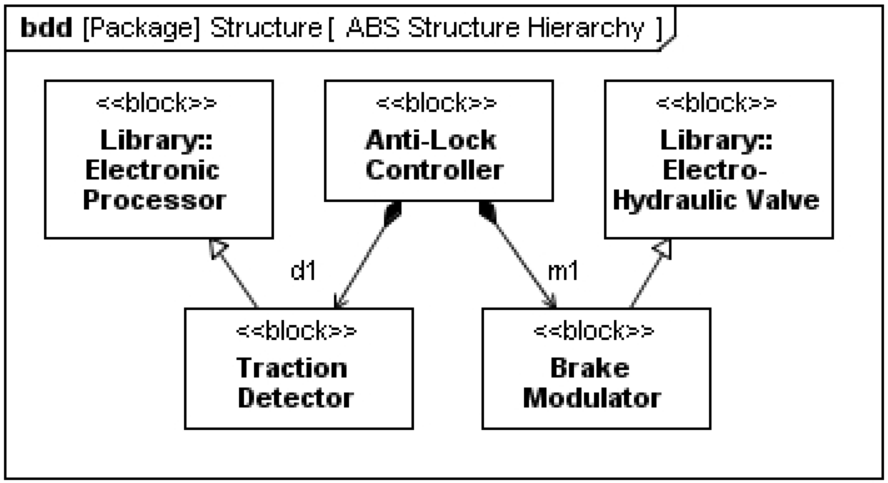
Summary

- A Block is the basic structural element used to model the system's structure
- Block Definition Diagrams are used to depict
  - Definition of blocks
  - How blocks relate to each other
- Block structural characteristics include part properties, value properties, and ports
- Block functional characteristics include operations and receptions
- Block relationships include associations and generalizations

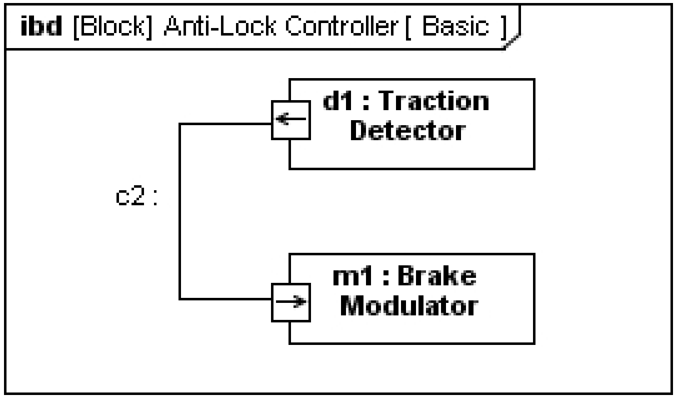
MODELING PART INTERCONNECTION ON AN IBD

Block Definition vs. Usage

Block Definition Diagram



Internal Block Diagram



Definition

- Block is a definition/type
- Captures properties, etc.
- Reused in multiple contexts

Usage

– Part is the usage of a block in the context of a composing block

– Also known as a role

Vehicle System Context Showing External Interfaces

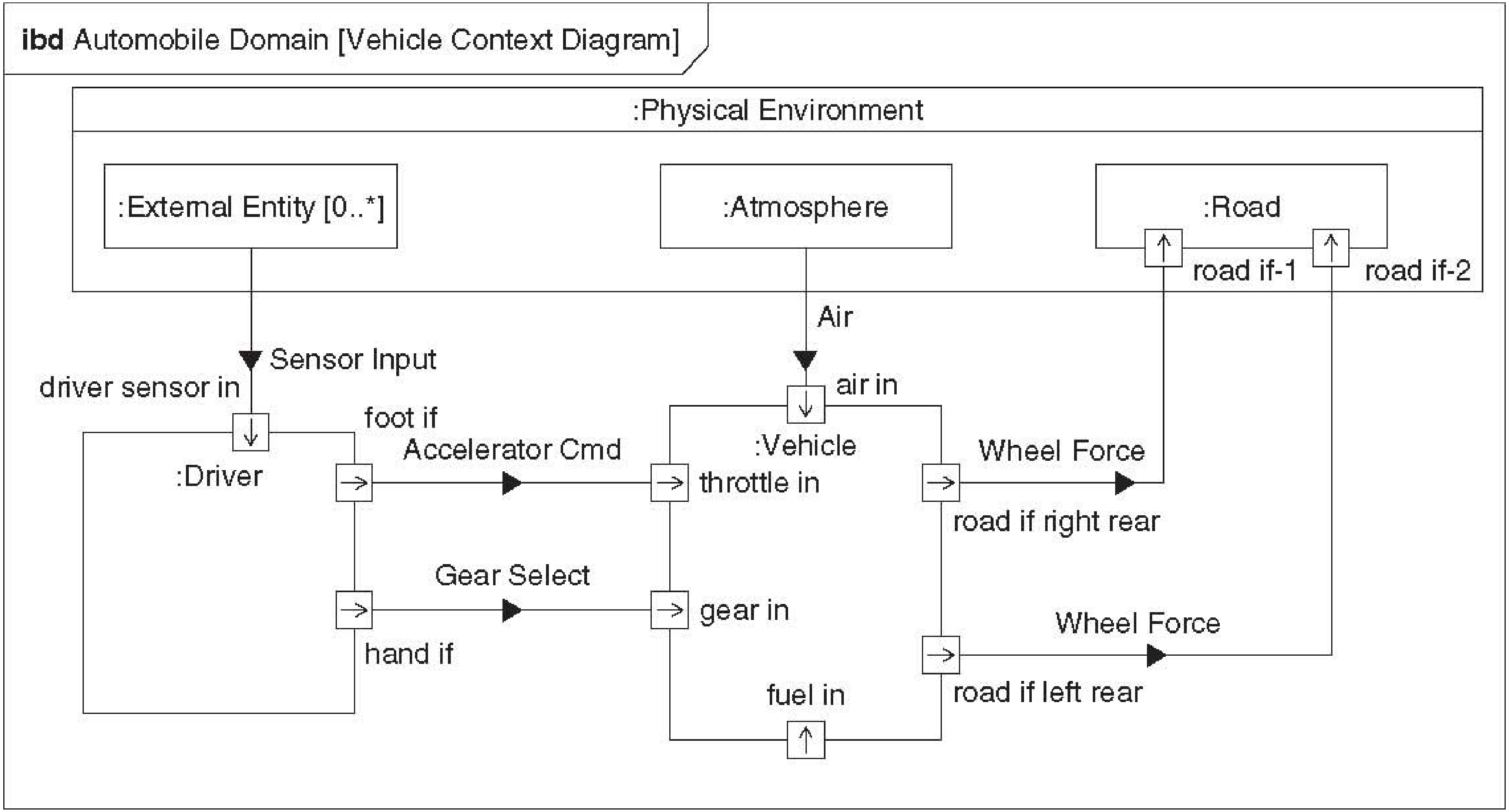


FIGURE 3.9

Power Subsystem Internal Block Diagram

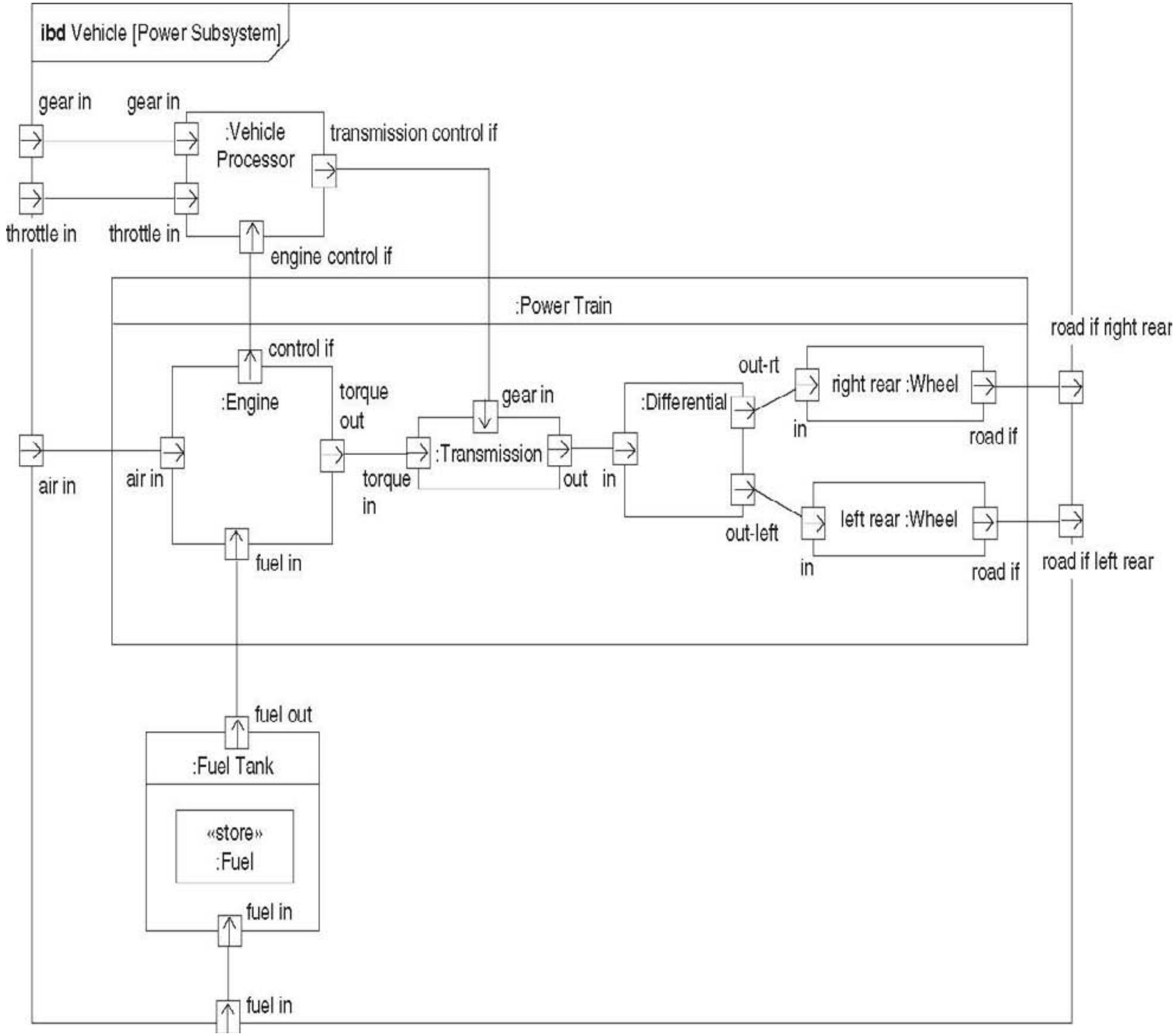
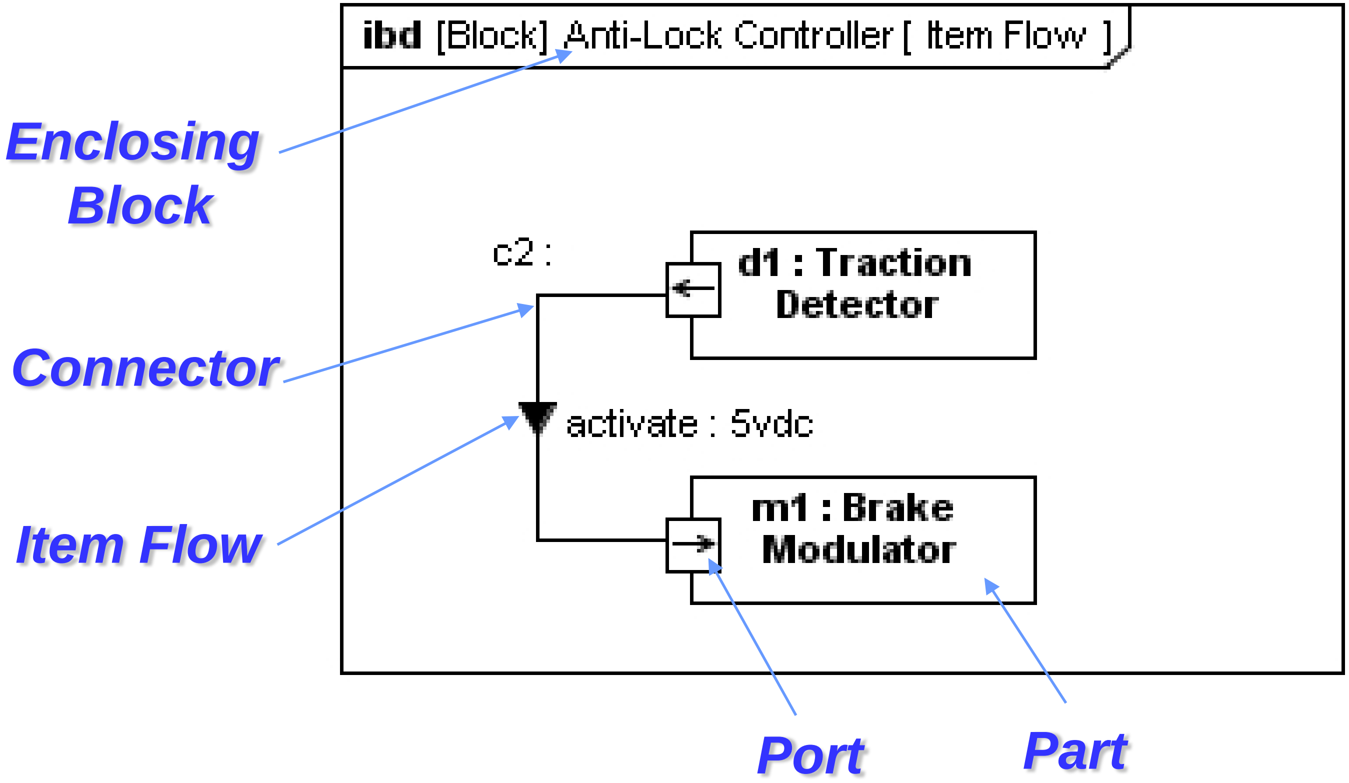


FIGURE 3.12

Internal Block Diagram (ibd)

Blocks, Parts, Ports, Connectors & Flows



Internal Block Diagram Specifies Interconnection of Parts

Modeling Standard Ports and their Connectors on an IBD

- **Standard ports specify interactions as services**
  - **Required interface specifies requests for services (socket symbol)**
  - **Provided interface specifies provided services (ball symbol)**

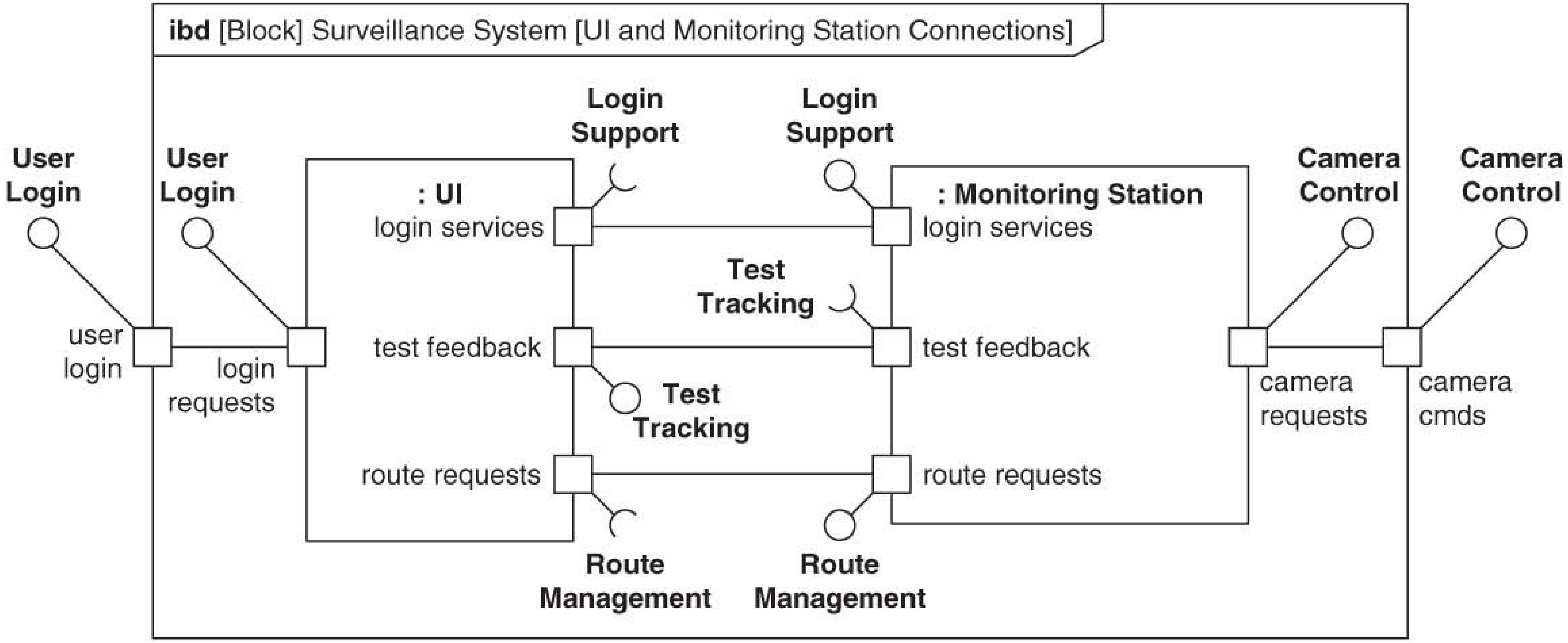
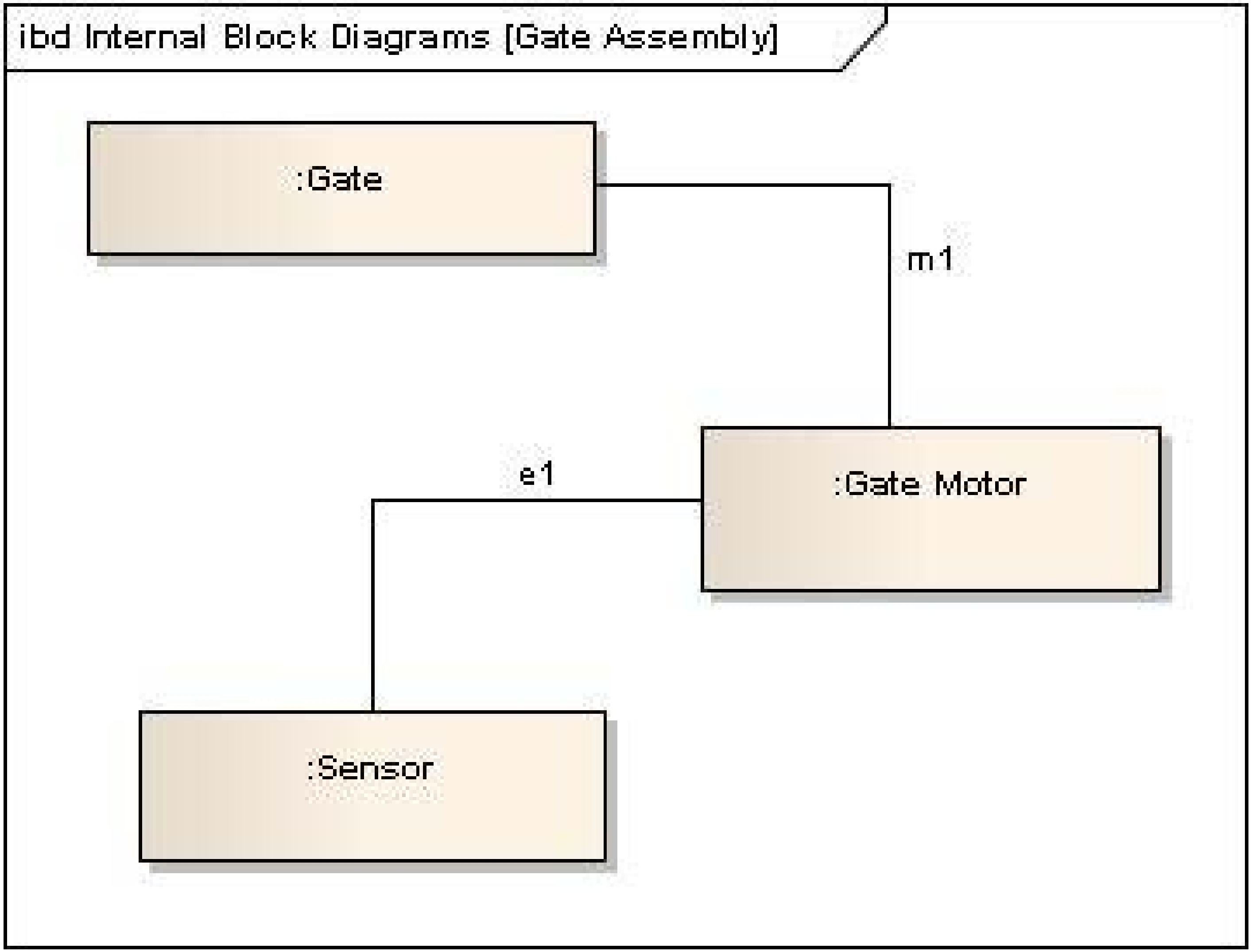
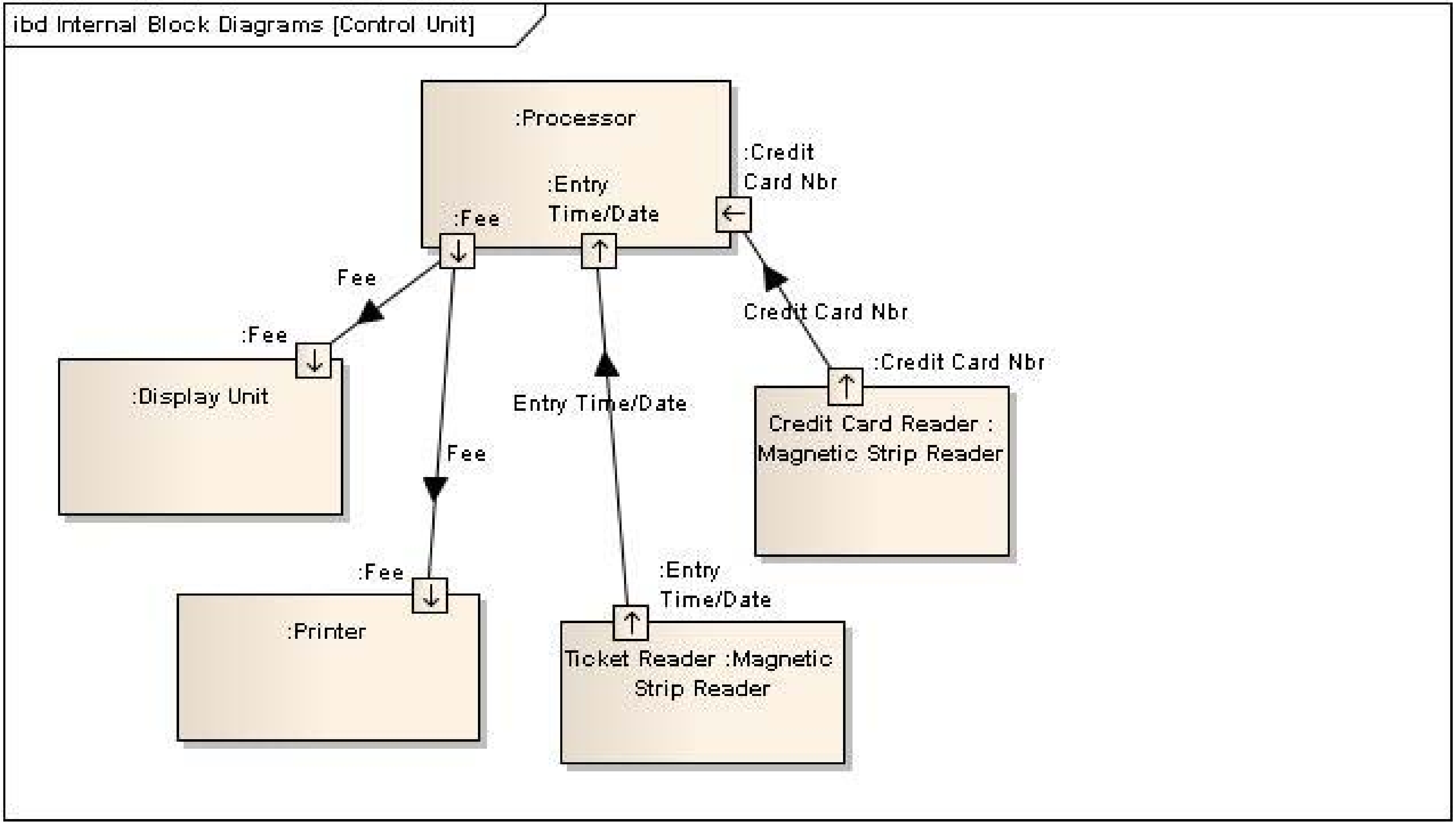


FIGURE 6.34

Internal Block Diagram for Gate Assembly



Internal Block Diagram for Control Unit



Summary

- Internal Block Diagrams are used to depict the internal structure of a block
- The frame of an IBD represents the enclosing block
- Internal Block Diagrams depict:
  - The usage of a block in a specific context
  - How parts/ports are connected
  - What flows between parts/ports
- Standard ports are used on an IBD to depict interfaces that specify the behavioral features (services) that a block either provides or requires

MODELING BEHAVIOR

MODELING FLOW-BASED BEHAVIOR WITH ACTIVITIES

Activities

- Activity specifies transformation of inputs to outputs through a controlled sequence of actions
- Secondary constructs show responsibilities for the activities using activity partitions (i.e., swim lanes)
- SysML extensions to Activities
  - Support for continuous flow modeling
  - Alignment of activities with Enhanced Functional Flow Block Diagram (EFFBD)

Control Power Activity Diagram

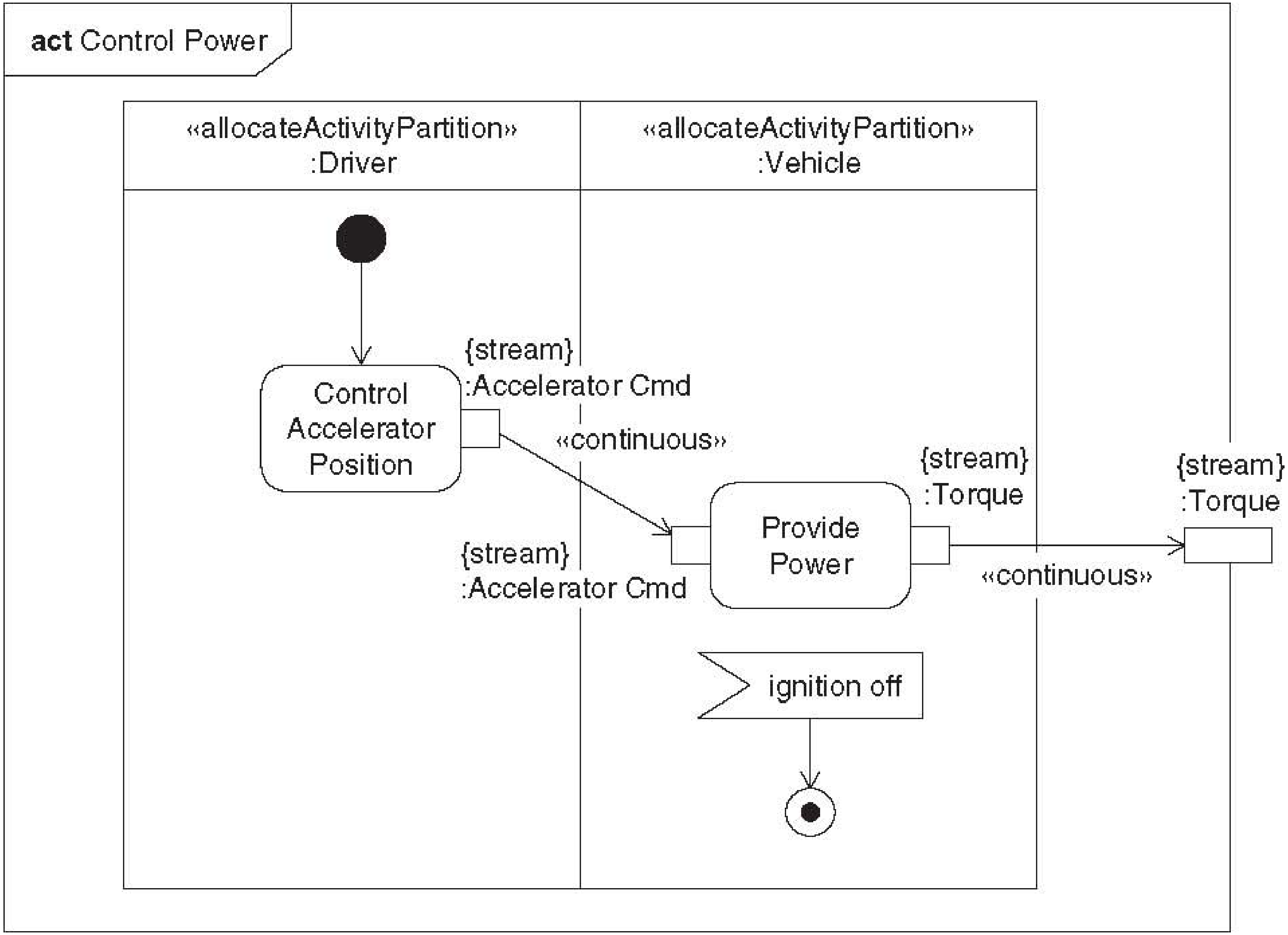
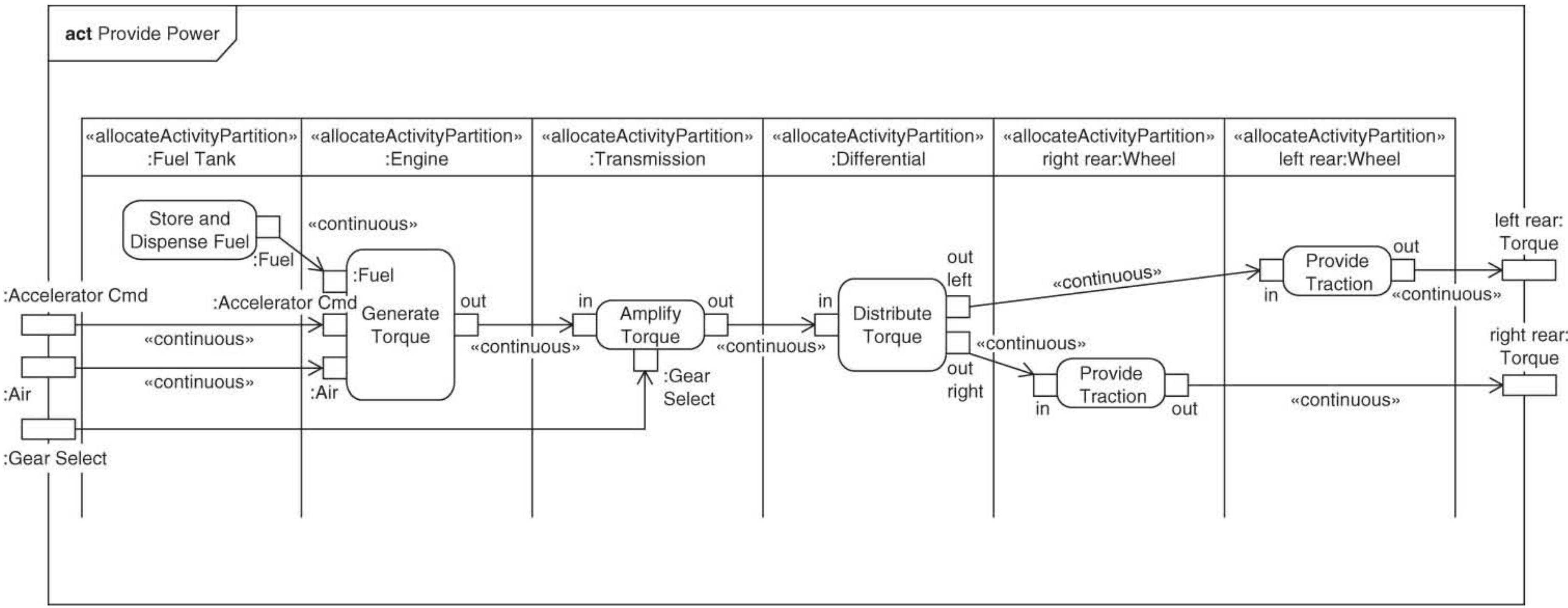


FIGURE 3.7

Provide Power Activity Diagram



© 2008 Elsevier, Inc.: A Practical Guide to SysML

FIGURE 3.11

Actions

- **Actions – describe how activities execute**
  - **Used to model the steps of the activity**
  - **Accept inputs and create outputs (depicted by ‘pins’)**
  - **Call Actions – represent activities that can be further decomposed into other actions**
    - **Allows for hierarchical modeling of activities**

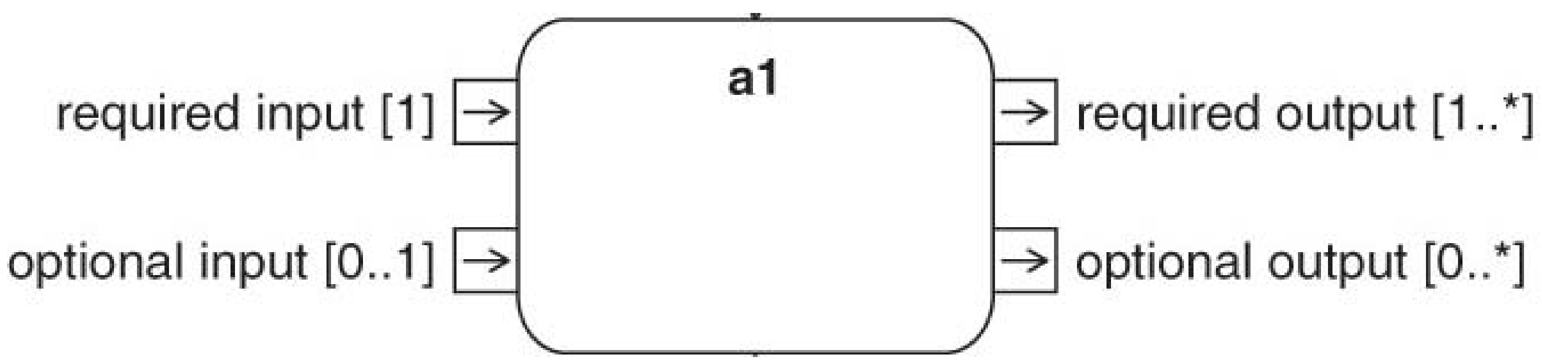


FIGURE 8.3

Decomposing an Activity Diagram with Call Behavior Actions

- Pins match Parameters in number and type
- Rake symbol denotes details are depicted on another diagram

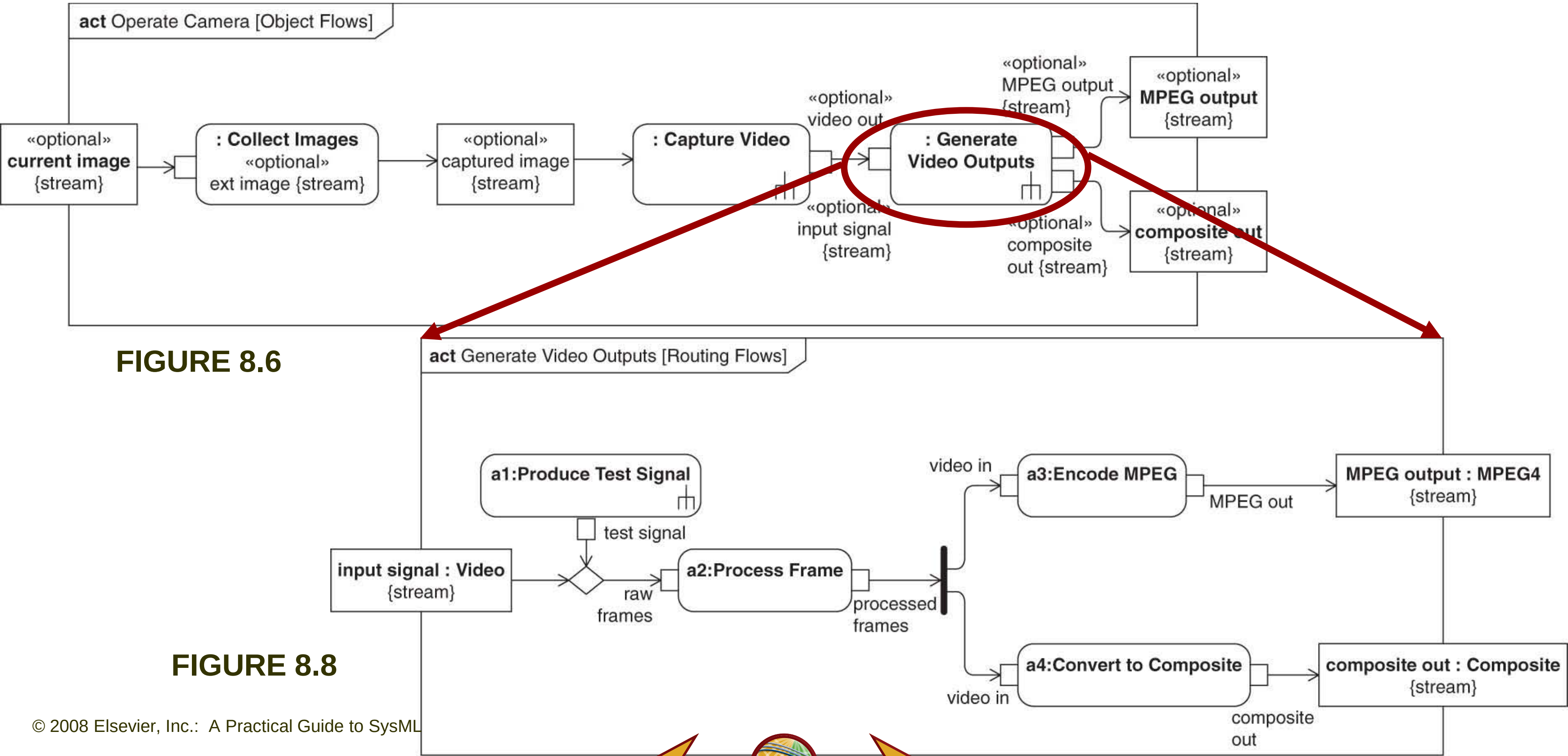


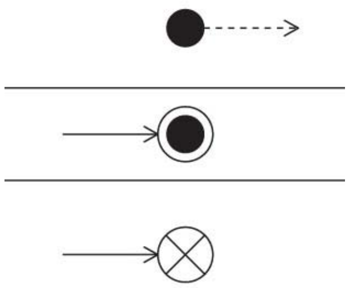
FIGURE 8.6

FIGURE 8.8

© 2008 Elsevier, Inc.: A Practical Guide to SysML

Initial, Activity Final, and Flow Final Nodes

- Initial Node – denotes where execution begins
  - Depicted by black circle
- Activity Final Node – denotes where execution terminates
  - Depicted by a bulls-eye
- Flow Final Node – terminates a particular sequence of actions without terminating the entire activity
  - Depicted by circle with cross-hair



Fork Nodes and Join Nodes

- **Fork Node – one input flow, multiple output flows**
  - **Output flows are independent and concurrent**
- **Join Node – multiple input flows, one output flow**
  - **Output occurs, only when all input tokens are available (default)**
- **Join Specification may override default**

Fork Node

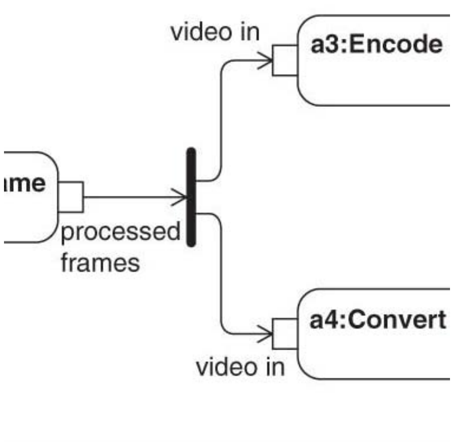


FIGURE 8.1

Join Node with Join Specification

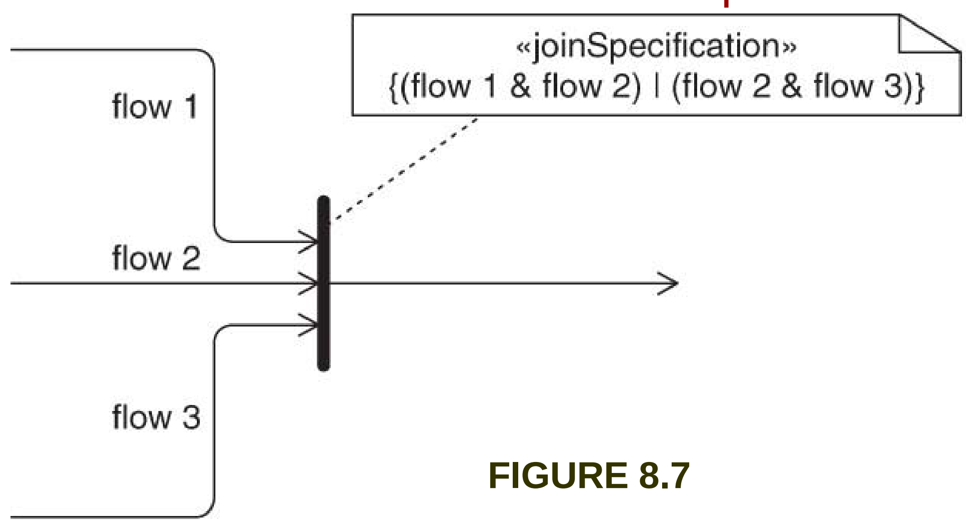


FIGURE 8.7

Decision Nodes and Merge Nodes

- Decision Nodes – one input, multiple output paths
  - Only one output path is valid, based on ‘guard’ conditions
  - Guards must be mutually exclusive
- Merge Node – multiple inputs, one output flow
  - Output flow is triggered upon arrival of any of the input flows

Decision Node

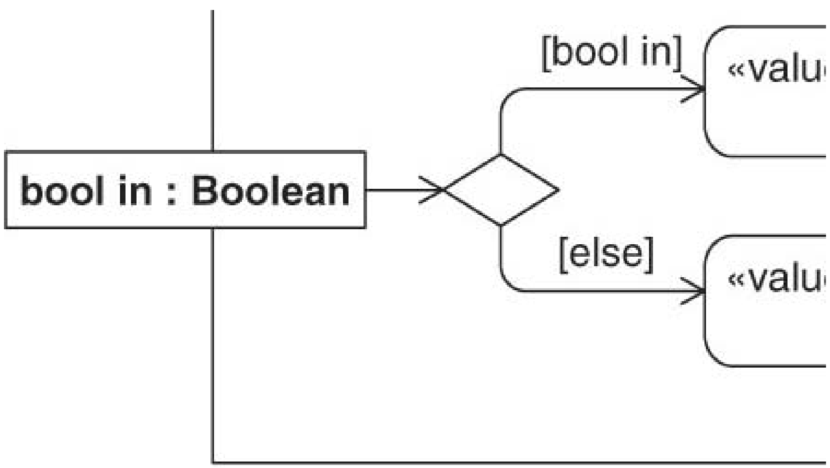


FIGURE 8.13

Merge Node

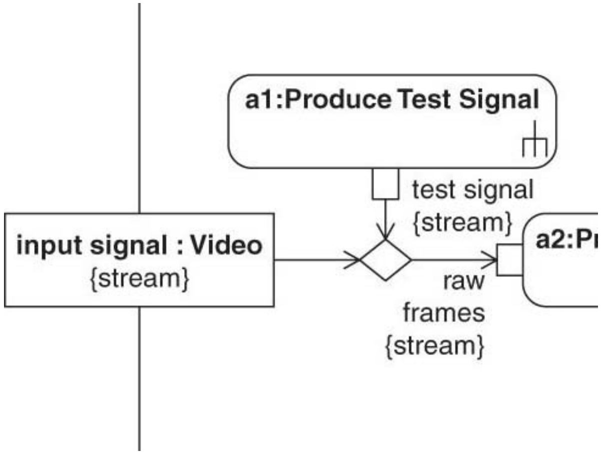


FIGURE 8.8

Control Flow

- Used to show sequence of actions
- Represents a control token
  - An action cannot start until it receives a control token on all input control flows
  - When an action is completed, it places control tokens on all outgoing control flows
- Can be depicted with a dashed arrow, to distinguish it from object flows
- Like object flow, can be used with:
  - Forks and Joins
  - Decision Nodes and Merges

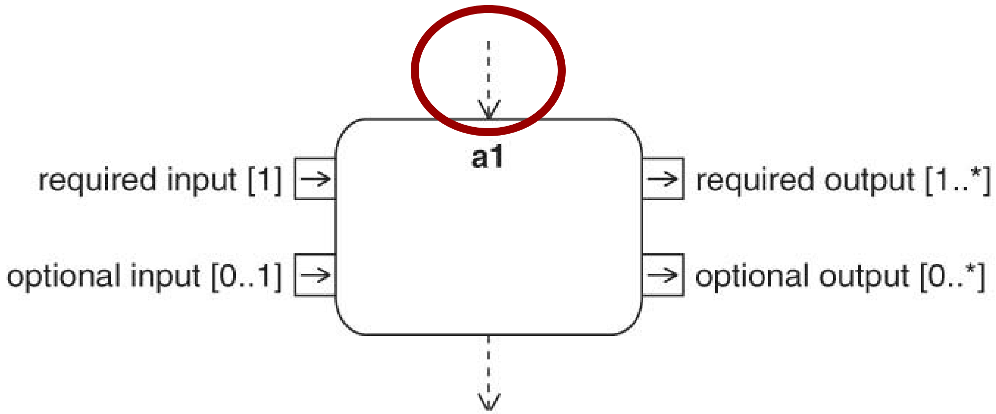


FIGURE 8.3

Partitions (aka Swimlanes)

- Allocates actions to an entity responsible for performing the action
- Can be used to specify functional requirements of an actor, component, or part
- Can be depicted horizontally or vertically

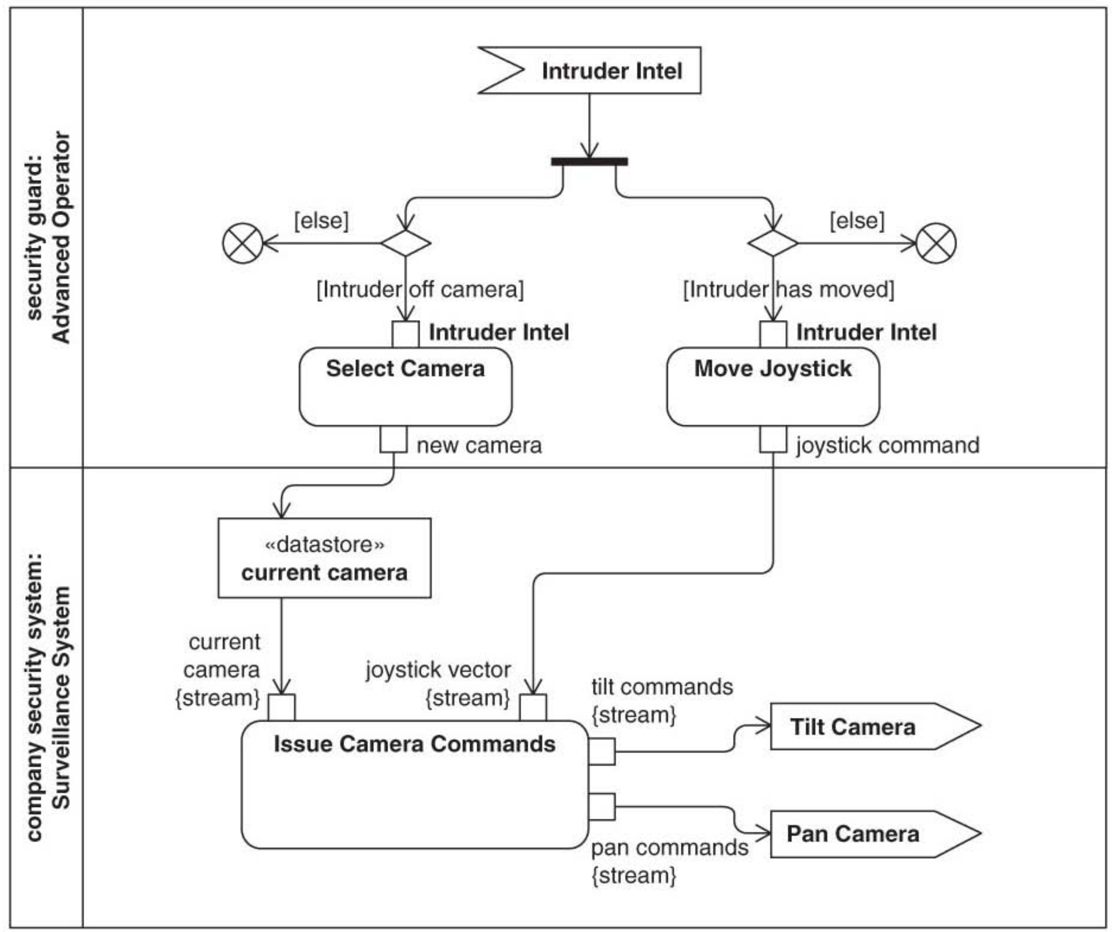
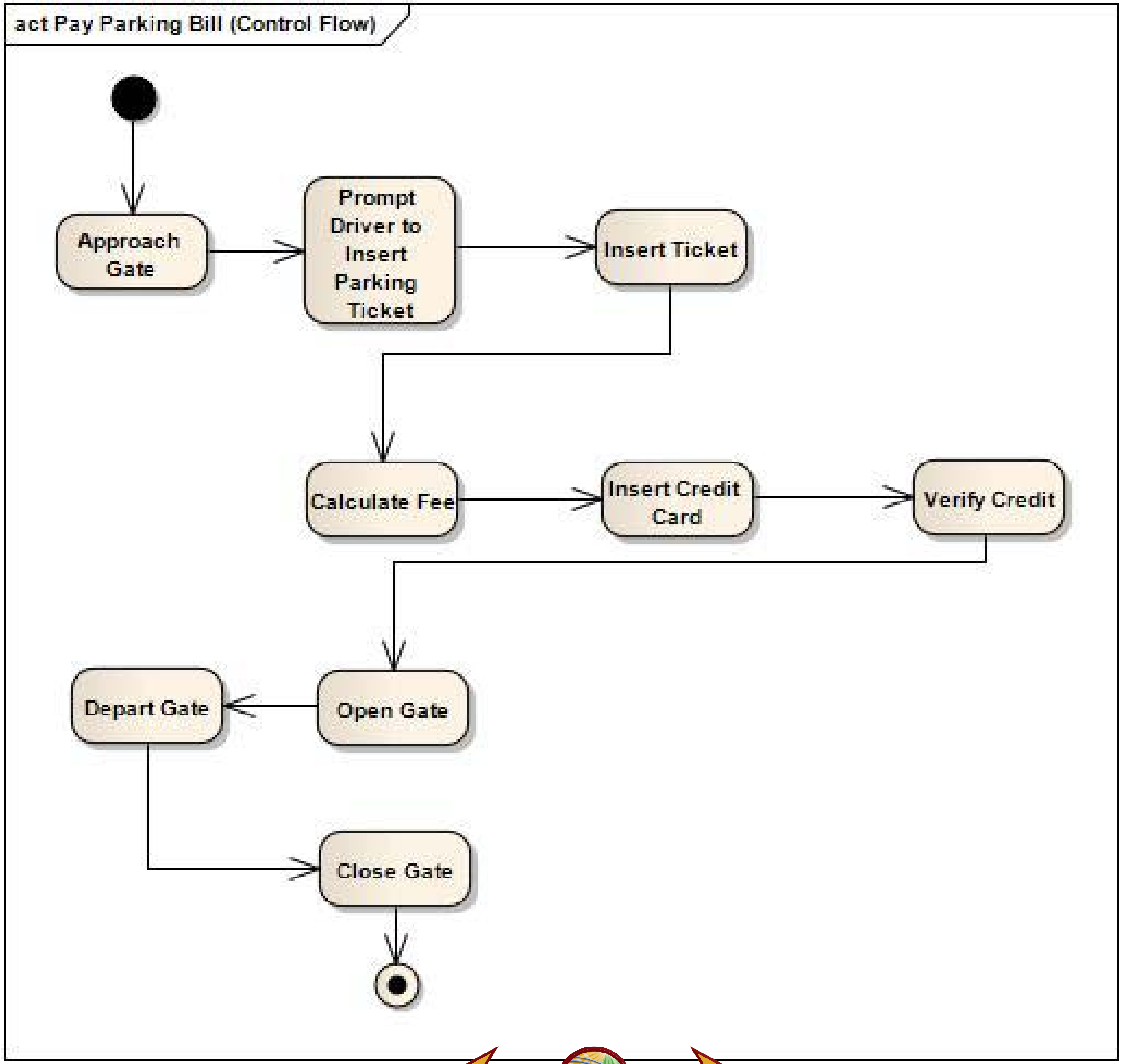
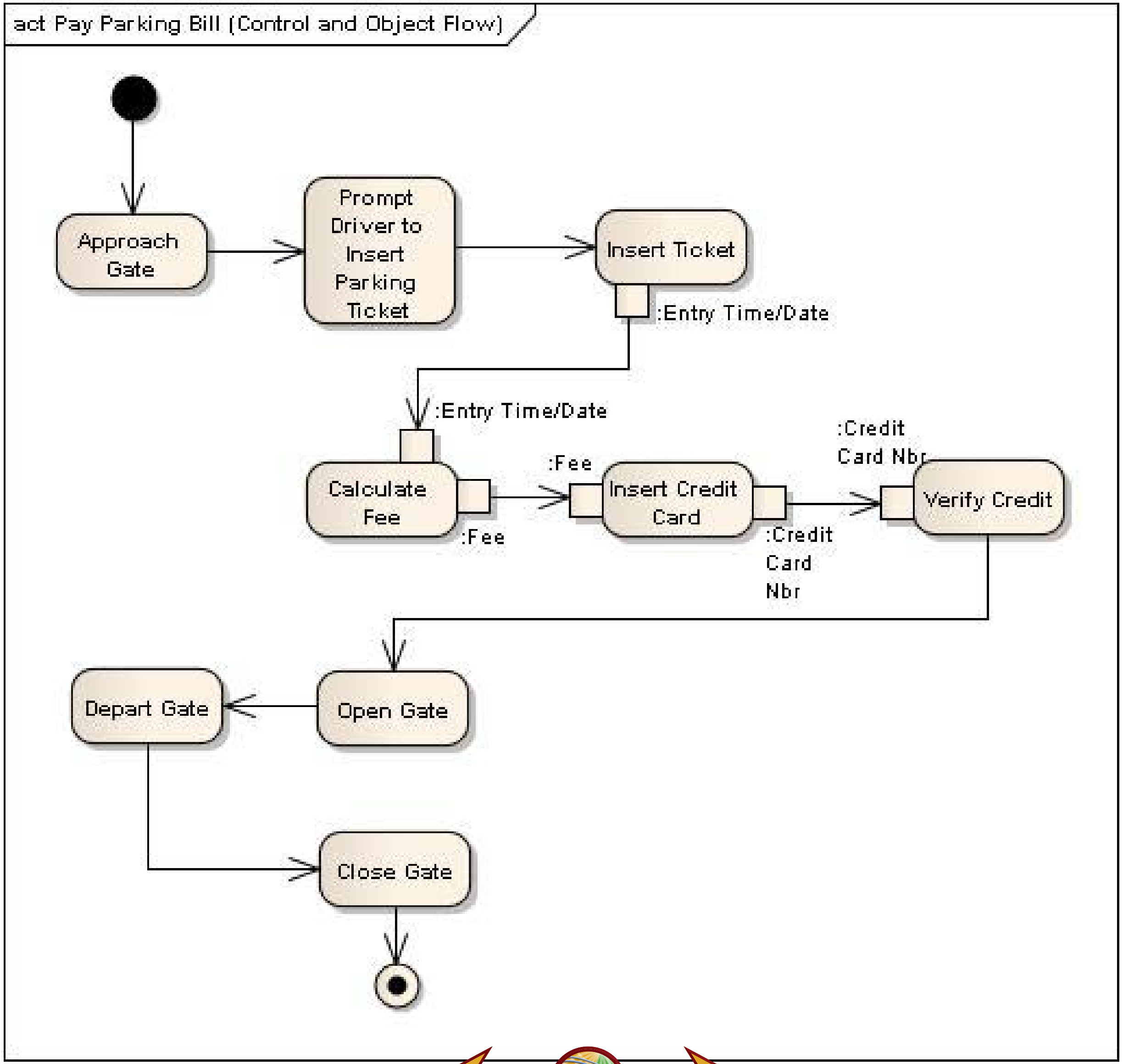


FIGURE 8.20

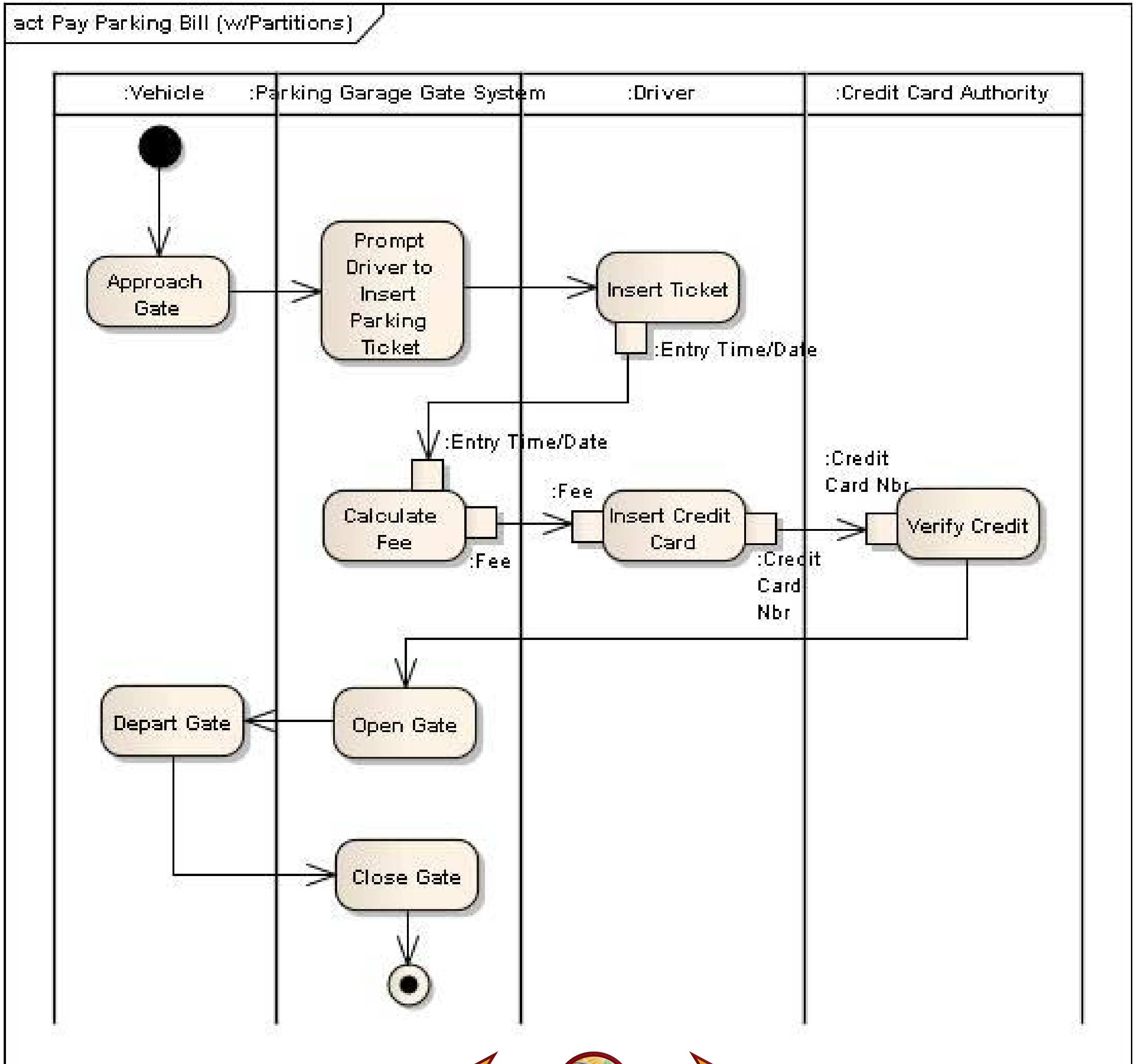
Activity Model (Primary Path)



Activity Model (w/Object Flow)

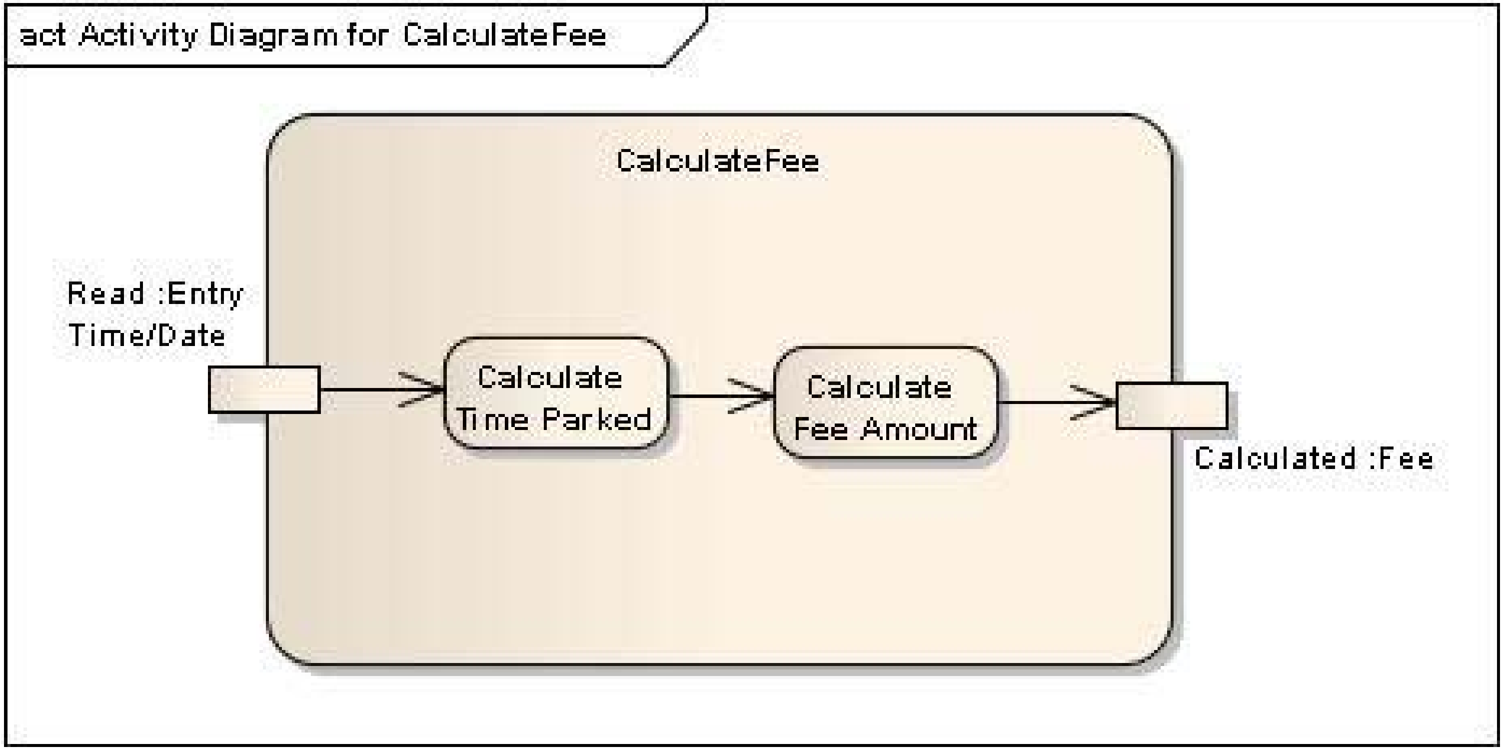


Activity Model (w/Partitions)



Decomposition of Calculate Fee

- Example below shows use of Input and Output Parameters for the Calculate Fee Activity
- Hierarchical relationship of Activities and Actions



Summary

- Activity Diagrams are used to model behavior that specifies the transformation of inputs to outputs through a controlled sequence of Actions
- Activities can have multiple inputs or outputs called parameters
- Activities are made up of actions
- Actions consume input tokens and produce output tokens via pins
- Inputs/outputs can either be streaming or non-streaming
- Object Flows are used to depict the flow of object tokens from one action to other actions
- Control Flows are used to depict the transfer of control from one action to other actions using control tokens
- Call behavior actions can be further decomposed by calling other activities
- Partitions are used to assign responsibility for actions to blocks or parts that the partition represent

MODELING MESSAGE-BASED BEHAVIOR WITH INTERACTIONS

Interactions

- Sequence diagrams provide representations of message based behavior
  - represent flow of control
  - describe interactions between parts
- Sequence diagrams provide mechanisms for representing complex scenarios
  - reference sequences
  - control logic
  - lifeline decomposition
- SysML does not include timing, interaction overview, and communications diagram

Start Vehicle Sequence Diagram

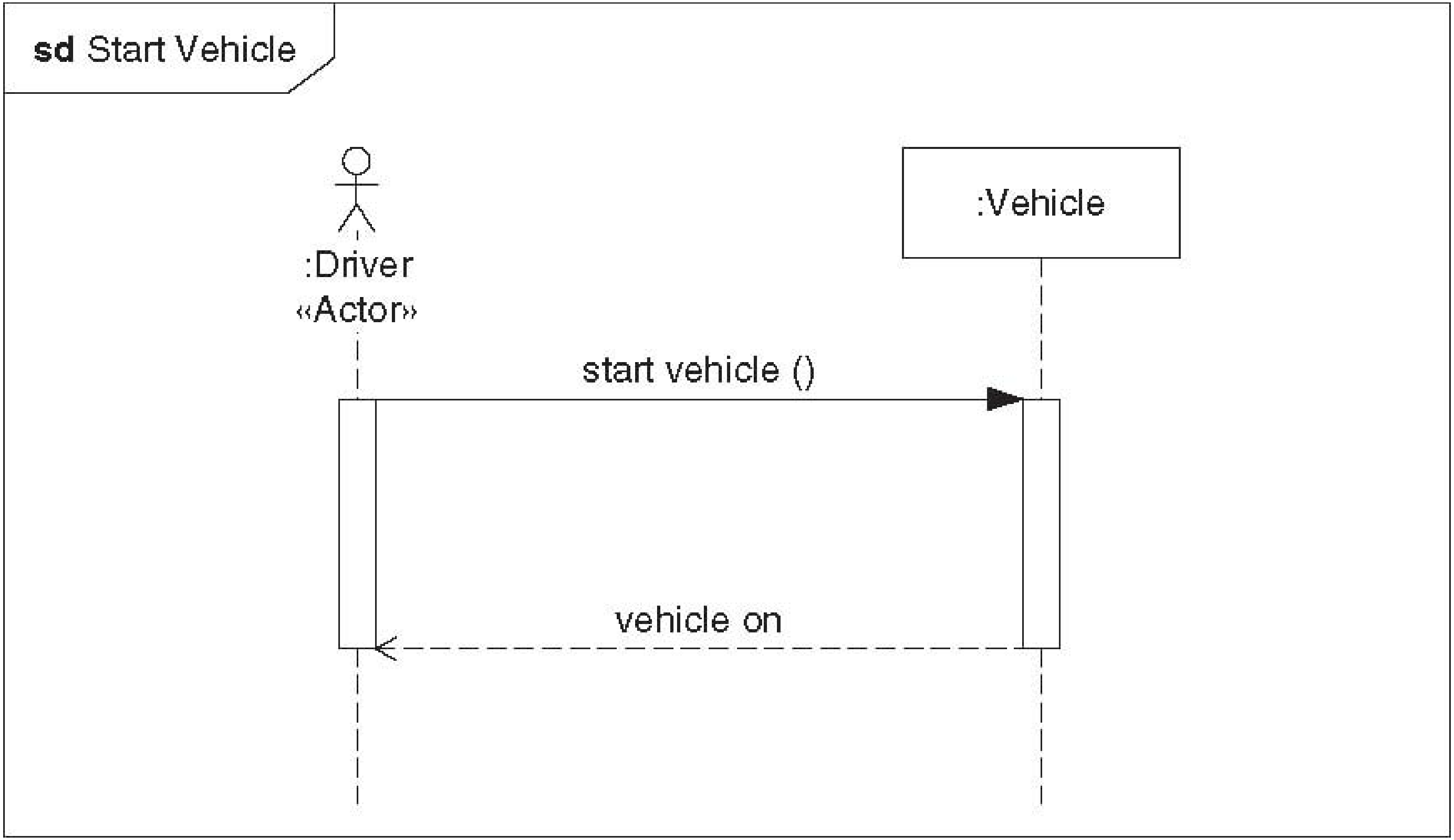


FIGURE 3.6

Drive Vehicle Sequence Diagram

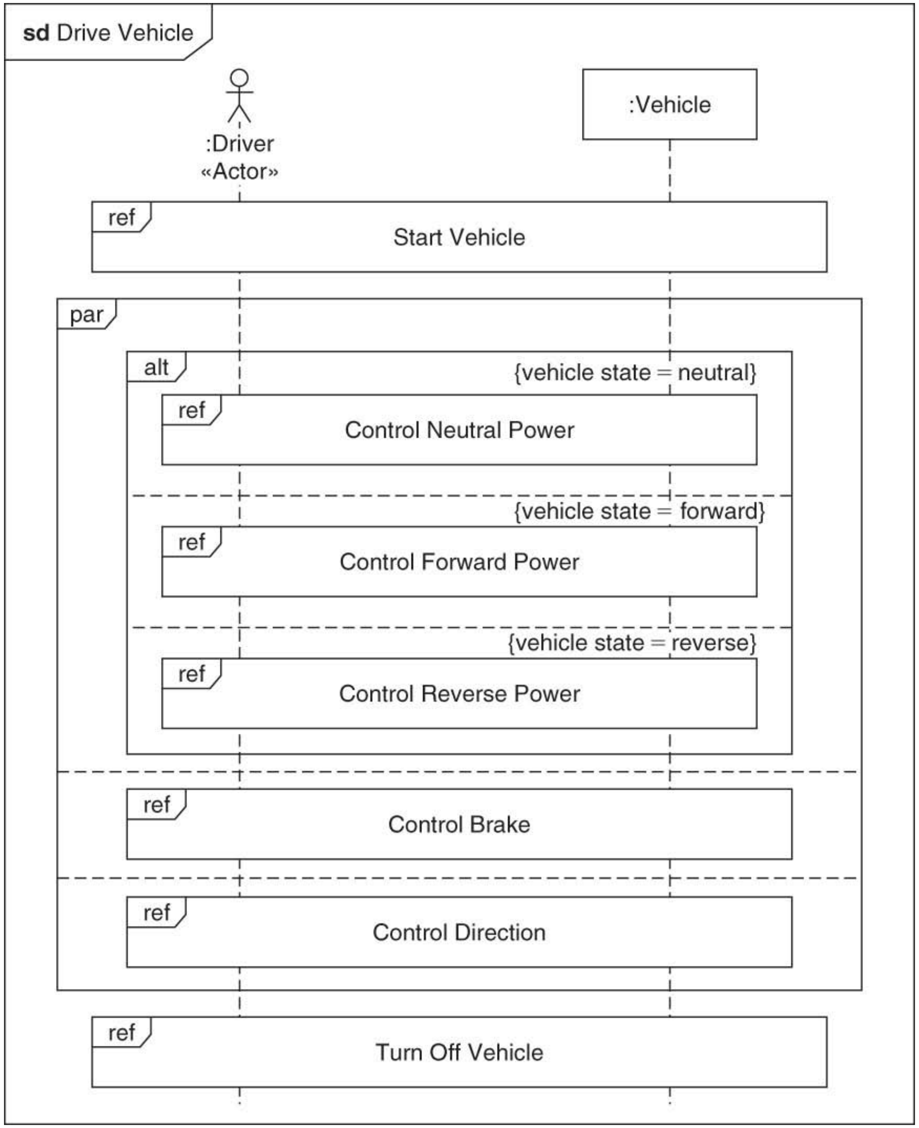
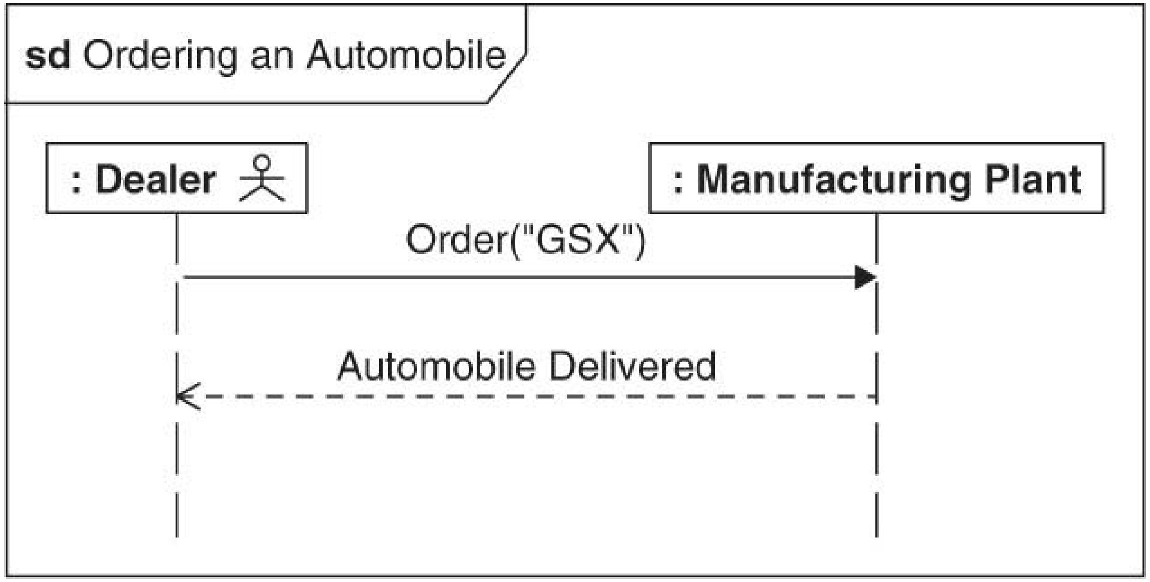


FIGURE 3.5

Sequence Diagram Components

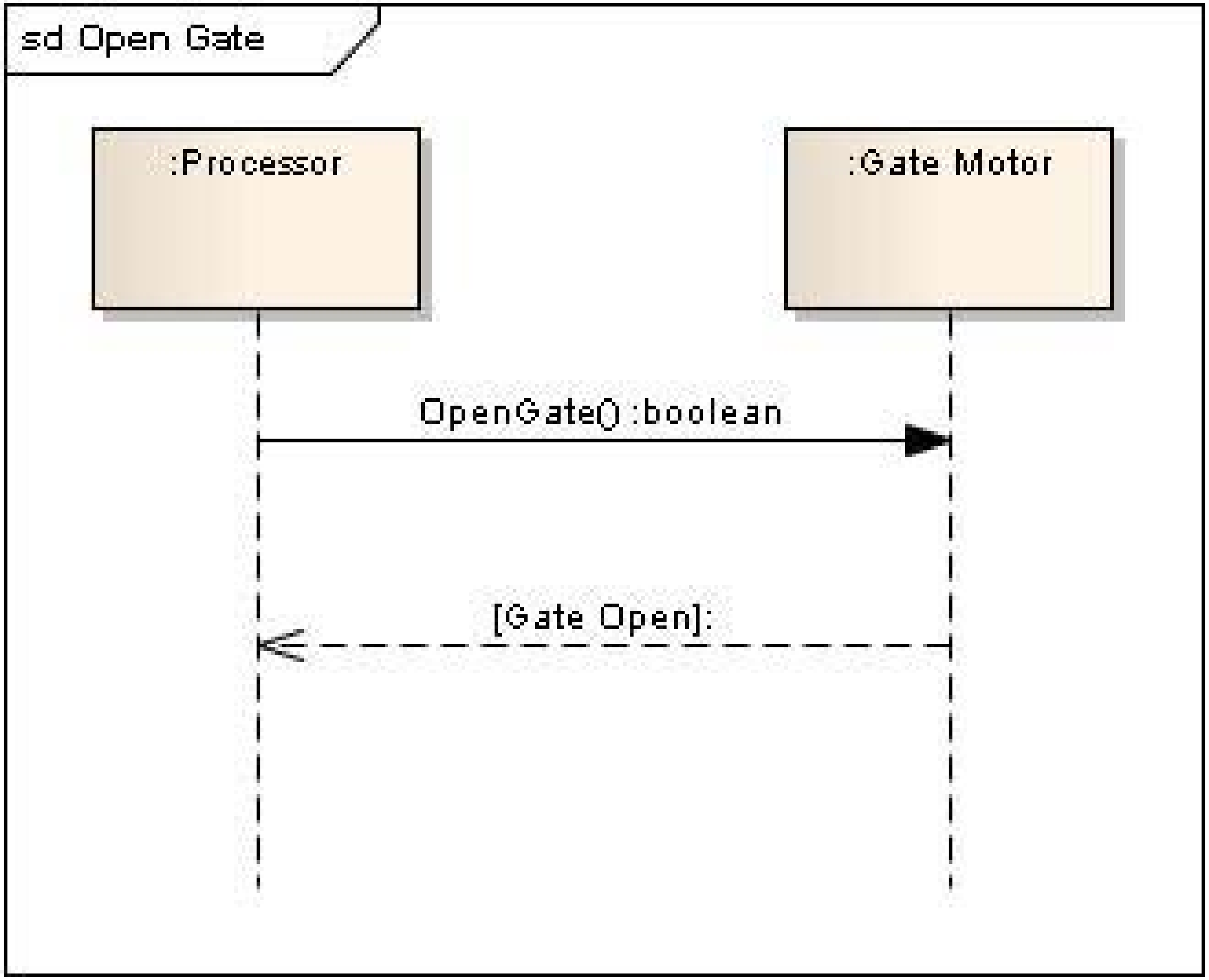
- Sequence diagrams can be comprised of the following:
  - Lifelines
    - Represents a Structural Element of a system
    - Depicts 'Time'
  - Messages
    - Asynchronous
    - Synchronous
    - Reply



© 2008 Elsevier, Inc.: A Practical Guide to SysML

FIGURE 9.4

Sequence Diagram for Opening the Gate



Summary

- Sequence Diagrams are used to depict the interactions between structural elements of a Block
- Sequence Diagrams are comprised of:
  - Lifelines
  - Messages
- Lifelines represent the structural element and depicts Time
- Messages can be either:
  - Asynchronous
  - Synchronous
  - Reply
- Messages represent a call for an operation
- Combined Fragments are used to depict complex interactions and include: alternate paths, parallel paths, optional paths or loops
- Reference Interactions depict re-use of common interactions

MODELING EVENT-BASED BEHAVIOR WITH STATE MACHINES

State Machines

- Typically used to represent the life cycle of a block
- Support event-based behavior (generally asynchronous)
  - Transition with trigger, guard, action
  - State with entry, exit, and do-activity
  - Can include nested sequential or concurrent states
  - Can send/receive signals to communicate between blocks during state transitions, etc.
- Event types
  - Change event
  - Time event
  - Signal event

Drive Vehicle States

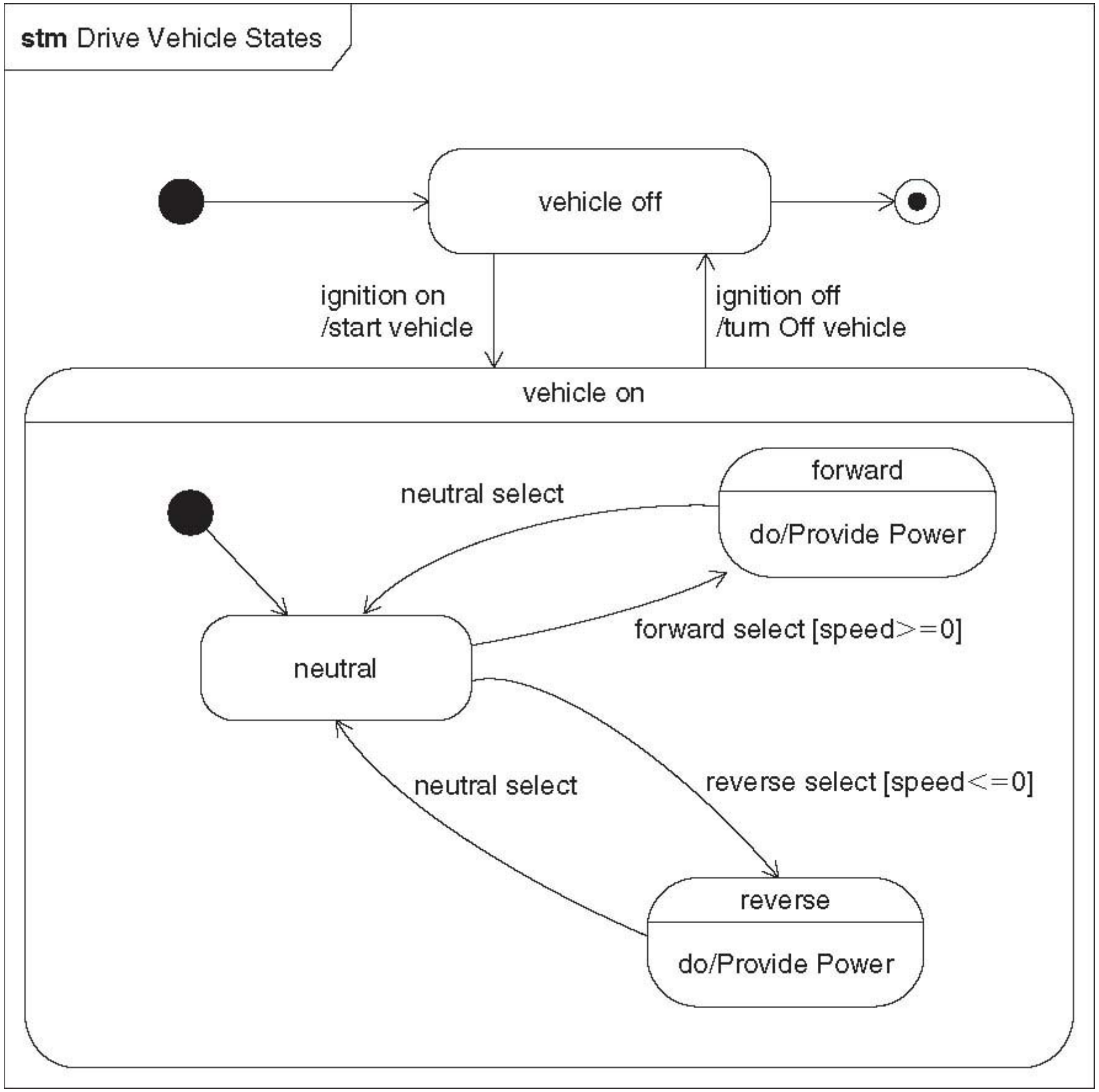


FIGURE 3.8

© 2008 Elsevier, Inc.: A Practical Guide to SysML

State Machine Diagram Components

- State Machine diagrams can be comprised of the following:
  - States
    - Behaviors
  - Transitions
  - Composite States

States

- States – represents a condition in the life of a block
- Initial State – represented by a black solid dot
- Final State – represented by a bulls-eye

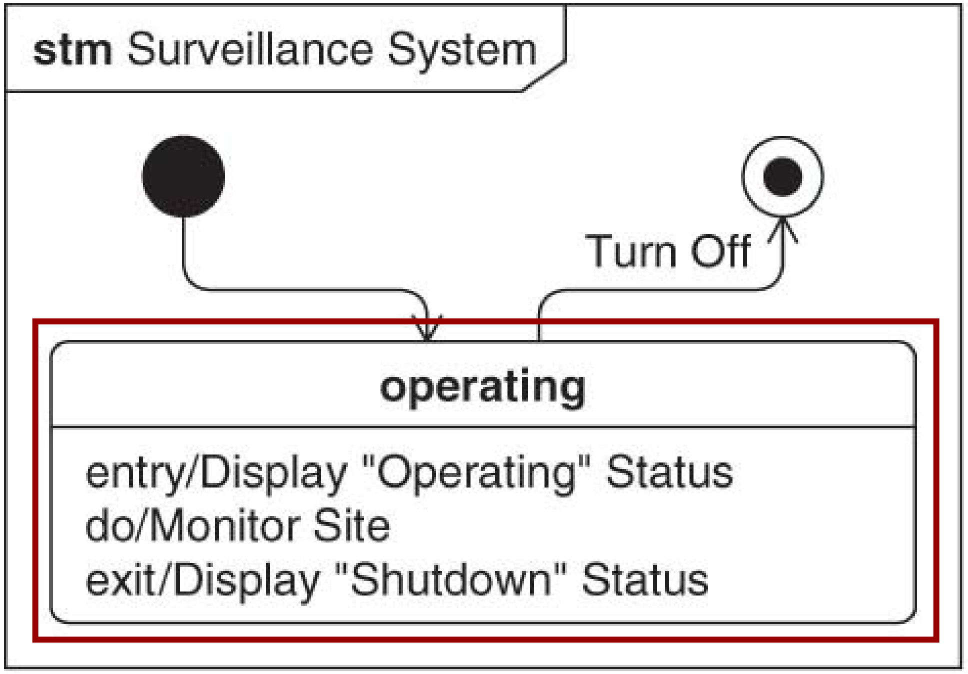


FIGURE 10.2

Behaviors

- Actions of a State
  - Types:
    - Entry – what happens when the state is entered
    - Exit – what happens when the state is exited
    - Do – what happens while in a state

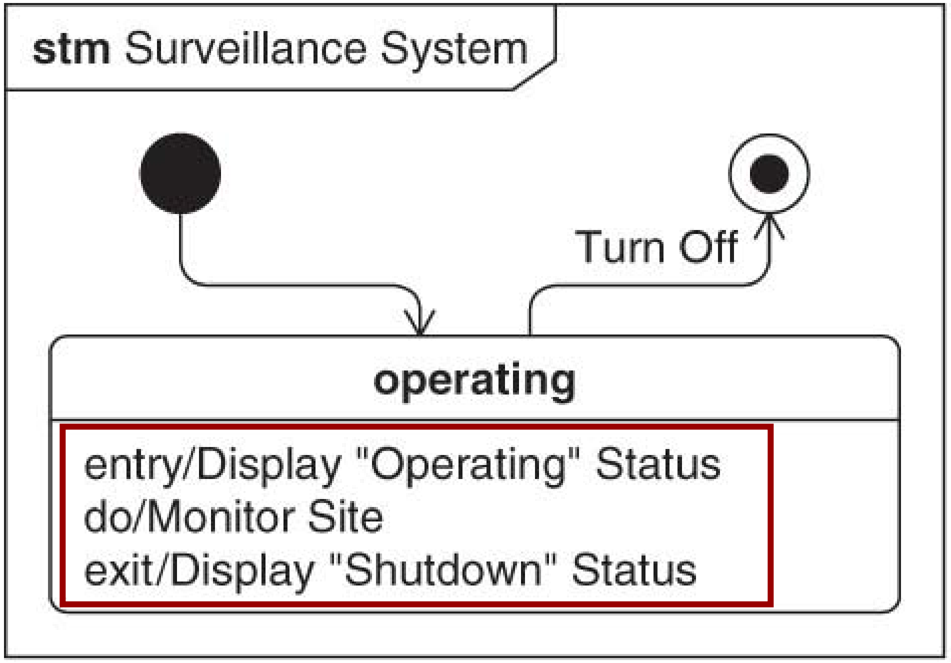


FIGURE 10.2

Transitions

- Used to show the flow from one state to another (solid arrow)
- Can consist of triggers, guards, and effects

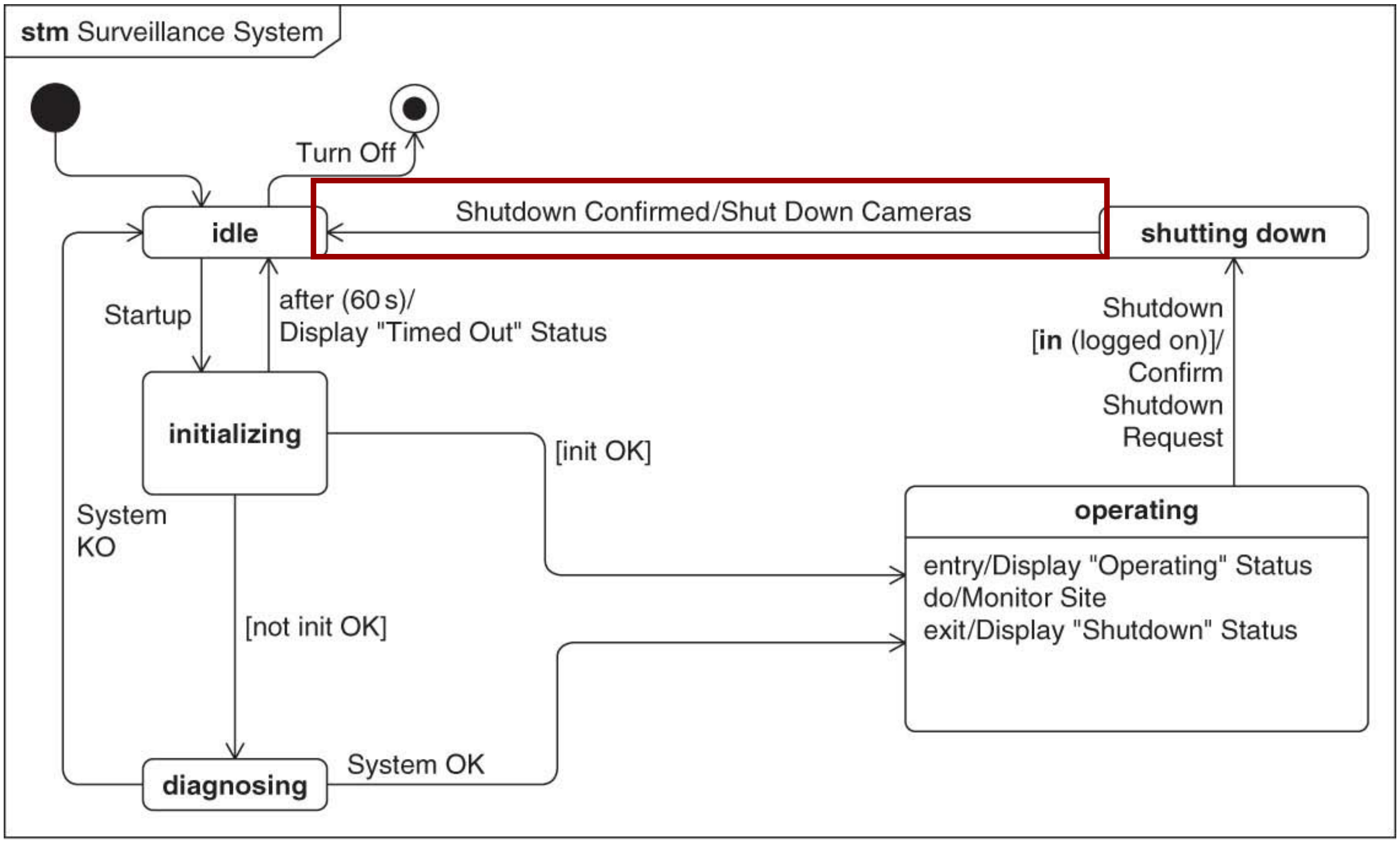


FIGURE 10.3

Composite States

- Means of depicting the hierarchy of states
- Sub-states – states that are unique to another state of an entity
- Composite States are depicted by enclosing sub-states within a state

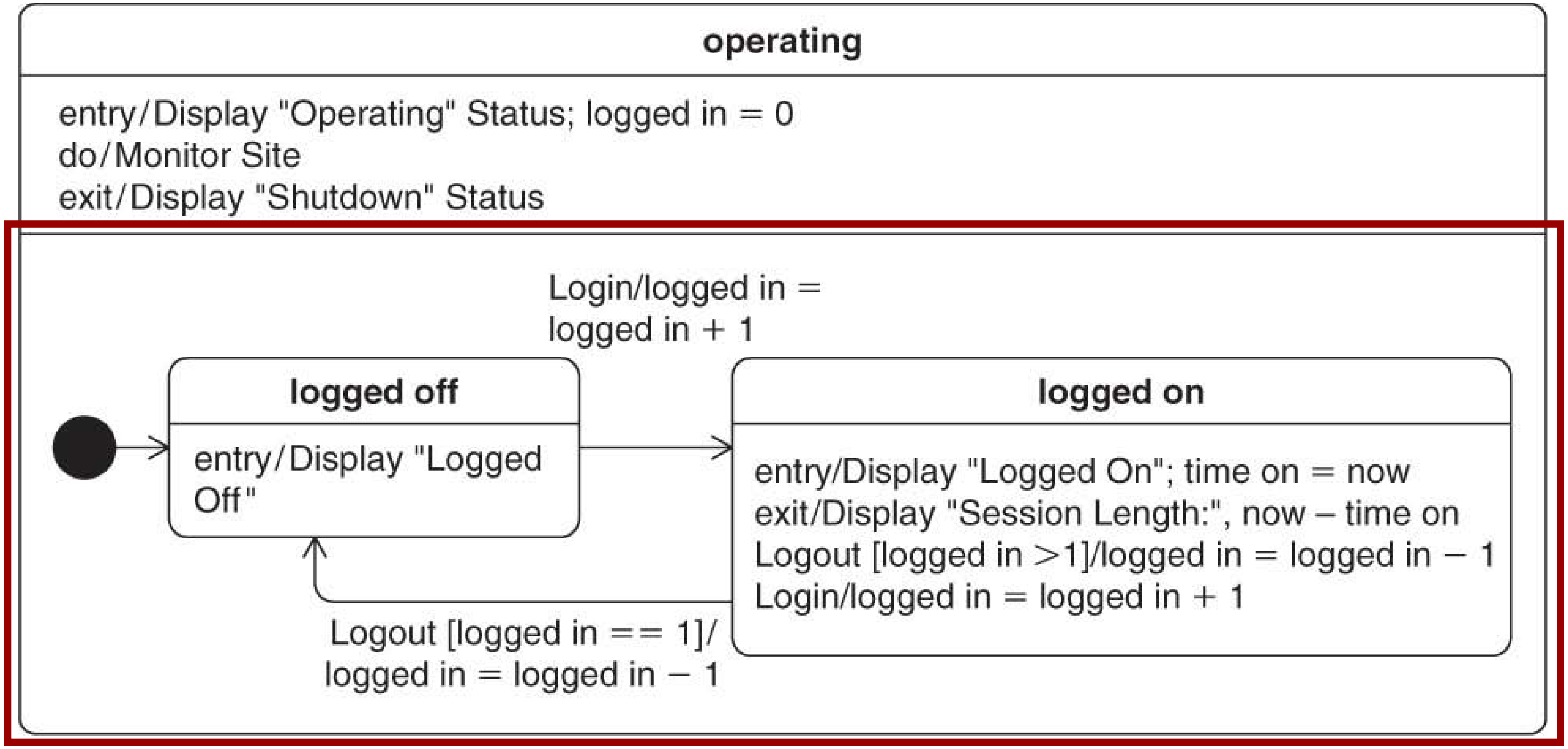
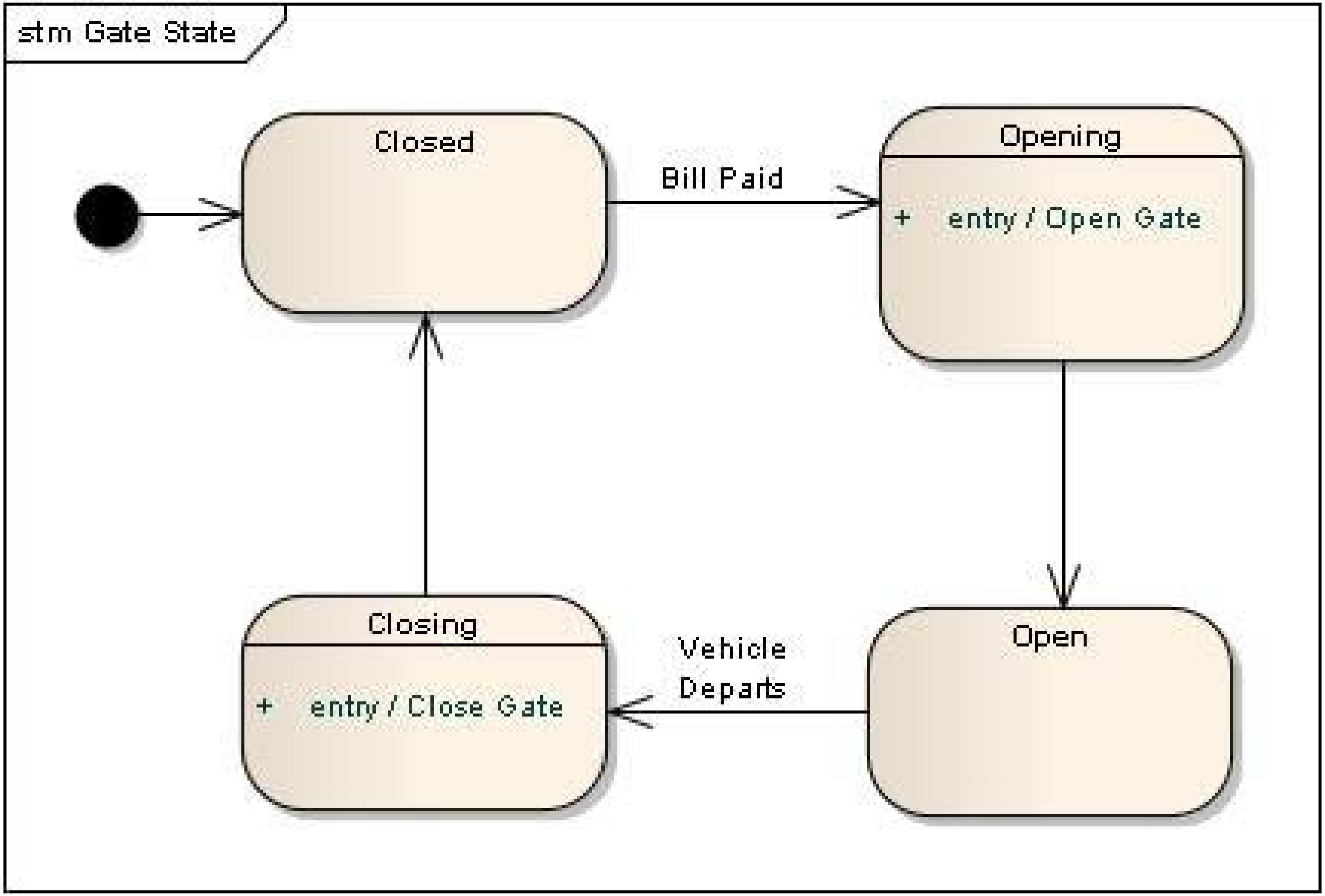


FIGURE 10.9

State Machine for Parking Garage Gate



Summary

- State Machines Diagrams are used to depict how a Block changes State
- State Machines can be comprised of:
  - States
  - Transitions
  - Composite States
- States represent a condition in the life of a Block
- Behaviors are the actions associated with a State
- Transitions are used to show how a Block changes from one State to another
- Transitions can consist of Triggers, Guards, and Effects
- Composite States are used to depict the hierarchy of States

MODELING CONSTRAINTS WITH PARAMETRICS

Parametrics

- **Used to express constraints (equations) between value properties**
  - **Provides support for engineering analysis (e.g., performance, reliability)**
  - **Facilitates identification of critical performance properties**
- **Constraint block captures equations**
  - **Expression language can be formal (e.g., MathML, OCL) or informal**
  - **Computational engine is provided by applicable analysis tool and not by SysML**
- **Parametric diagram represents the usage of the constraints in an analysis context**
  - **Binding of constraint parameters to value properties of blocks (e.g., vehicle mass bound to parameter 'm' in F= m × a)**

Parametrics Enables Integration of Engineering Analysis with Design Models

Vehicle Acceleration Analysis Parametric Diagram

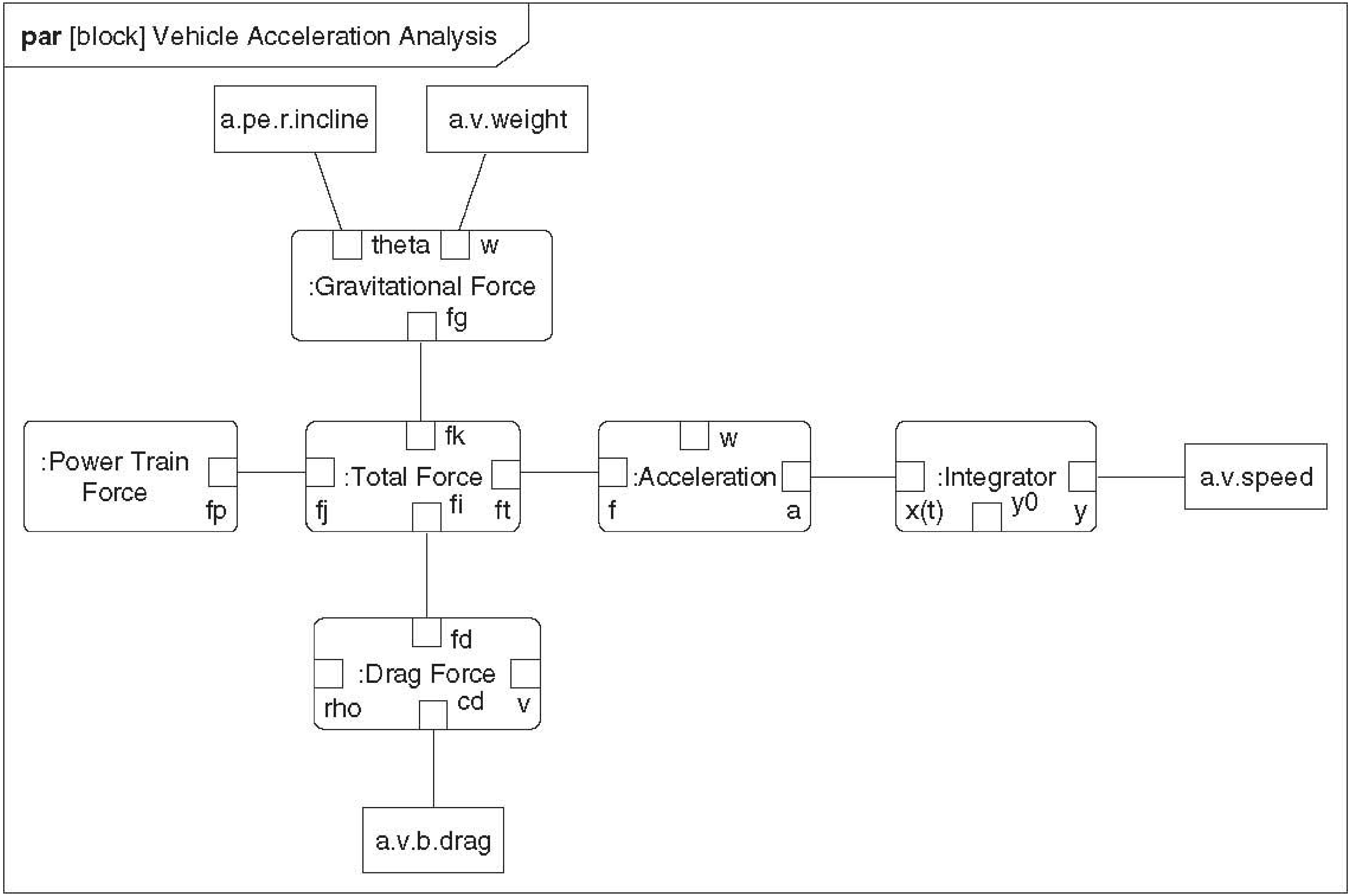


FIGURE 3.14

Defining Constraints in Constraint Blocks

- Constraint Blocks
  - Support the construction of parametric models
  - Define equations so that they may be re-used and inter-connected
  - Define a set of parameters
    - Contained in the ‘parameters’ compartment
  - Define an expression that constrains the parameters
    - Contained in the ‘constraints’ compartment
  - Depicted with the keyword <<constraint>>

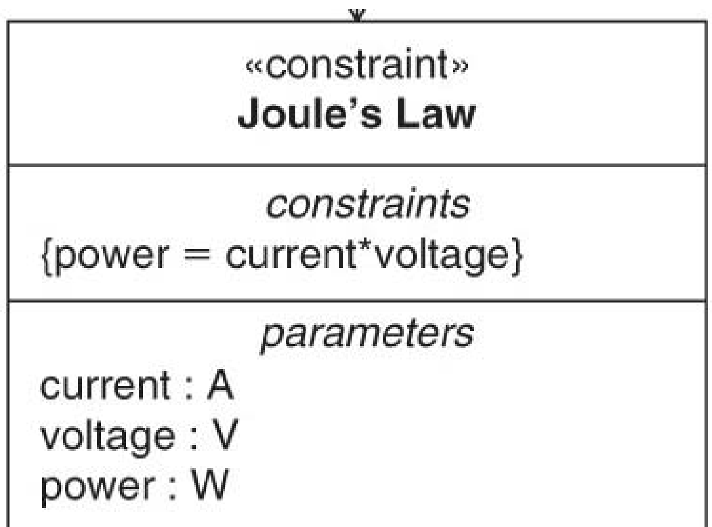
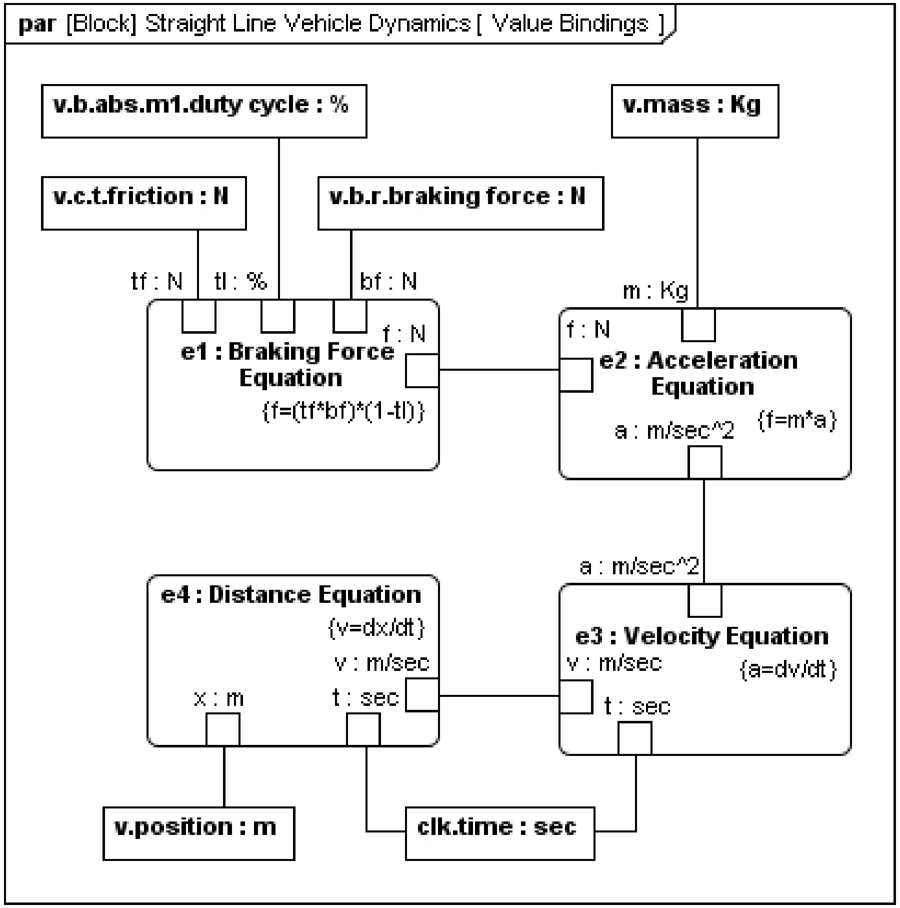


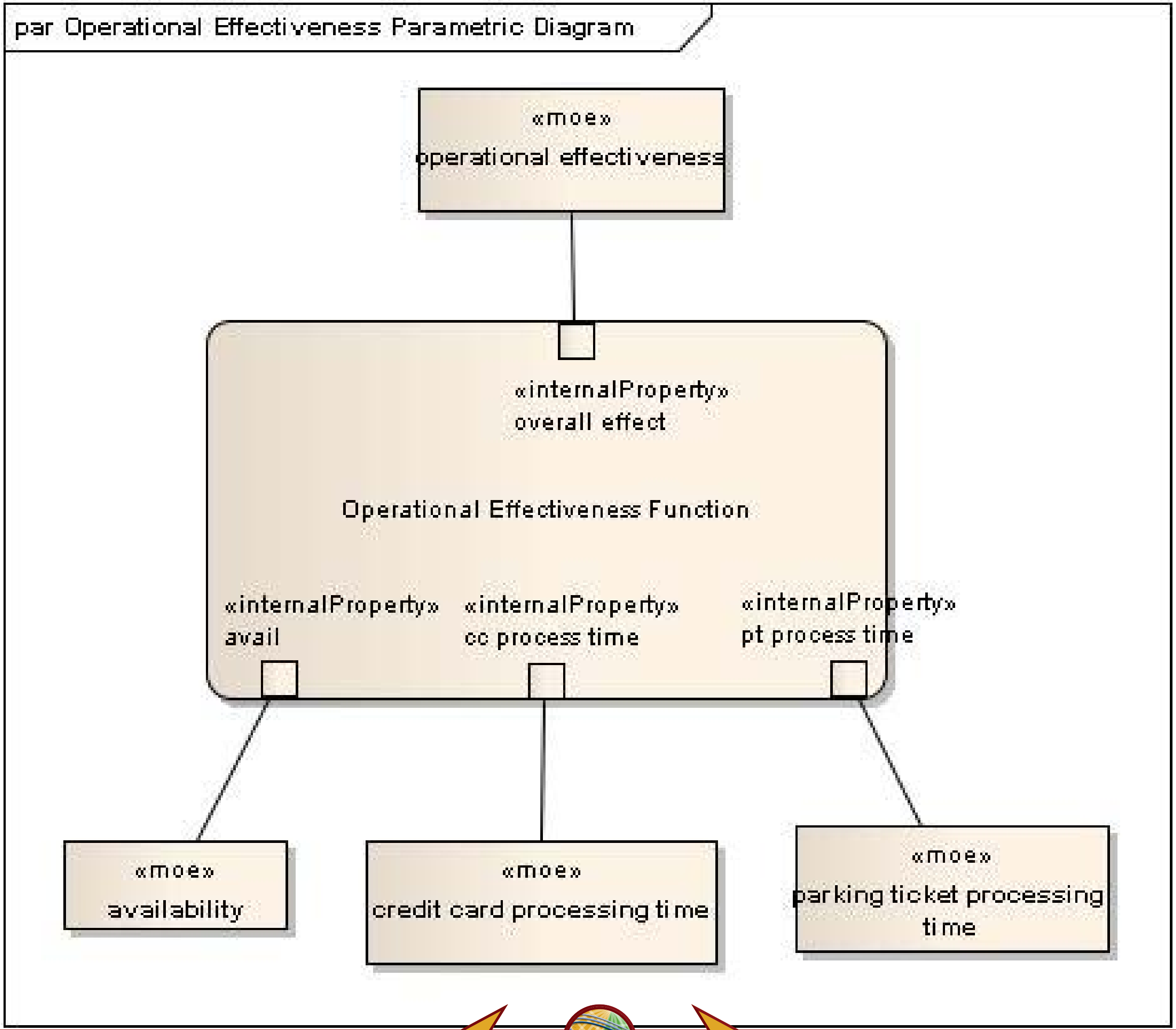
FIGURE 7.1

Defining Parametric Models

- Parametric models:
  - Depict a network of equations that constrain the properties of blocks
  - The properties of the system are bound to the parameters of the analysis equations (e.g. vehicle mass is bound to 'm' in F=m x a)
- Example: in the figure, properties of the vehicle are bound to the parameters of the equations used to analyze vehicle stopping distance
- Parametric models thus help identify the properties of the system that are critical to satisfying requirements



Top-Level Parametric Diagram for Gate System



Summary

- Parametric diagrams
  - Capture the analysis as a network of equations
  - Help ensure consistency between the system design model and multiple engineering analysis models
  - Help to manage technical performance measures
- Constraint Blocks
  - Define parameters and constraint expressions
  - Represented on a Block Definition Diagram
- Constraint Property
  - Usage of constraint blocks
  - Represented on a Parametric Diagram

MODELING CROSS CUTTING RELATIONSHIPS WITH ALLOCATIONS

Allocation Relationships

- **Allocation Relationships: Mapping Between Any Two Named Model Elements**
- **A Named Model Element is Allocated to (allocatedTo) or Allocated From (allocatedFrom) Other Model Elements.**
- **Example: System Behavioral Allocation (or Functional Allocation)**
  - **Allocation of System Activities to Blocks**
    - **Each Block Responsible for Executing a Particular Activity**

Allocation Relationships

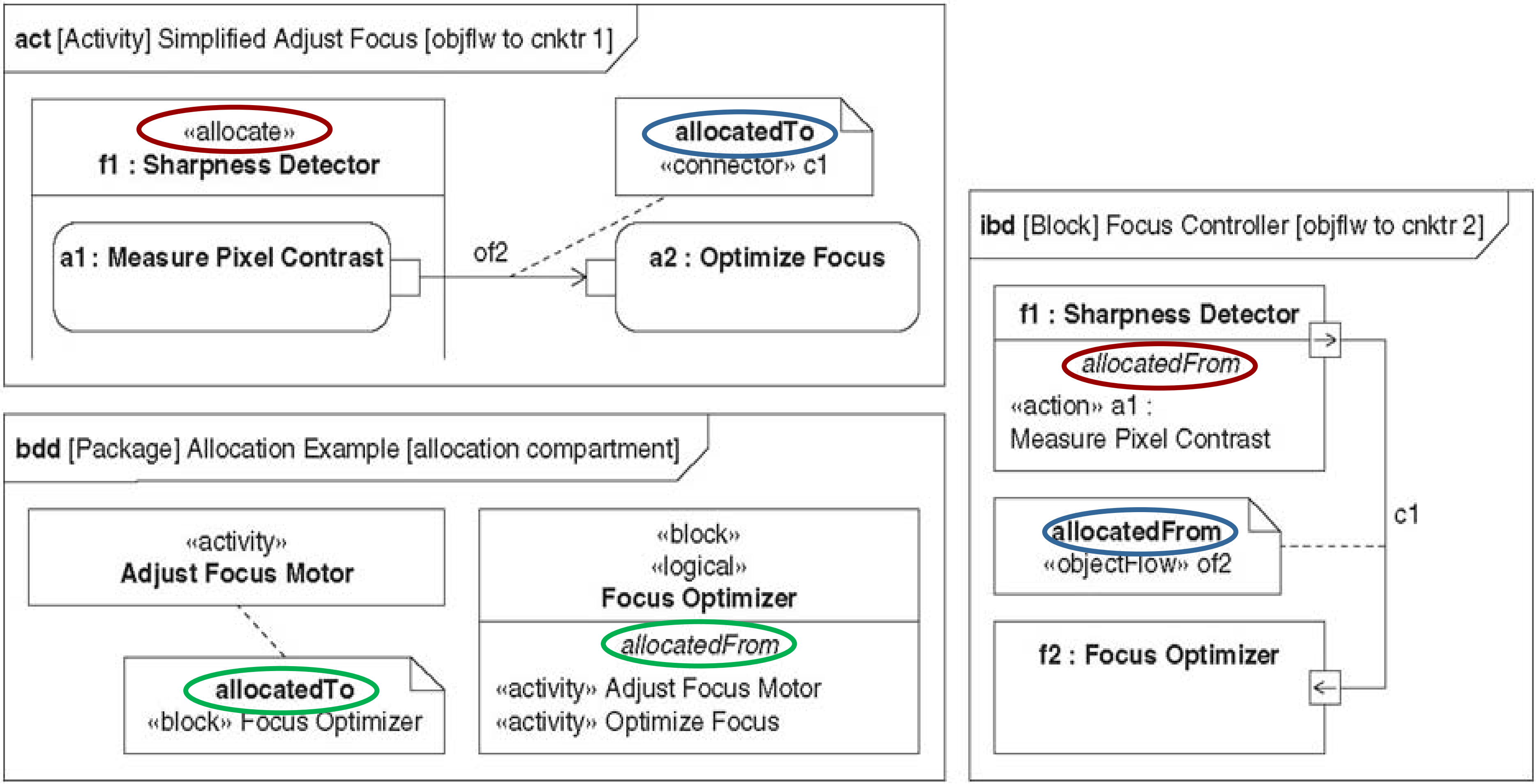


FIGURE 13.1

Allocation Notation

- <u>Tabular (Table or Matrix) Notation</u>: Multiple Allocation Relationships
  - Not specifically prescribed by SysML specification (Tools Vary)
  - Useful for concise, compact Allocations Representations

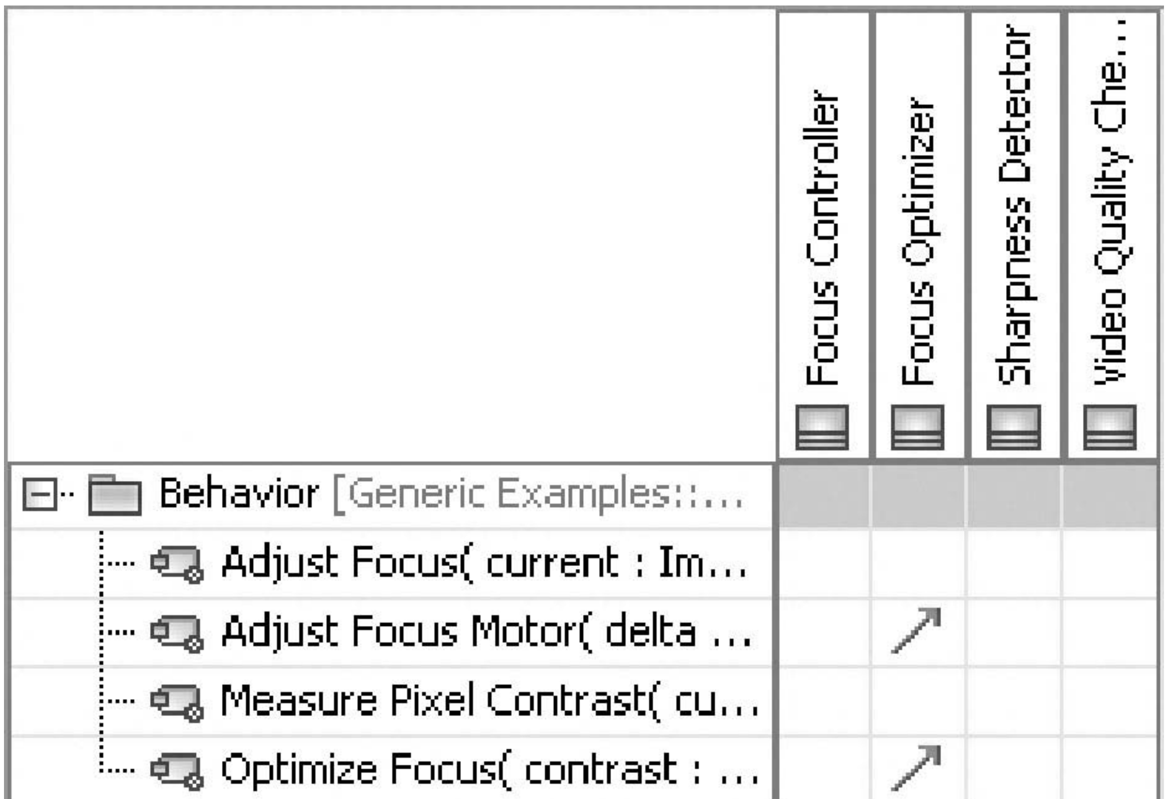


FIGURE 13.5

Functional Allocations for Parking Garage Gate

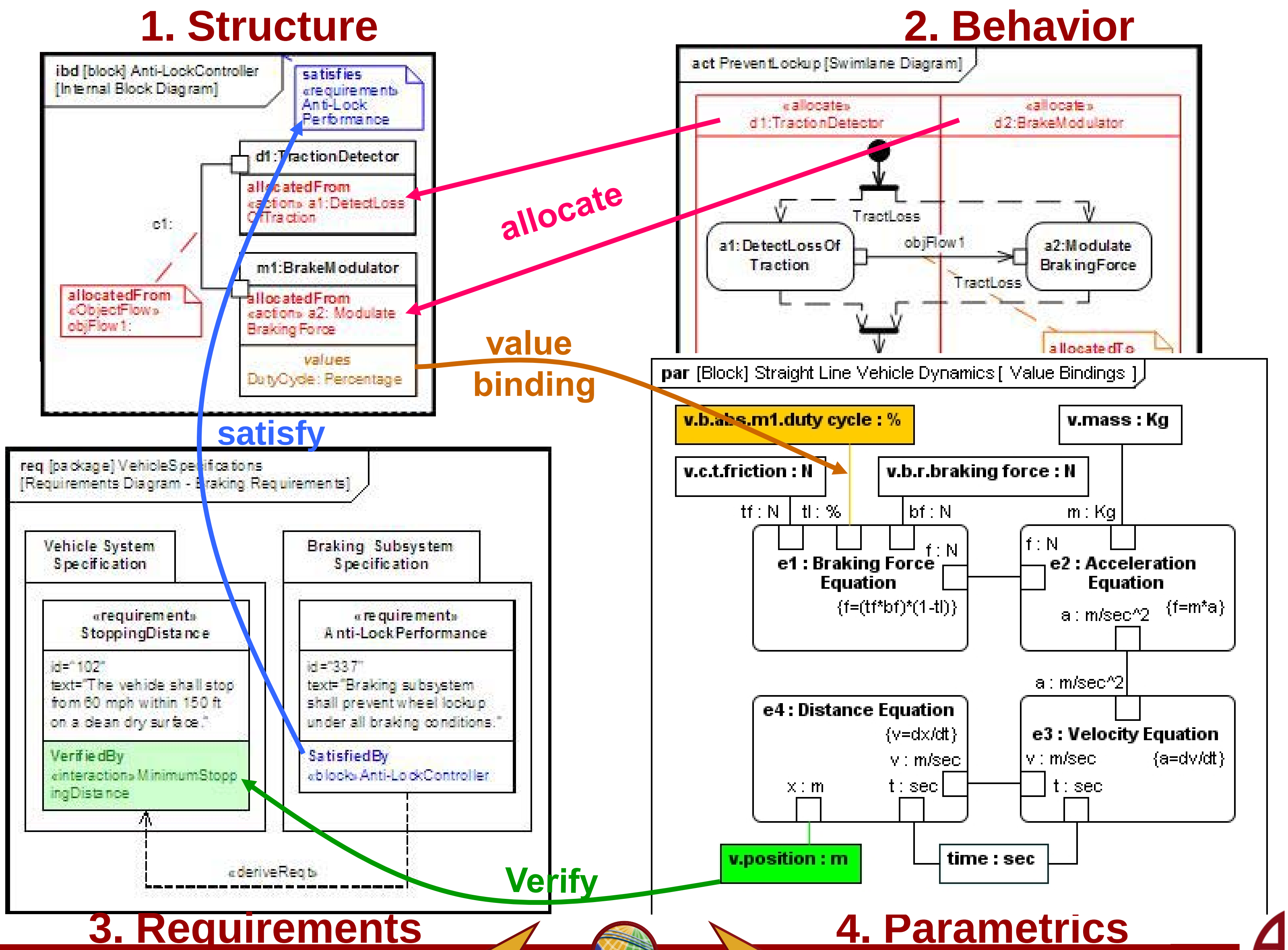
Relationship Matrix

Source: Activity Diagrams ... Type: Activity Link Type: Allocate Profile:

Target: Parking Garage ... Type: Block Direction: Source -> Targe Refresh Op

	Control Unit	Credit Card Authority	Credit Card Reader	Display Unit	Gate	Gate Assembly	Gate Motor	Magnetic Strip Reader	Parking Garage Gate Domain	Printer	Processor	Sensor	Ticket Reader	Vehicle
CalculateFee											↑			
CloseGate							↑							
DisplayInformation				↑										
OpenGate							↑							
PrintReceipt										↑				
ReadCreditCard			↑											
ReadTicket													↑	
SenseVehicle												↑		
VerifyCredit											↑			

Cross Connecting Model Elements



APL

Summary

- Allocations are used to depict mapping of model elements to one another
- There are many types of allocation, including: behavior, structure, and properties
- Allocations allows:
  - Allocating activities to blocks
  - Allocating requirements to blocks
  - Allocating logical elements to physical elements
- Allocation can be represented graphically though the following notations: Direct, Compartment, and Callout
- Tabular representations offer a compact representation of multiple allocation relationships

Class Exercise
Dishwasher Example - Sample Artifacts

Primary

- Requirement diagram – dishwasher spec
- Block definition diagram – top level
- Internal block diagram – dishwasher black box
- Use case diagram
- Activity diagram – black box scenario
- Block definition diagram – input/output definitions
- Block definition diagram – dishwasher hierarchy
- Internal block diagram – dishwasher white box
- Activity diagram – white box scenario
- Requirement diagram - traceability

Optional

- Parametric diagram
- State machine diagram
- Sequence diagram

PROCESS SUMMARY

System Modeling Activities – OOSEM
Integrating MBSE into the SE Process

Major SE Development Activities

Analyze Needs
- BDD – Top Level
- Mission Use Case Diagrams

Define System Requirements
- System Use Case Diagrams
- IBD – Black Box
- Activity Diagram – Black Box Scenario

Define Logical Architecture
- BDD – Input/Output definitions
- BDD – Hierachy
- IBD – White Box
- Activity Diagrams – White Box Scenarios

Synthesize Allocated Architecture
- Allocations

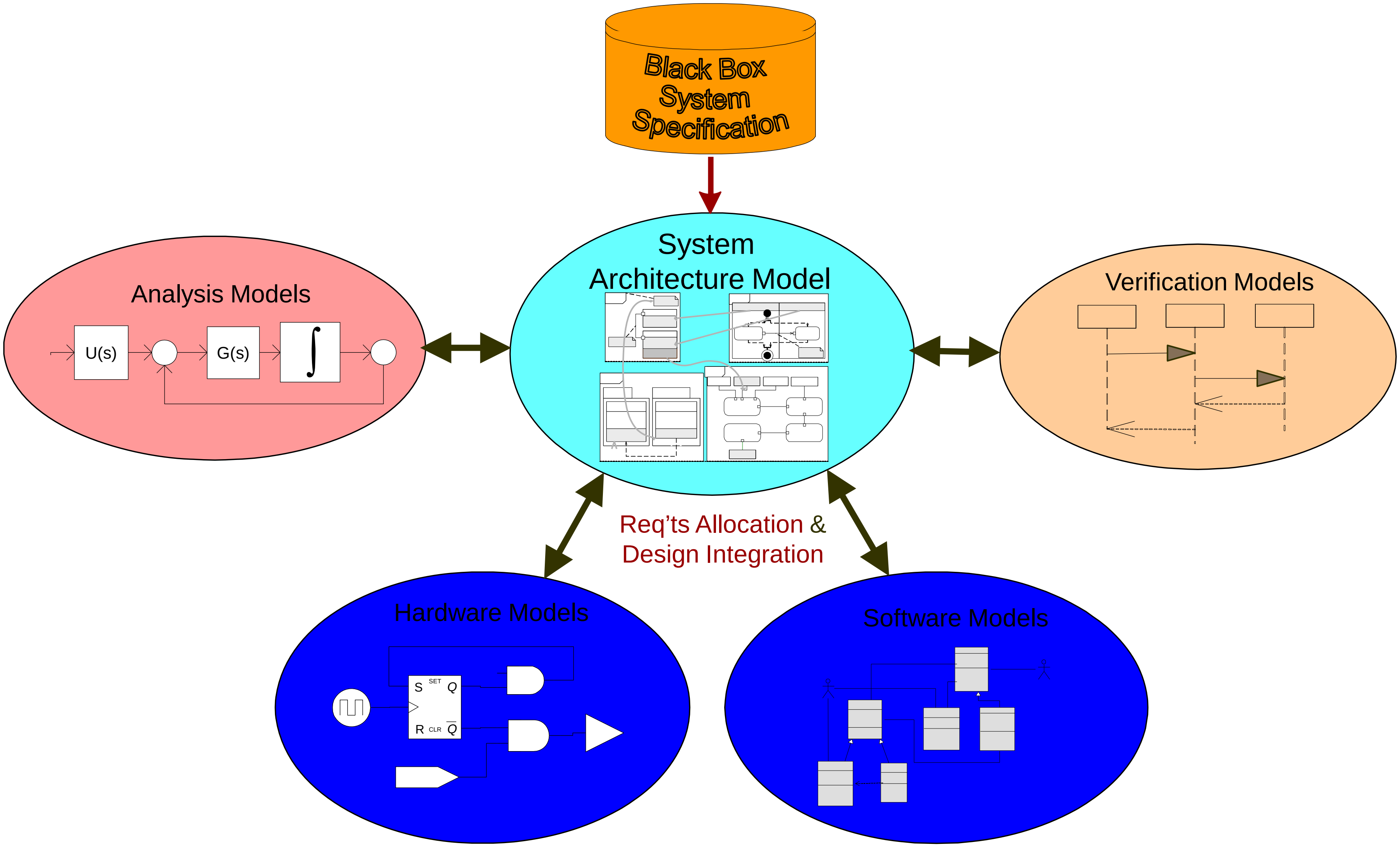
Optimize & Evaluate Alternatives
- Parametric Diagrams

Manage Requirements
- Reqt's Diagrams Spec & Traceability

Support Validation & Verification
- Test cases
- Test procedures

Common Subactivities

System Architecture Model Provides an Integration Framework



APL

TOOLS OVERVIEW

Tools Overview

- Tool Integration
- Suggested Tool Selection Criteria
- Partial List of SysML Tools

Tool Integration

- Classes of Tools in a Systems Development Environment
  - Project Management
  - Systems Modeling
  - Performance Simulation
  - Requirements Management
  - Configuration Management and Data Management
  - Verification and Validation
  - Engineering Analysis
  - HW and SW Modeling
  - Document Generation

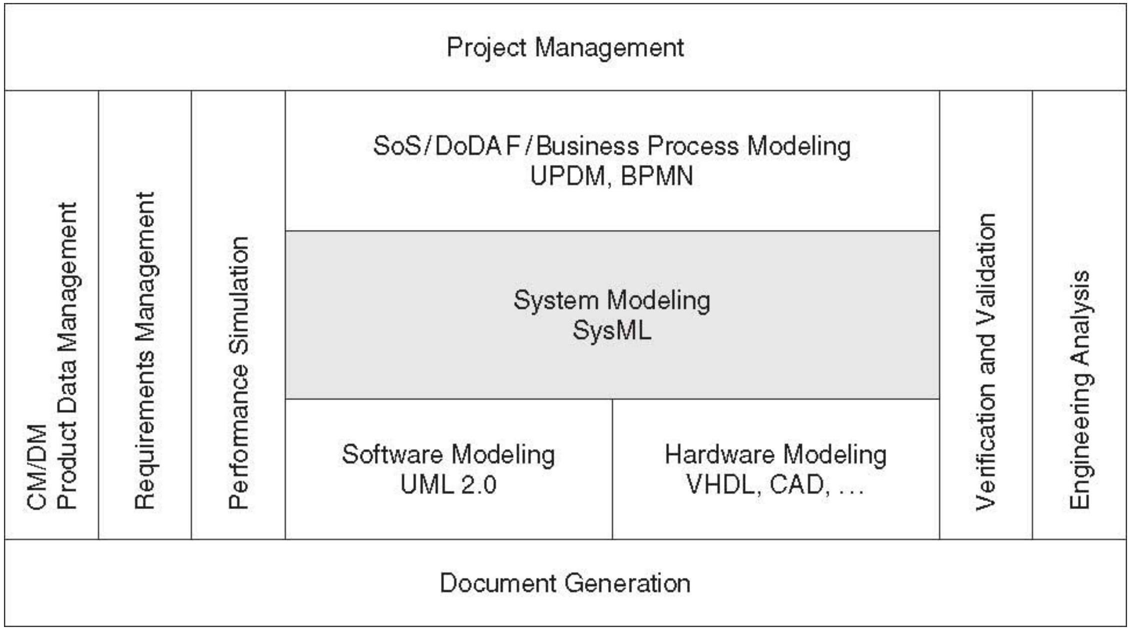


FIGURE 17.2

Tool Integration

- Data Exchange Mechanisms
  - Manual
  - File-based exchange (XMI)
  - Interaction-based exchange (API)
  - Repository-based exchange
- Data Exchange Standards
  - XML Metadata Interchange
  - Application Protocol 233
  - Diagram Interchange Standards
  - Model Transformation

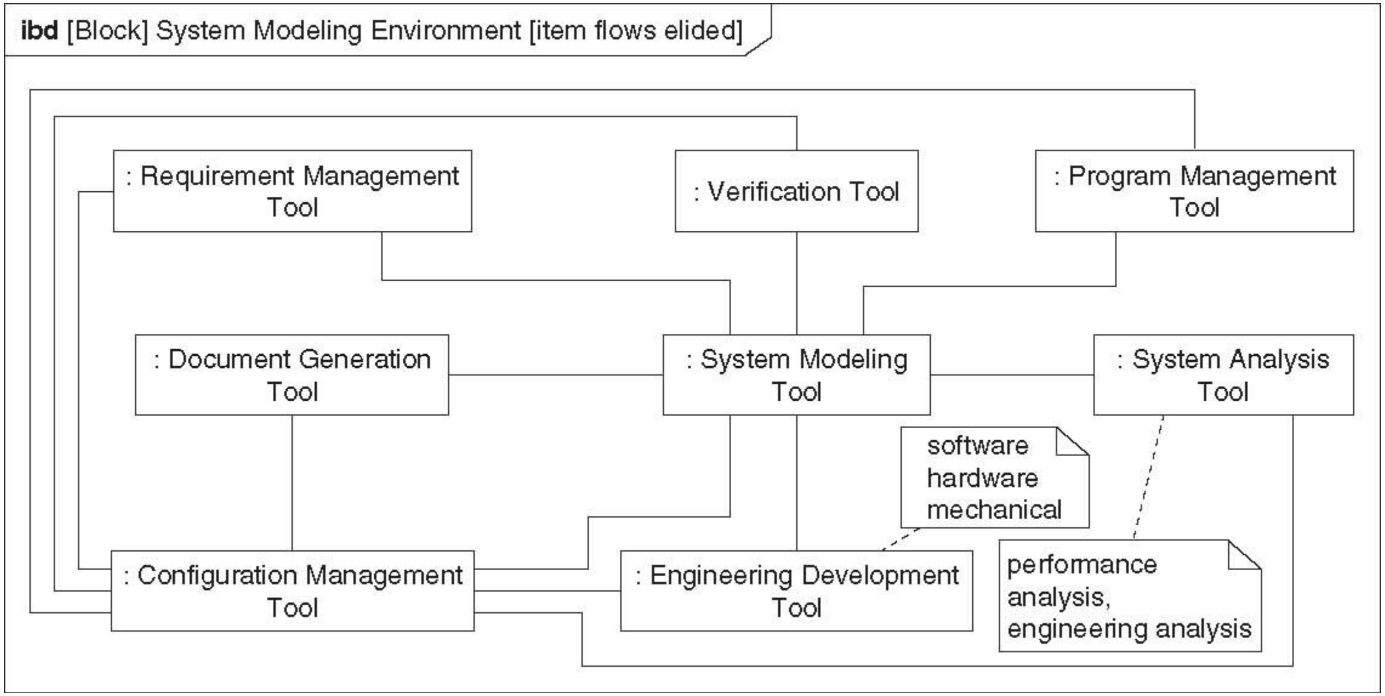


FIGURE 17.3

Suggested Tool Selection Criteria

- Conformance to SysML specification
- Usability
- Document Generation capability
- Model execution capability
- Conformance to XMI
- Conformance to AP233
- Integration with other engineering tools
- Performance (maximum number of users, model size)
- Model checking to verify model conformance
- Training, online help, and support
- Availability of model libraries
- Life-cycle cost (acquisition, training, support)
- Vendor viability
- Previous experience with tool
- Support for selected model-based method (e.g. automated scripts, standard reports, etc.)

Partial List of SysML Tools (taken from SysML RFI 2009 Survey Responses)

- IBM - Rhapsody
- No Magic - Magic Draw
- Sparx Systems - Enterprise Architect
- Artisan – Studio
- INTERCax ParaMagic (Magic Draw plug-in)
- Others
  - Microsoft Visio – SysML Template (Pavel Hruby)
  - ….

WRAP-UP

Deploying MBSE

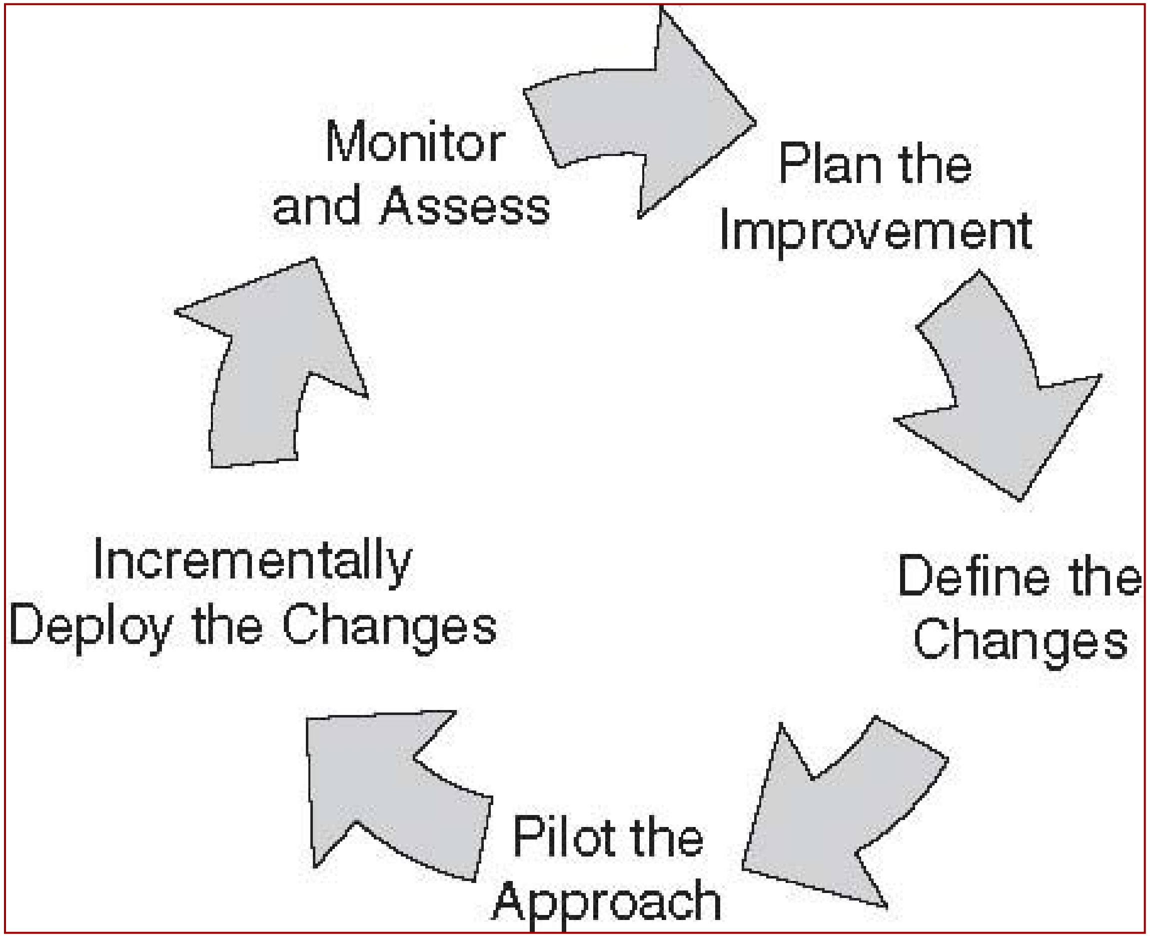


FIGURE 18.1 © 2008 Elsevier, Inc.: A Practical Guide to SysML

Deploy MBSE into your organization as part of your improvement process

Summary

- **SysML sponsored by INCOSE/OMG with broad industry and vendor participation and adopted in 2006**
- **SysML provides a general purpose modeling language to support specification, analysis, design and verification of complex systems**
  - **Subset of UML 2 with extensions**
  - **4 Pillars of SysML include modeling of requirements, behavior, structure, and parametrics**
- **Multiple vendor implementations available**
- **Standards based modeling approach for SE expected to improve communications, tool interoperability, and design quality**
- **Plan SysML transition as part of overall MBSE approach**
- **Continue to evolve SysML based on user/vendor/researcher feedback and lessons learned**