



**ENES 489P Hands-On Systems Engineering Projects**

***Introduction to UML and SysML***

Mark Austin

E-mail: `austin@isr.umd.edu`

Institute for Systems Research, University of Maryland, College Park

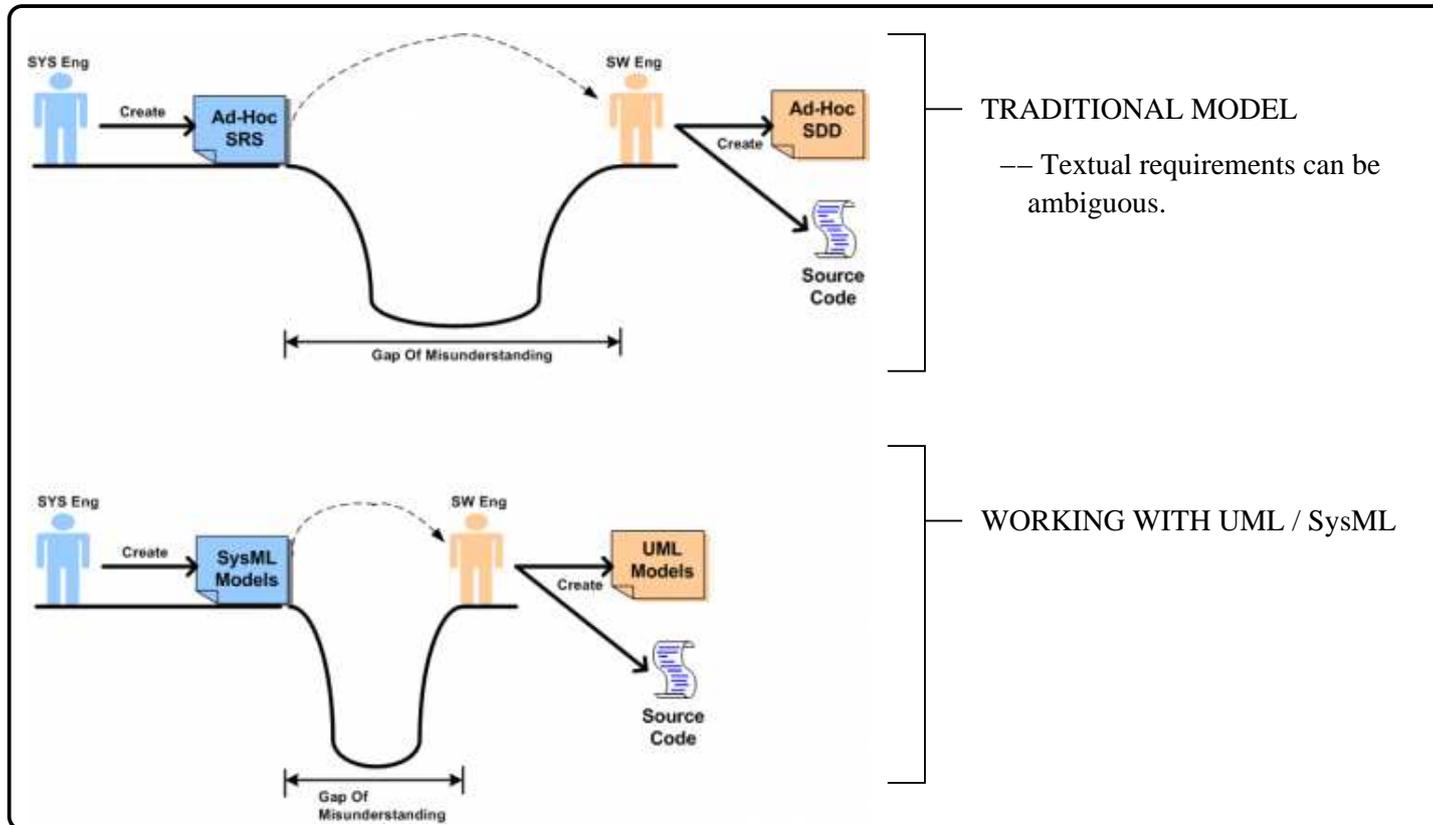
# Systems Engineering with UML and SysML

## Topics:

1. Motivation and Approach
2. What are UML and SysML?
3. System Development Processes
4. System Architecture View (new to UML 2.0)
5. Behavior Modeling with Activity and Sequence Diagrams
6. Finite State Machines and Statecharts
7. Case Study: Operation of a Museum
8. Case Study: Operation of a Traffic Intersection

# Need for Visual Modeling Languages

## Motivation – What’s wrong with the traditional way of doing things?



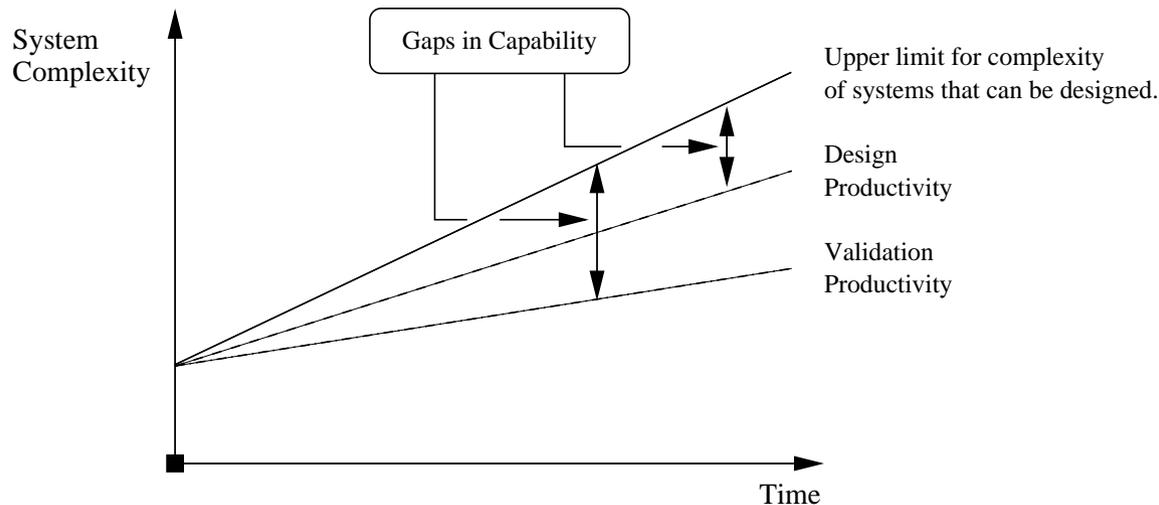
Source: Adapted from <http://bulldozer00.wordpress.com/uml-and-sysml/>

# Need for Visual Modeling Languages

## Motivation – Looking Ahead

Because future engineering systems will be more complex than today, designers will need to be more productive ...

**... just to keep the duration and economics of design development in check.**



Future designs will also need to be more agile than in the past.

# Need for Visual Modeling Languages

## Finding a way to Move Forward

The pathway forward can be found by looking to the past, where ...

**... major increases in designer productivity have nearly always been accompanied by new methods for solving problems at higher levels of abstraction.**

Therefore, we seek ...

**... new ways of working at higher levels of abstraction,**

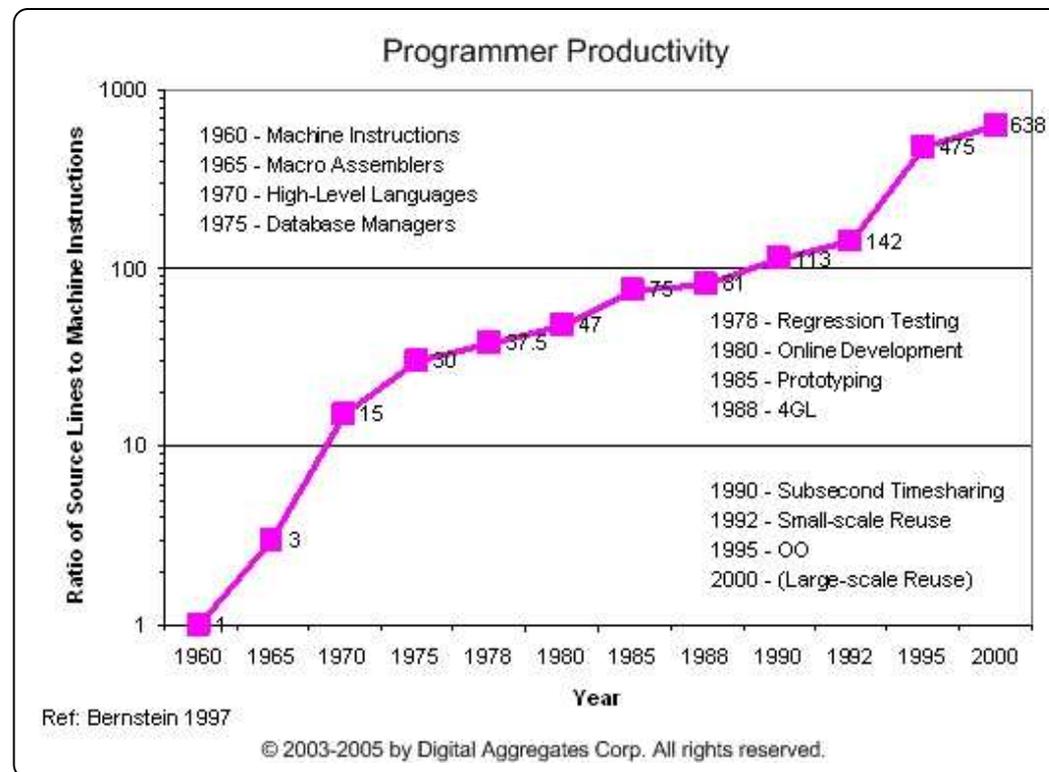
and

**... maximizing opportunities for adaptation by delaying decisions on implementation for as long as possible.**

# Visual Modeling Languages

## Example. Evolution of Abstractions in Software Development

Machine code, assembly language, high-level languages (e.g., Fortran), object-oriented programming (e.g., Java), scripting languages (e.g., Python).



# Visual Modeling Languages

## Visual Modeling Formalisms

Visual modeling formalisms ...

**... map real-world products and processes to a graphical representation or blueprint. These formalisms use symbols to represent real-world entities.**

## Appropriate Formalisms for Engineering Development

Task	Modeling Formalism
Architecture and Design	Visual Modeling Languages
Calculations	Algebra
Algorithms	Programming Languages

The development of real-world engineering systems is complicated by a need to ...

**... satisfy physical constraints on behavior.**

# Visual Modeling with UML and SysML

## Goals of UML?

The goals of the Unified Modeling Language (UML) and the System Modeling Language (SysML) are (Rational, 1997)..

**... to provide users with a ready-to-use, expressive visual modeling language (notation) so they can describe and exchange meaningful models.**

## Uses of UML?

Most engineers use UML informally, ...

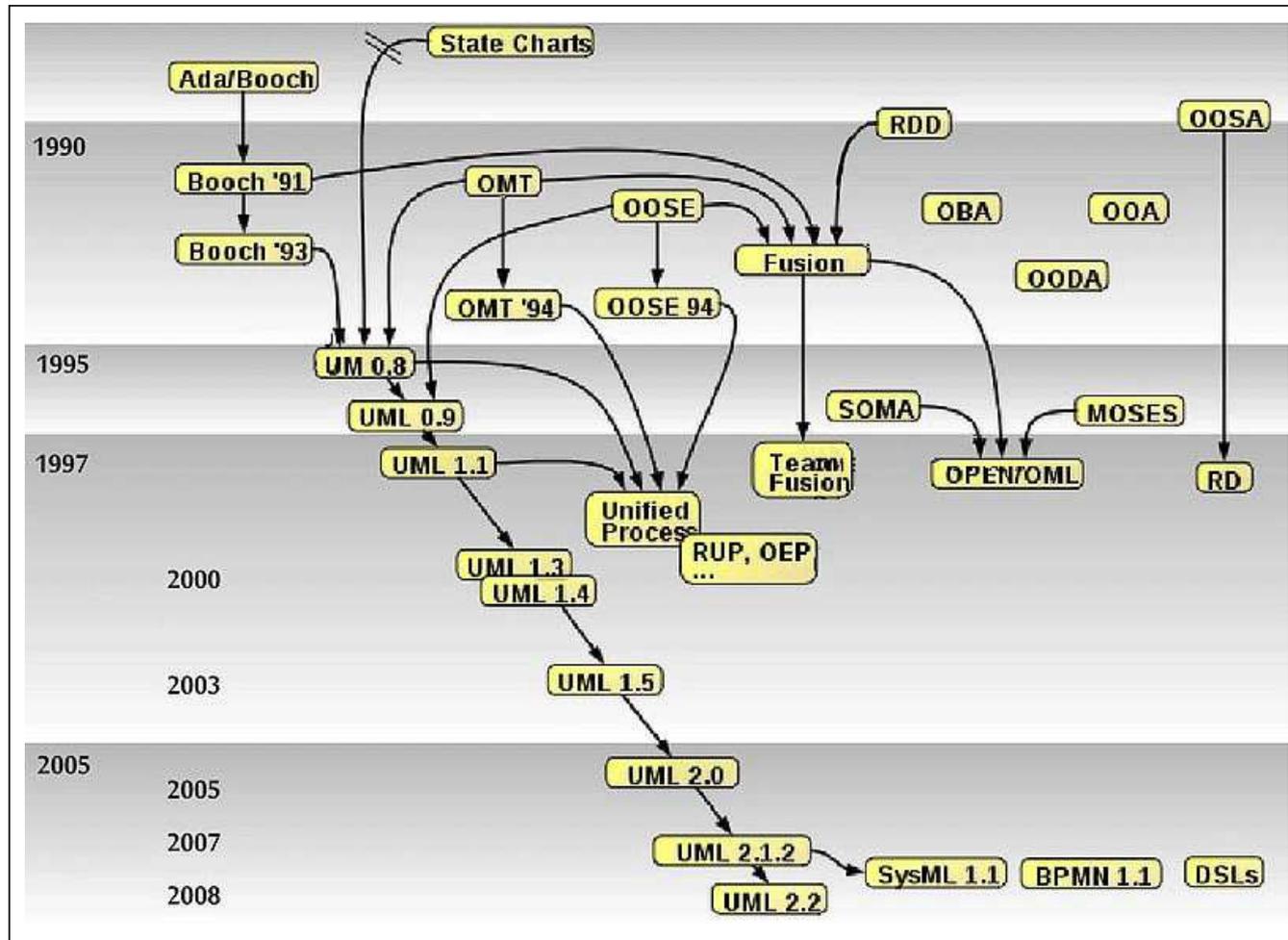
**... that is, diagrams are sketched as abstractions of a system description.**

Semi-informal uses of UML aim to create ...

**... a one-to-one correspondence between UML and the system being described.**

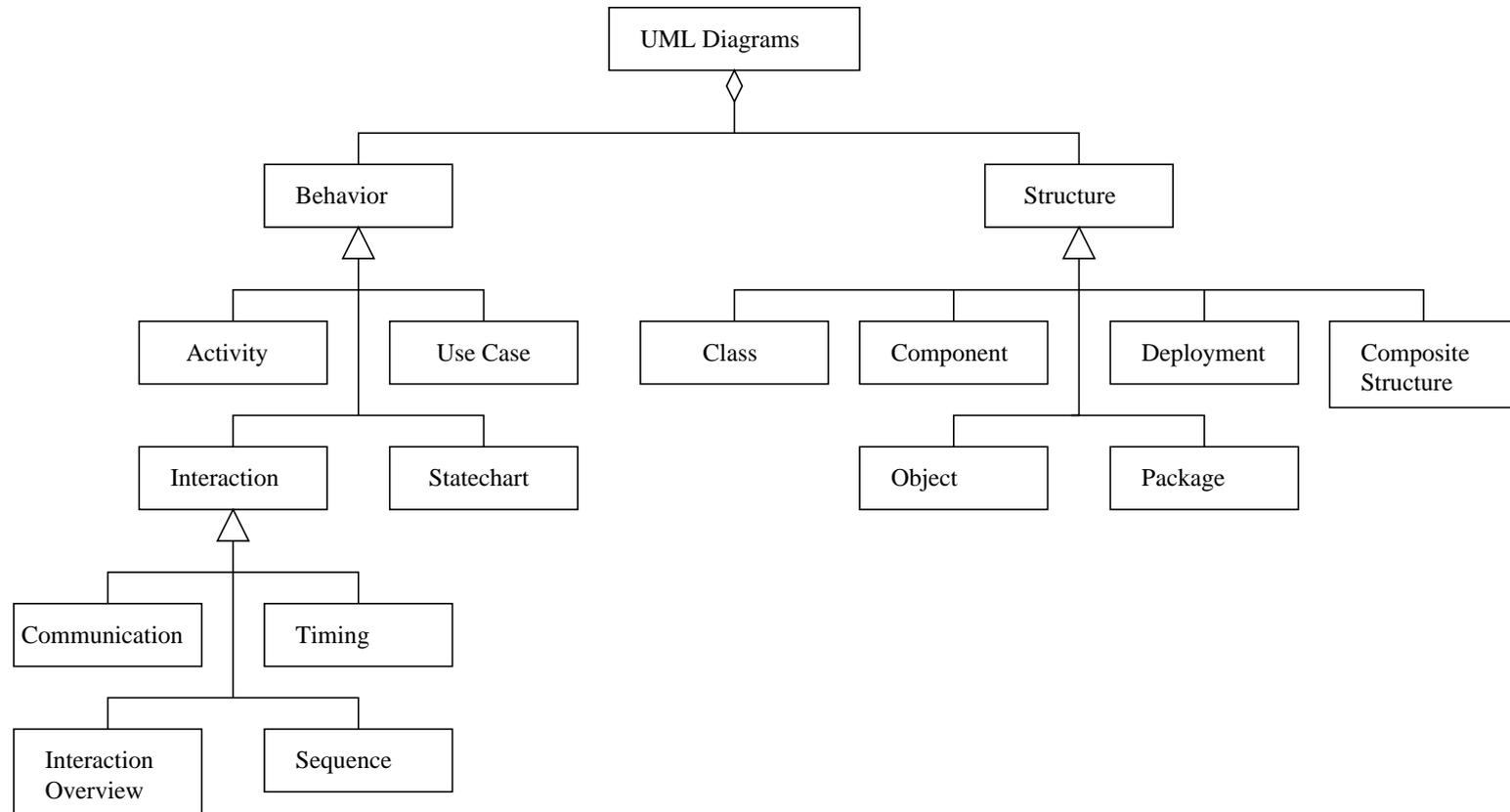
# Visual Modeling with UML and SysML

## History of UML?



# High-Level View of UML

## Taxonomy of Diagram Types in UML 2



# Visual Modeling with UML

## Taxonomy of Diagrams in SysML

---

### Structure Diagrams

Block Diagram

Block Definition Diagram

(extends UML Class Diagram)

Internal Block Diagram

(extends UML Composite

Structure Diagram)

Parametric Constraint Diagram

Parametric Definition Diagram

Parametric Use Diagram

### Behavior Diagrams

Activity Diagram

(extends UML Activity Diagram)

Use Case Diagram

State Machine Diagram

Sequence Diagram

### Cross-Cutting Diagrams

Allocation Diagram

Package Diagram

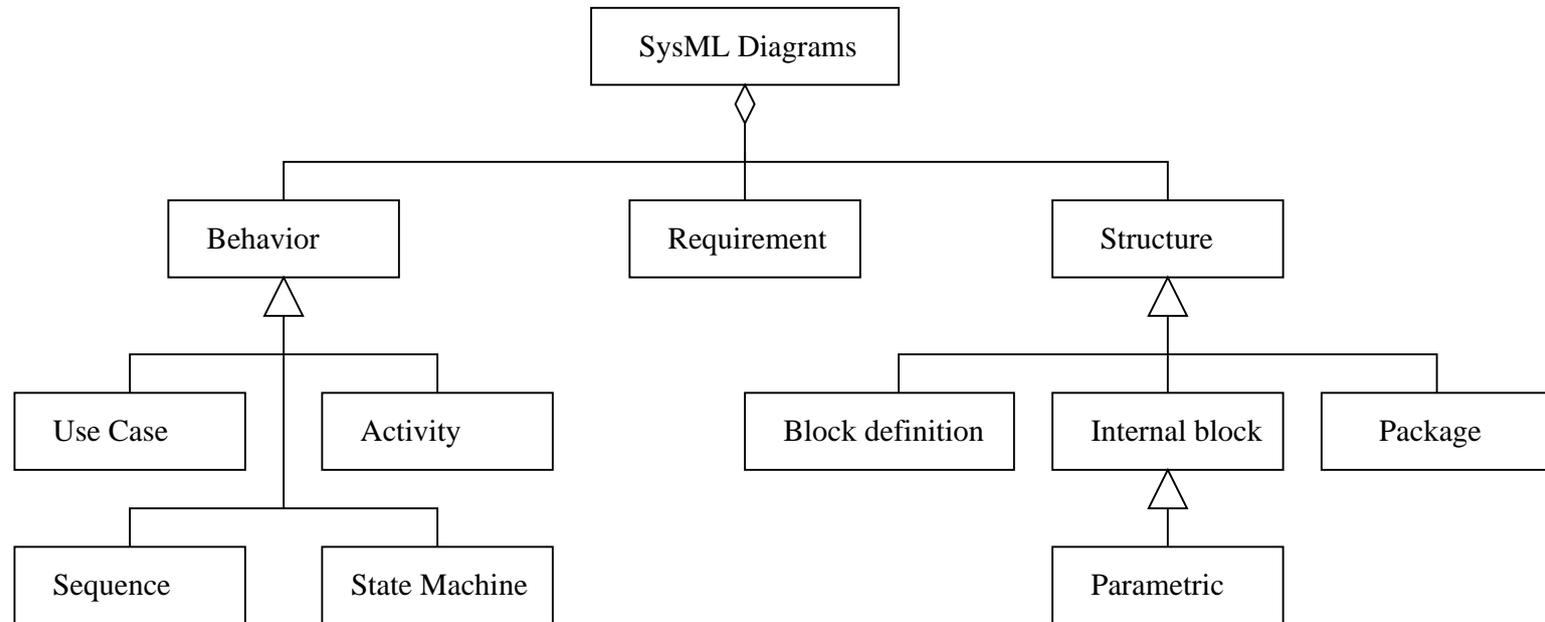
(extends UML Package Diagram)

Requirement Diagram

---

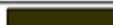
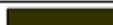
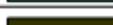
# Visual Modeling with SysML

## Taxonomy of Diagrams in SysML (Source: Adapted from SysML tutorial)

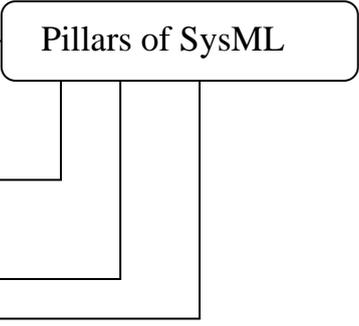


# Visual Modeling with SysML

## Side-by-side comparison on UML and SysML

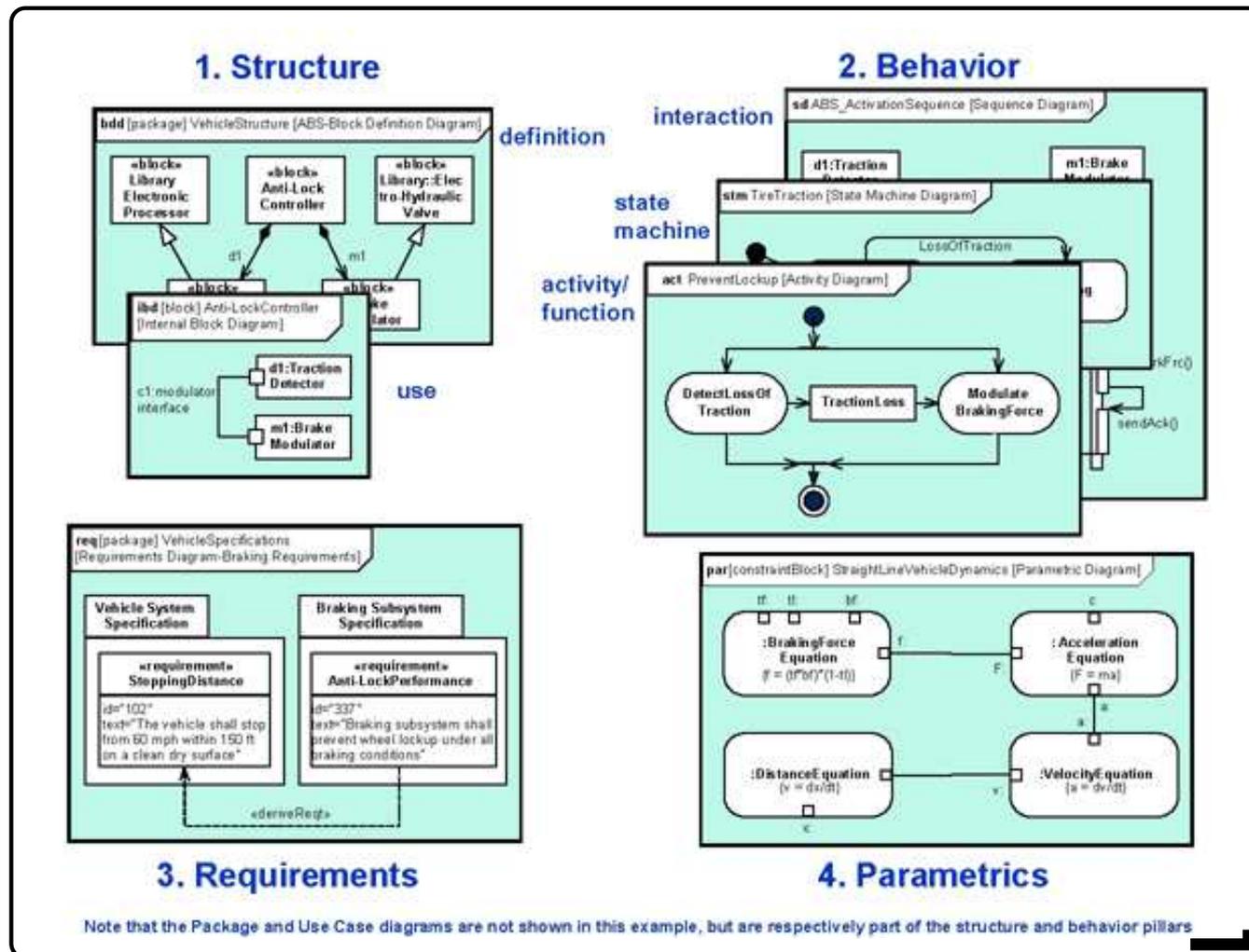
DIAGRAM	UML	SysML
Block Definition Diagram		
Internal Block Diagram		
Use Case Diagram		
Sequence Diagram		
Activity Diagram		
State Machine Diagram		
Parameteric Diagram		
Requirement Diagram		
Class Diagram		
Package Diagram		
Component Diagram		
Composite Structure Diagram		
Deployment Diagram		
Object Diagram		
Interaction Overview Diagram		
Communication Diagram		
Timing Diagram		

Pillars of SysML



# Visual Modeling with SysML

## Pillars of SysML



# Visual Modeling with SysML

## SysML Requirements Diagram

Requirements diagram notation:

- Provides a means to show the relationships among requirements including constraints.
- Shows how requirements relate to other model elements.
- Relationships among requirements can be used to define a requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements, and refining requirements. [SysML 08]"

Therefore, the requirement diagram provides ...

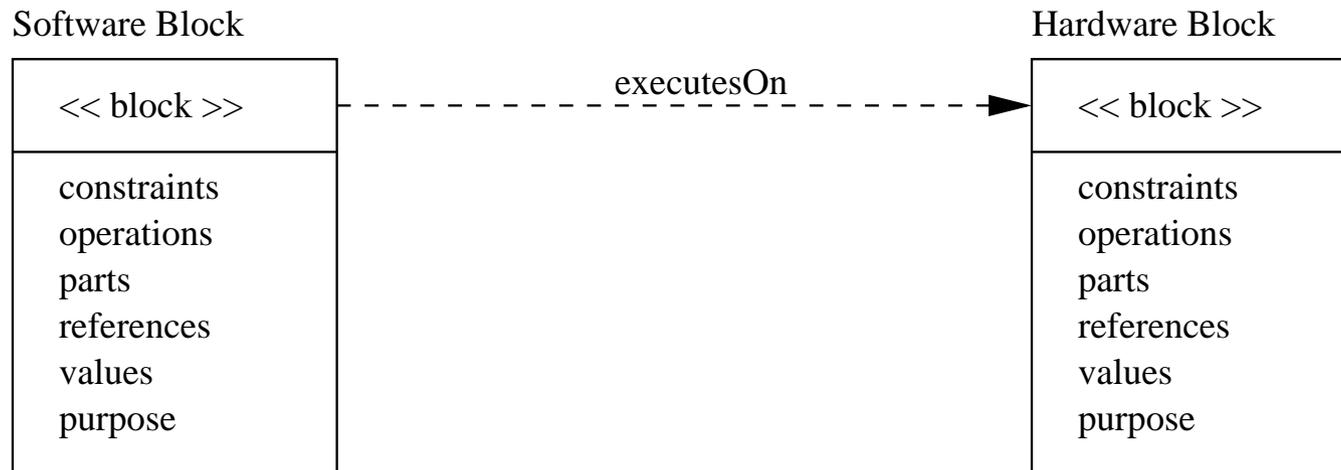
**... a bridge between the typical requirements management tools and the system models.**

# Visual Modeling with SysML

## SysML Block Definition Diagram

- This diagram is used to show features and relationships at a high-level of abstraction, even before decisions on technology/implementation have been made.
- Block diagrams present blocks that can represent hardware or software or even a combined hardware/software unit.

### Example. Block Definition for Software-Hardware Dependency



# Visual Modeling with SysML

## SysML Parameteric Diagram

Parametric diagrams represent ...

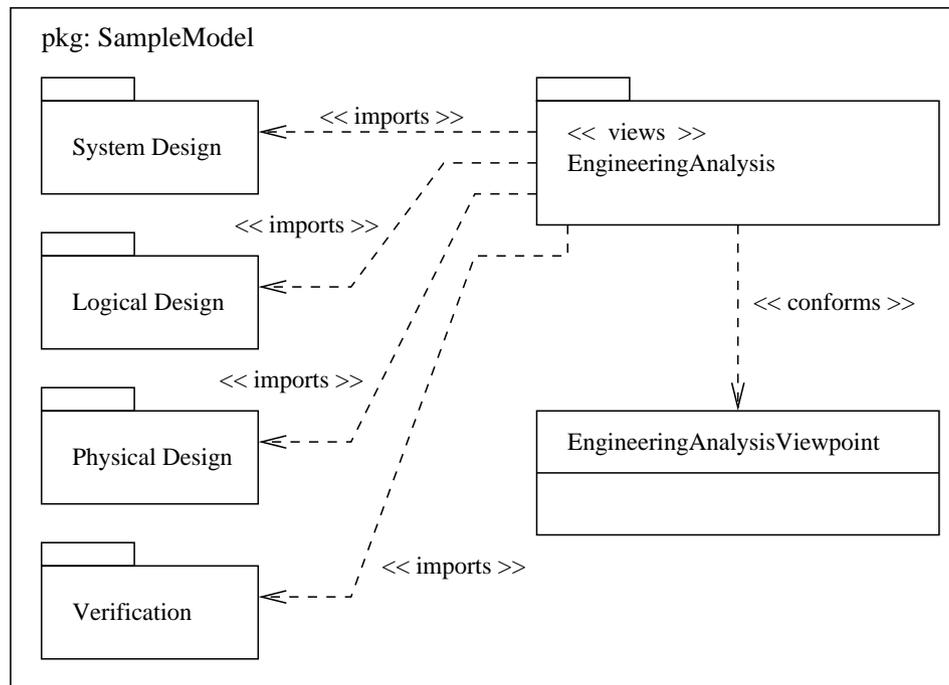
**... constraints on system property values such as performance, reliability, and mass properties.**

As such, they provide ...

**... a means for specification and design models to be integrated with engineering analysis models.**

# Visual Modeling with SysML

## SysML Package Diagram



SysML support for package diagram - views. Views and viewpoints are consistent with IEEE 1471 definitions. As such, views conform to a particular viewpoint which, in turn, imports model elements from multiple packages.

# Visual Modeling with SysML

**Compared to UML, SysML offers the following benefits:**

- **Block Stereotypes**

The SysML Block Stereotype is based on the UML concept of composite structures. Blocks can have internal features (attributes, operations) and can own ports.

The extension of UML ports in SysML as flowports provides a far more complete system model in which blocks can be connected (physically and/or logically) to other blocks.

- **Allocations**

SysML extends the UML trace comment with their new allocation property.

Functional allocation is the assignment of functions (requirements, specifications, behaviors, etc.) to system components.

Support for functional allocations is needed especially in the development of larger systems where design and implementation may not occur at the same place or time.

UML versions 1 and 2 make little reference to functional allocation (aside from swimlanes in an Activity diagram).

# Visual Modeling with SysML

**Compared to UML, SysML offers the following benefits:**

- **Requirements Modeling**

SysML provides modeling constructs to represent requirements and relate them to other modeling [system] elements.

SysML introduces an actual requirements node which contains information about requirements such as identifier, text, source, and method of verification.

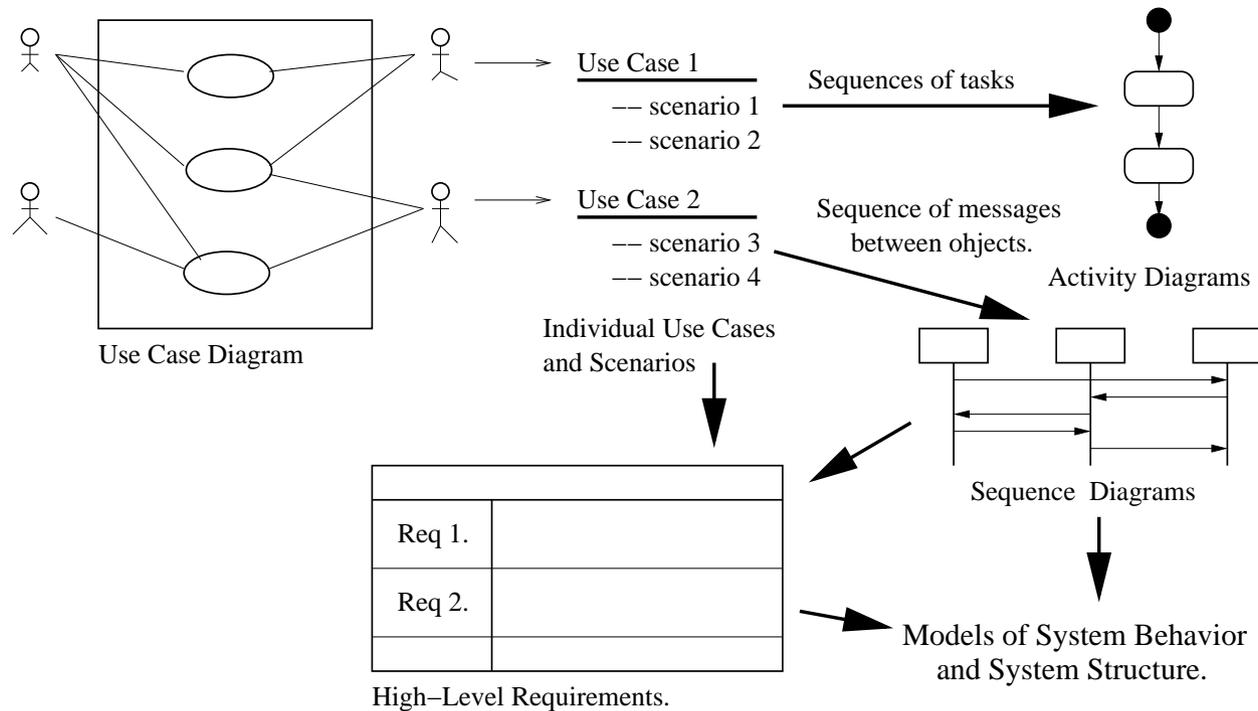
These requirements nodes can be used in Block Definition Diagrams (SysML version of a UML class diagram) to show a hierarchy of requirements.

Requirements can also be mapped to other elements by derivation, verification, and satisfaction paths (e.g., a diagram can show how a specific requirement is assigned to a component in the system structure.)

# System Development Processes

## Goals, Scenarios, Use Cases and Requirements

Pathway from operations concept to simplified models for behavior and structure, to requirements....



# System Development Processes

## Key Points

1. The functional description dictates what the system must do.  
Here, we employ a combination of use cases (and use case diagrams), textual scenarios, and activity and sequence diagrams to elicit and represent the required system functionality.
2. Activity diagrams show system functionality/behavior as a directed graph.  
Hence, they aid in the generation of functional requirements (i.e., tasks that must be supported by the system) and non-functional requirements (e.g., safety).
3. Sequence diagrams show the flow of messages between objects. Thus, they lead to requirements on elements that must exist in the system structure, as well as interface requirements.

# System Development Processes

## Key Points (cont'd)

4. A complete system description will also include statements on minimum levels of acceptable performance and maximum cost.

Since a system does not actually exist at this point, ...

**... these aspects of the problem description will be written as design requirements/constraints.**

5. Further design requirements/constraints will be obtained from the structure and communication of objects in the models for system functionality (e.g., required system interfaces).

# Guidelines for using Visual Representations

Activity	Visualization
Use cases	<ul style="list-style-type: none"><li>● Use case diagrams.</li><li>● Use case diagrams with generalization relationships.</li></ul>
Scenarios	<ul style="list-style-type: none"><li>● Text for linear sequences of tasks.</li><li>● Activity diagrams for fragments of behavior.</li><li>● Sequence diagrams for message passing between objects.</li></ul>
Requirements	<ul style="list-style-type: none"><li>● Table format for textual description of requirements.</li><li>● Trees and graphs for requirements emanating from single/multiple sources.</li><li>● Requirements diagram (SysML).</li></ul>

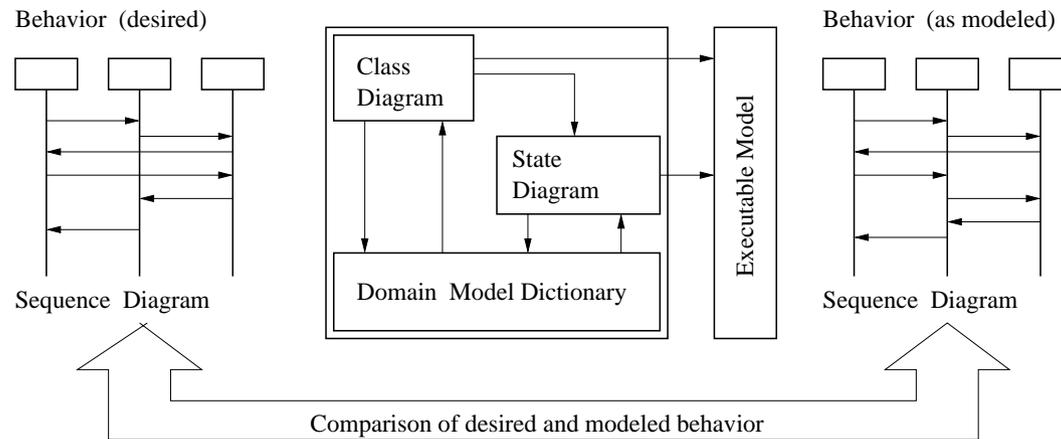
# Guidelines for using Visual Representations

Activity	Visualization
<b>Behavior</b>	<ul style="list-style-type: none"><li>● Activity diagrams only containing task nodes.</li></ul>
<b>Structure</b>	<ul style="list-style-type: none"><li>● Class diagrams (UML).</li><li>● Composite structure diagram (new to UML 2).</li><li>● Block diagram (SysML).</li></ul>
<b>System Design</b>	<ul style="list-style-type: none"><li>● Activity diagrams containing combinations of tasks and state nodes.</li><li>● State machine viewpoint: statechart diagrams.</li><li>● Interaction viewpoint: sequence and collaboration diagrams.</li></ul>

# System Development Processes

## Evaluation of System Behavior

We need to make sure that the desired model of system behavior matches the implemented model of system behavior.



At the end of each interaction, the "desired" and "as modeled" behaviors are compared.

- When the comparison is good, we can proceed to the next (lower) level of object decomposition and/or to modeling of a new behavior (i.e., use case).
- When the comparison is bad (or insufficient), the object classes and their state diagrams need to re-engineered.

# System Architecture View

## Block Diagrams/Shortcomings of Class Diagrams

In UML (version 1), class diagrams have been the principal mechanism for expressing system structures.

As we move toward the representation and analysis of real-time/embedded systems a fundamental question is:

**...are class diagrams still sufficient?**

NO! Because classes abstract out certain specifics, class diagrams are not suitable for performance analysis.

UML 2 overcomes this shortcoming. System structures are captured by ...

**... class, composite structure, component, deployment, object, and package diagrams.**

# System Architecture View

## Objects and Class Notation

Examples of object instance names....

Object Name

Object Name : Class Name

: Class Name

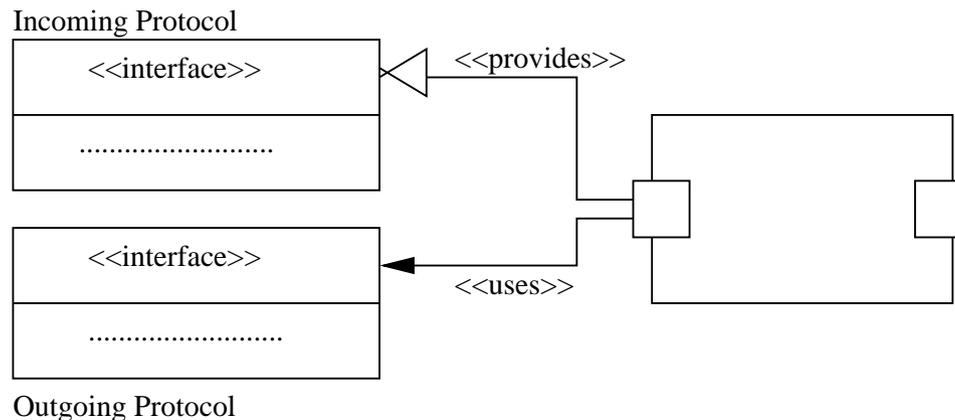
## Summary of the syntax ...

Syntax	Description
o	An object named o.
o : C	An object named o from class C.
: C	An anonymous object from class C.
/R	An anonymous object playing the role R.
/R : C	An anonymous object of class C playing the role R.
o / R	An object o playing the role R.
o / R : C	An object o from class C, playing the role R.

# System Architecture View

## Classes with Ports and Interfaces Port semantics.

Classes can now be modeled with ports and interfaces, supported by stereotypes (<<provides>> and <<uses>>) to support type-checking, and pre- and post-conditions.



Two kinds of interfaces for classes:

- Provided interfaces describe the services that a class implements.
- Required interfaces describe the services that other classes must provide before the class can operate properly in a given environment.

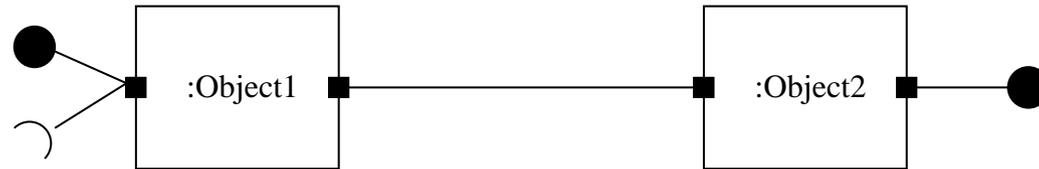
# System Architecture View

## Composite Structure Diagrams

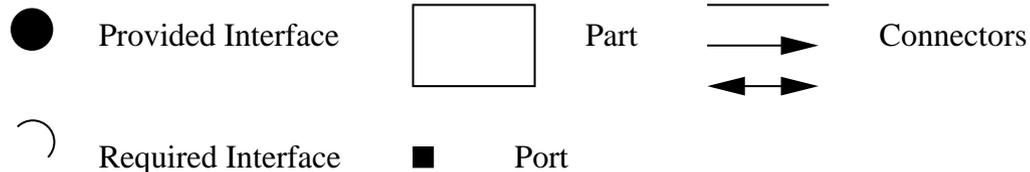
Composite structure diagrams present ...

**... the internal structure of a classifier (such as a class or component) in terms of collaborative parts (sets of instances), how they interact together and how they communicate with their container through ports, interfaces and connectors.**

## Assembly of Communicating Objects

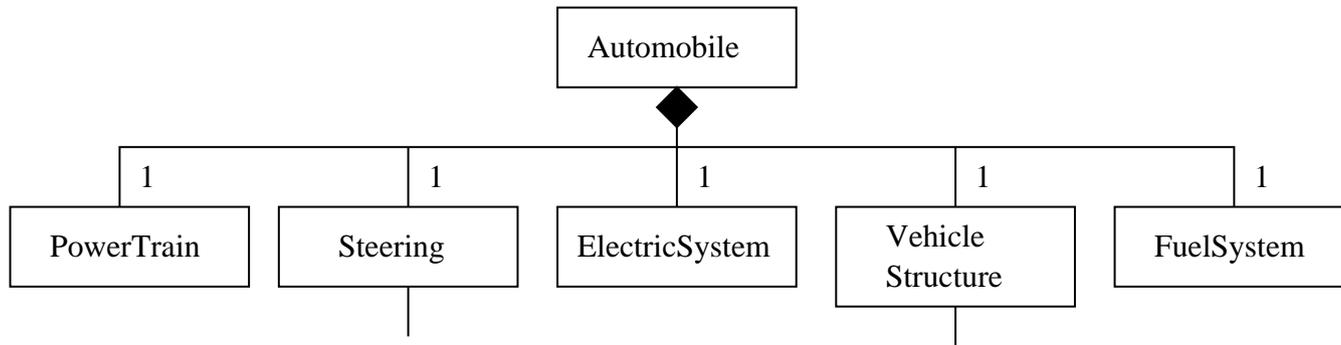


### Glossary



# System Architecture View

## Example. Compositional Hierarchy for an Automobile



The Automobile class is a composition of ...

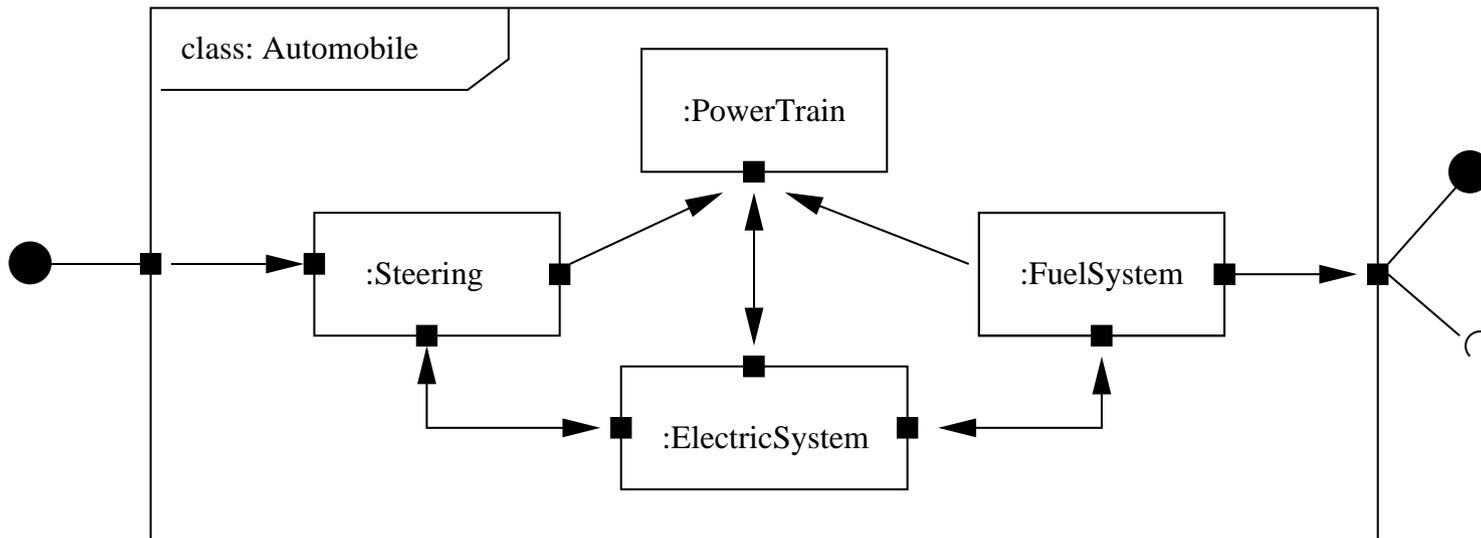
**... PowerTrain, Steering, ElectricSystem, VehicleSystem and FuelSystem classes.**

For the sake of brevity, lower-level details of the class hierarchy have been omitted.

# System Architecture View

## White- and Black-box Modeling for an Automobile

(a) White-box view of the Automobile class.



(b) Black-box view of the Automobile class.



# Behavior Modeling with Activity Diagrams

## Definition

Activity diagrams document sequences of tasks making up a single activity.

They are especially useful for:

**... for activities governed by conditional logic, and flows driven by internal processing (as opposed to external events).**

Hence, activity diagrams are appropriate for situations where ...

**... all (or most) of the events represent the completion of internally-generated actions and/or where asynchronous events occur.**

## Format for Activities and States

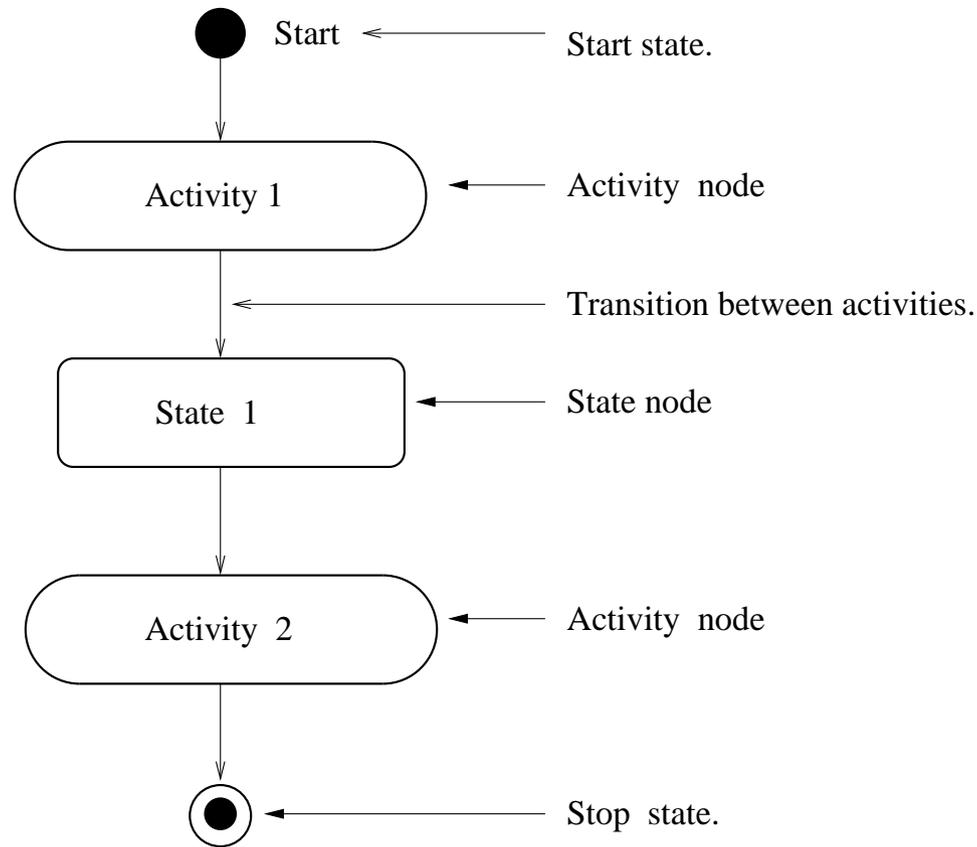
Activity Name

State Name

A state in an activity diagram is a point where some event needs to take place before an activity can continue.

# Behavior Modeling with Activity Diagrams

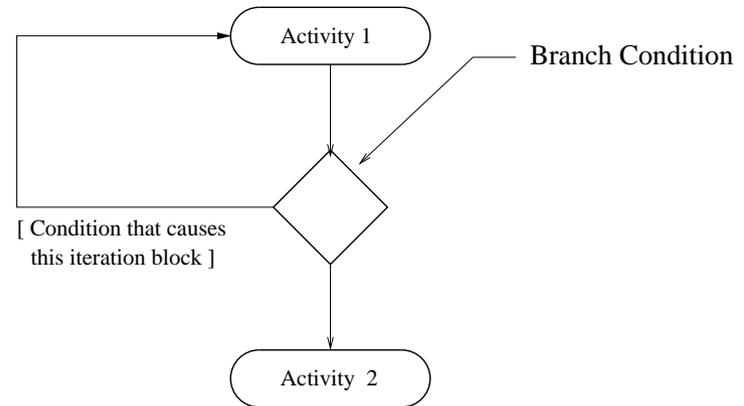
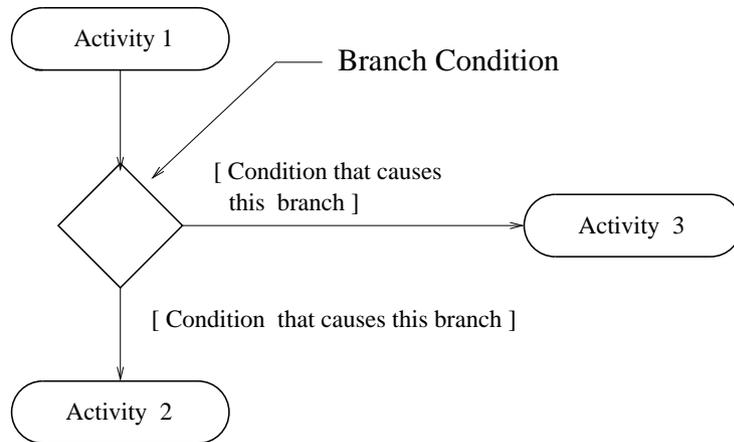
## Example 1. Main elements of an activity diagram



# Behavior Modeling with Activity Diagrams

## Use of Branching and Looping Constructs

Activity diagrams may express a decision point where the evaluated value of a guard condition determines the pathway of execution.

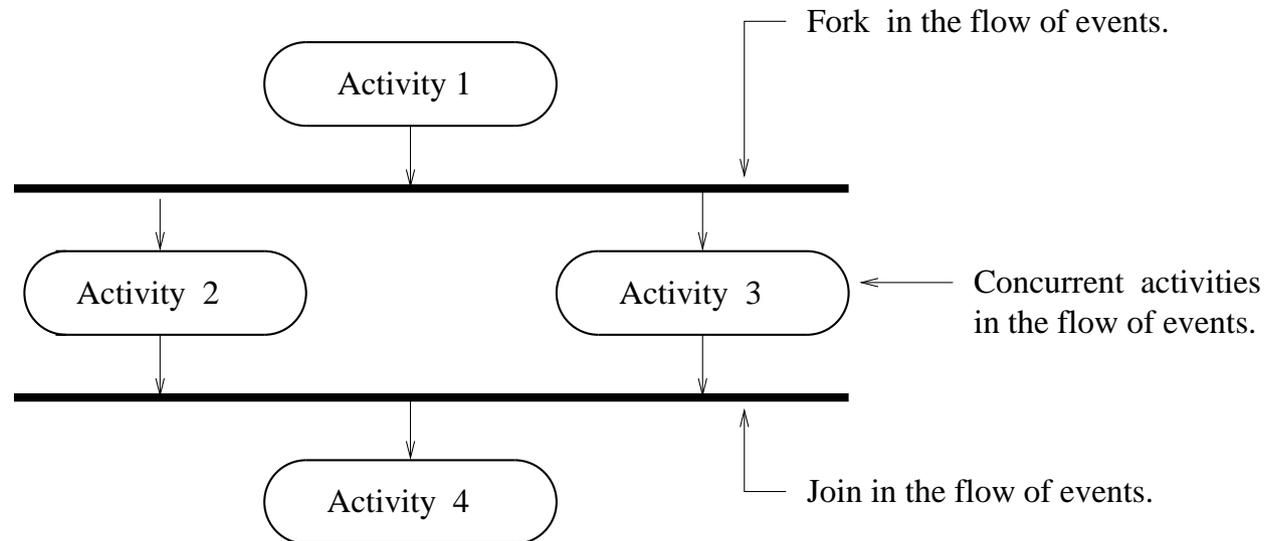


Diamond shapes represent transitions to different branches in an activity diagram.

# Behavior Modeling with Activity Diagrams

## Constructs for Synchronization

Synchronization bars give activity diagrams the ability to model flows of event that are concurrent.

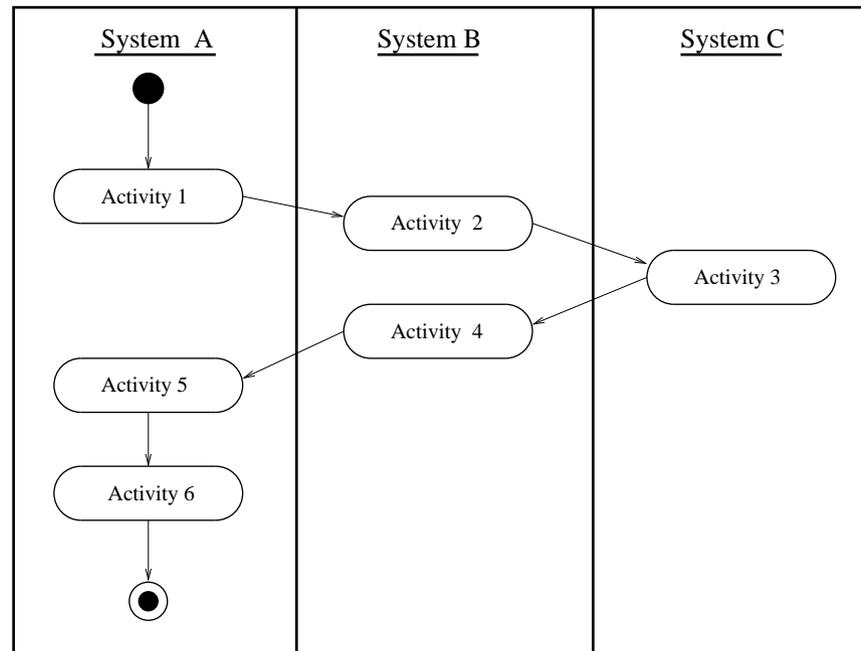


# Behavior Modeling with Activity Diagrams

## Swimlanes

Swim lanes are ...

**... a notation for indicating where an activity takes place (e.g., in a business).**

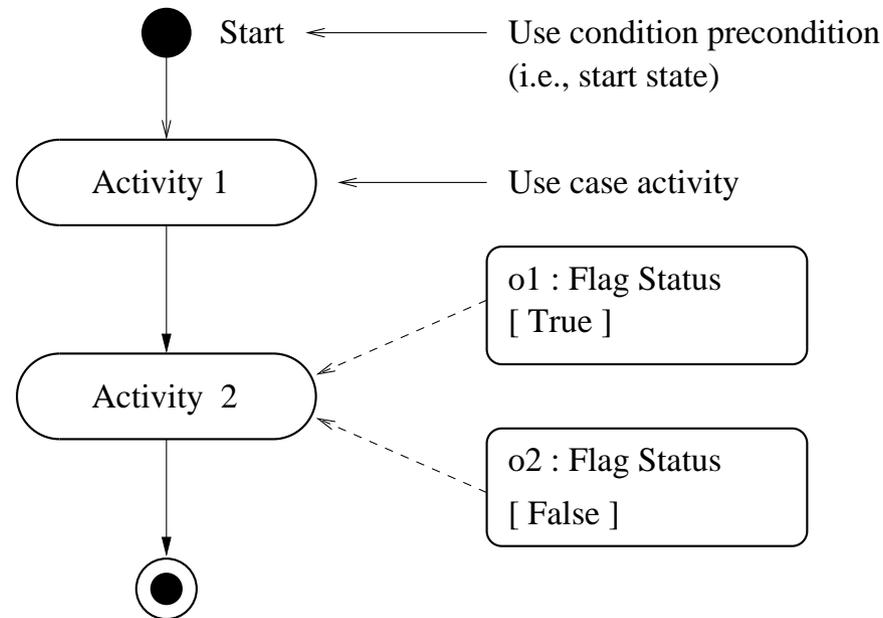


Swim lanes are defined by columns in an activity diagram (e.g., participating actors), and activities in the diagram are organized into swim lanes.

# Behavior Modeling with Activity Diagrams

## Displaying Objects on Activity Diagrams

Sometimes it is useful to indicate on an activity diagram how a flow of work affects an object.



Rectangular boxes contain the object name and the state of appropriate variables that are the result of the work flow.

# Behavior Modeling with Sequence Diagrams

## Purpose

A sequence diagram presents an interaction (i.e., a flow of messages) between objects to achieve a desired operation or result.

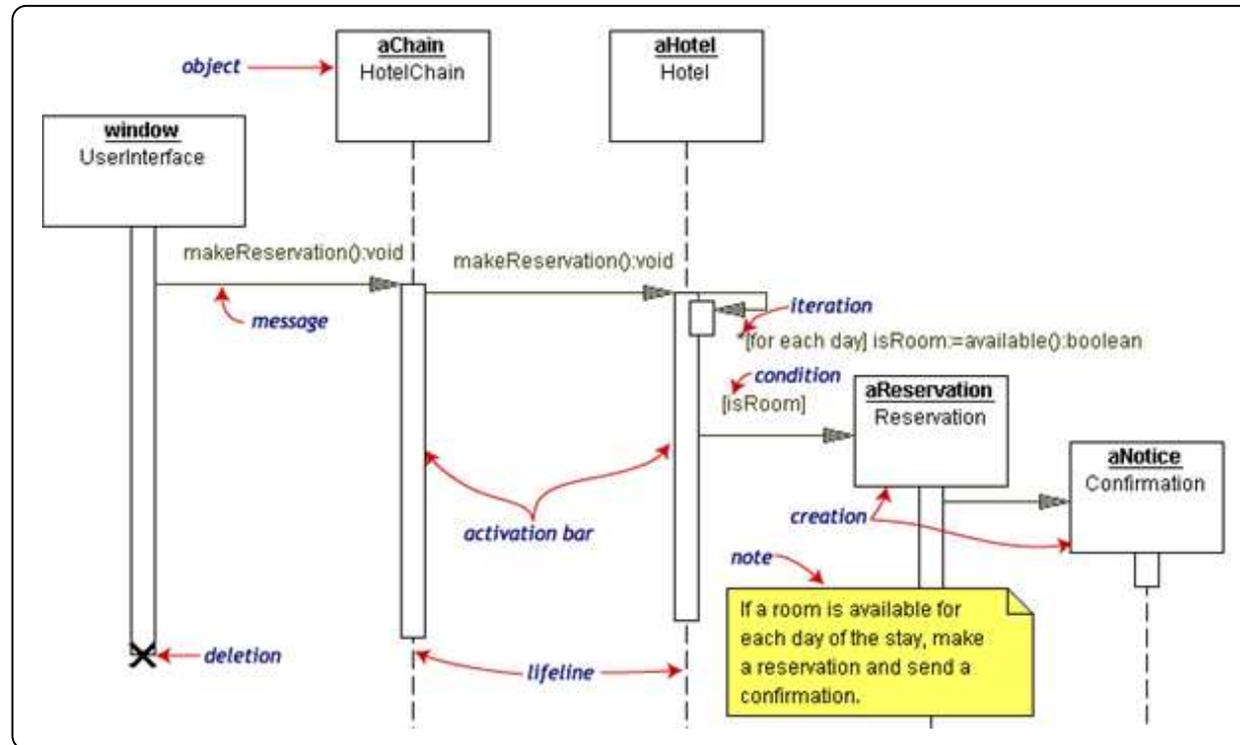
**...sequence diagrams are an appropriate form of visualization after the "system objects" have been identified.**

Sequence diagrams enable designers to perform three key tasks:

- Allocate behavior among boundary objects, entity objects and controllers that will become full objects in the system model.
- Show the detailed interactions that occur over time among the objects associated with each use case.
- Finalize the distribution of operations among classes.

# Behavior Modeling with Sequence Diagrams

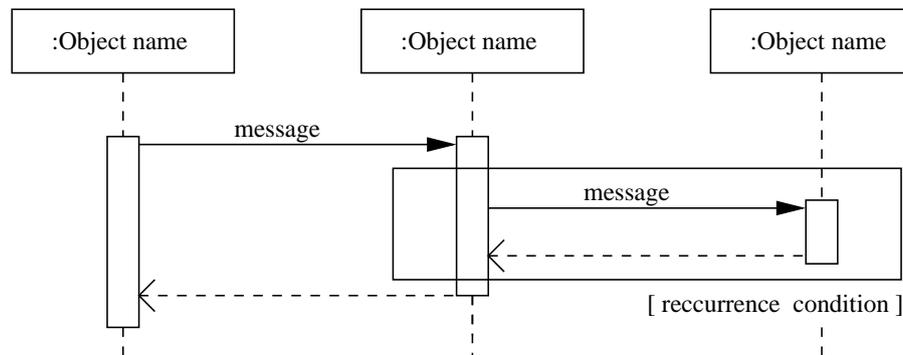
## Schematic of Semantics in a Typical Sequence Diagram



# Behavior Modeling with Sequence Diagrams

## Use and Notation for Iteration

When a sequence of messages takes place within an iteration construct (e.g., a while looping construct in C), the messages can be grouped together within a rectangle

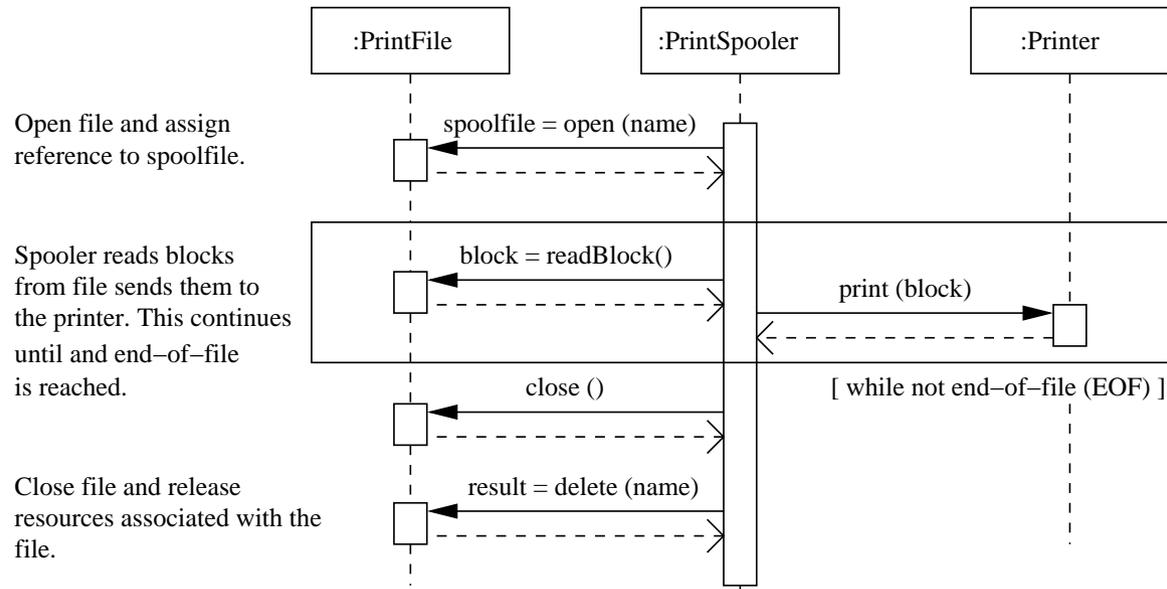


with the test condition for continued loops positioned at the bottom of the rectangle (see Bennett et al, pg's 186).

# Behavior Modeling with Sequence Diagrams

## Use of Textual Annotations

When comments are added to a sequence diagram, the usual positioning is along the left-hand side at the same vertical positioning as the message or activation applies to.



Here, a file is opened, and the spooler reads blocks from the file and sends them to the printer until an end-of-file is reached. Finally, resources associated with the file are released.

# Behavior Modeling with Sequence Diagrams

## Guidelines for Creating Sequence Diagrams

### 1. Define Context of Sequence Diagram

Sequence diagrams can model interactions at the system and subsystem levels.

### 2. Identify the Objects

You can develop a first-cut estimate of the objects that can accomplish the desired behavior with the use cases.

### 3. Draw the Instance Diagrams

Instance sequence diagrams are created by laying out the objects left to right. Add a focus of control to visualize nesting or a point in time where an activation takes place.

### 4. Consider Alternative Scenarios

### 5. Finalize the Distribution of Operations among Classes

Experience indicates that when the robustness analysis is complete, the static model should contain at least 75-80% of the attributes appearing in the logical design.

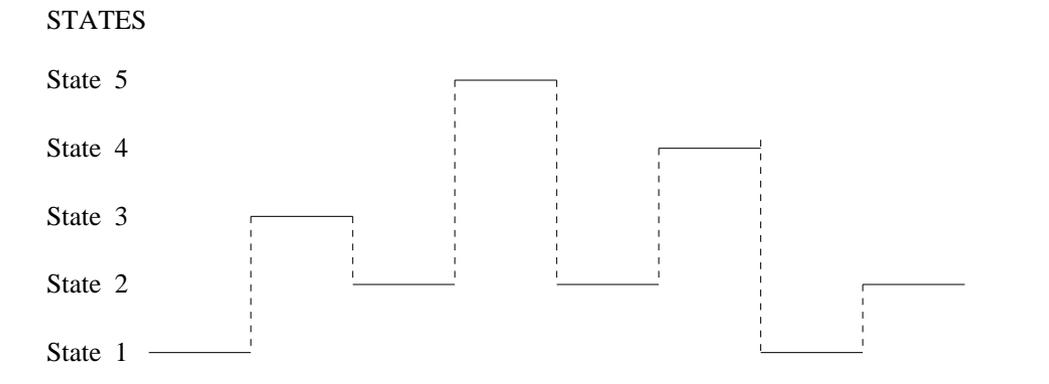
# Finite State Machine Models

## Definition

A state transition diagram is ...

... a graphic representation of the real-time (or on-line) behavior of a system.

State machine behavior can be viewed as a sequence of states versus time.



- **States** summarize past inputs relevant to the current behavior of the system.
- **Transitions** take a system from one state to another. They fire one at a time.
- **Events** are an input/message or interval of time.

# State Machine Behavior

## Recognition and Handling of Events

A state machine will ...

**... only recognize those events defined in the model.**

All other events will be discarded.

## Types of Events and associated Actions

Type of Event	Action
Signal event	The system receives a signal from an external agent.
Call event	A system operation is invoked.
Timing event	A timeout occurs.
Change event	A system property is changed by an external agent.

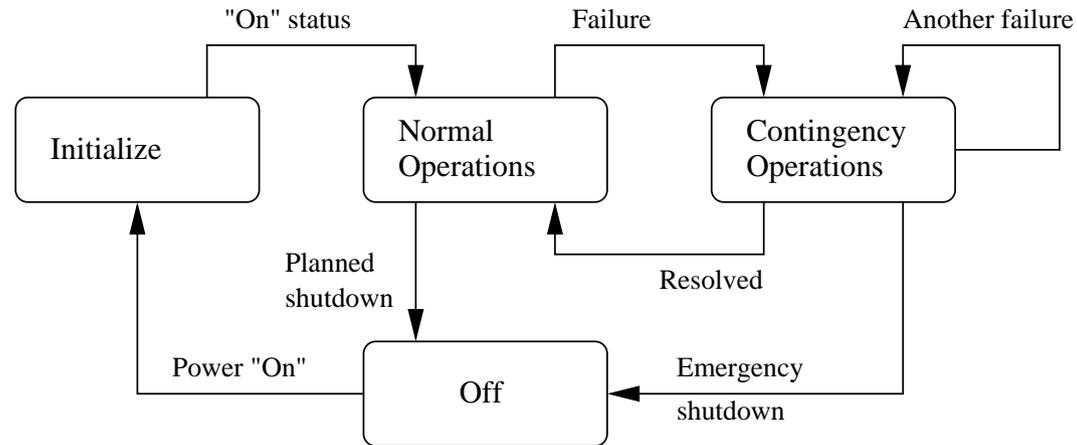
# State Machine Behavior

## State Machine Mechanisms

1. The machine begins at an initial state;
2. The machine waits for an event for an indefinite interval;
3. The event presents itself to the machine;
3. If the event is not accepted in the current state, it is ignored;
4. If the event is accepted in the current state, the designated transition is said to fire.  
The associated action (if any) is produced and the state designated as the resultant state becomes the current state.  
The current and resultant states may be identical.
5. The cycle is repeated from step 2, unless the resultant state is the final state.

# State Machine Behavior

## Example 1. State machine behavior of a spacecraft computer system



### Points to note:

- The boxes in the state diagram show the valid states of the system, and the conditions needed to achieve each state.
- Support is provided for graceful shutdown in emergency situations.
- The remaining states relate to what the system needs to do under normal and contingency operating conditions.

# Finite State Machine Models

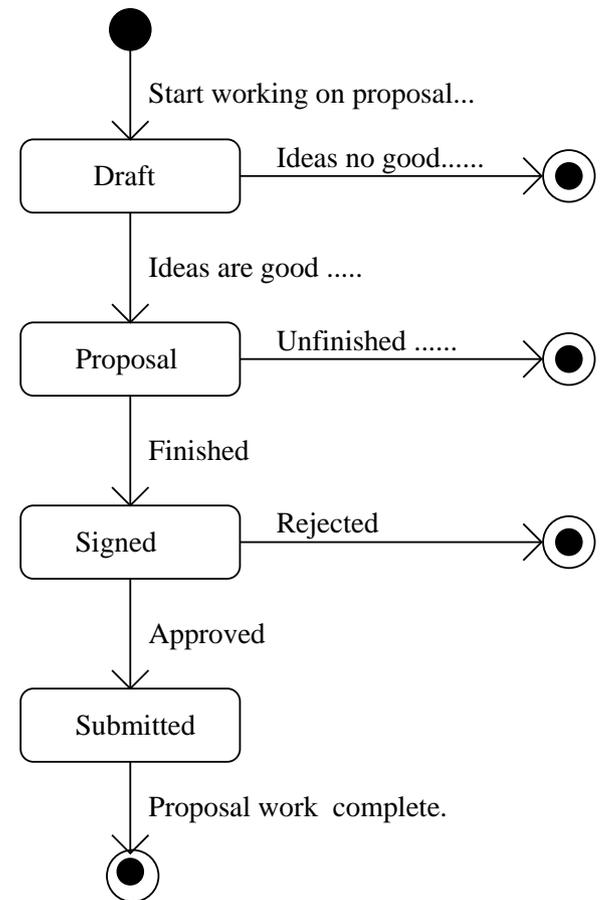
## Example 2. Statechart with multiple exit points.

This example documents ...

**... the states of a research proposal as it progresses through the phases of development, company-level approval, and submission.**

Points to note are as follows:

- Preparation of the proposal draft may be abandoned because there aren't any good ideas....
- A final draft of the proposal may not happen, perhaps because there is insufficient time to work on it.
- Proposals can be rejected because of budget and regulatory concerns.



# Finite State Machine Models

## General Syntax for Guard Conditions

```
action-label / action
```

Some actions will automatically occur soon after the state has been entered. Some actions will automatically occur immediately before the state is exited.

## Event Triggered Actions

For those cases where an action is triggered by an event, the syntax is:

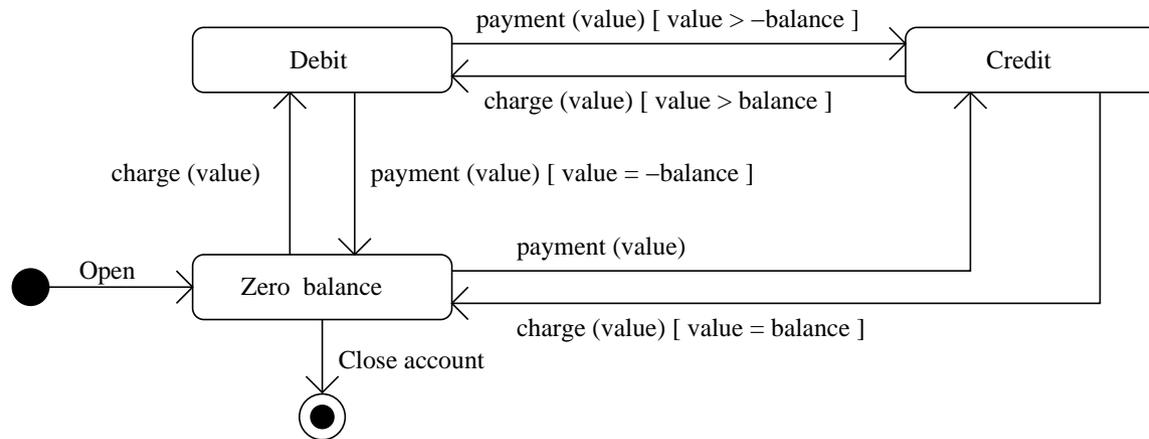
```
event-name ( parameters ) [ guard-condition ] / action
```

Here, ...

- Parameters is a comma-separated list of parameters supplied by the event,
- Guard-condition is a condition that must be true for the event to trigger the action.

# Finite State Machine Models

## Example 3. Behavior Modeling of a Savings Account with Guard Conditions.



### Points to Note

- The account is opened and closed with a Zero Balance state.
- After the account is opened, a payment will move the account into a "Credit" state. Conversely, a charge will move the account into a "Debit" state.
- Subsequent transactions are governed by the guard conditions.

If the quantity of money in the account is positive, then the account will have a "Credit" ...and so forth.

# Finite State Machine Models

## Assessment/Limitations

The benefits of basic state machine models are as follows:

- Easy to use graphical languages (e.g., UML).
- Powerful mathematical algorithms for synthesis of hardware and software and verification.

However, basic state machine models are limited in several respects:

- Basic state machine models do not scale well – even for small-to- moderate sized engineering problems, the number of states can be unmanageable.
- Basic state machine models only support a single thread of concurrency.
- A single state machine cannot directly represent the aggregate behavior of two or more independent processes running concurrently.

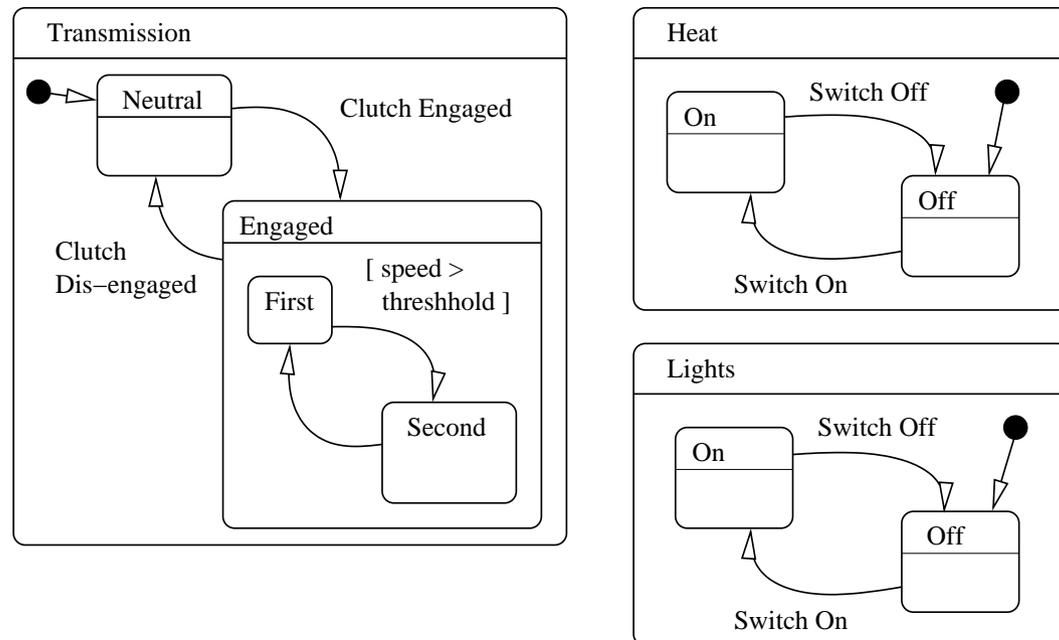
# Statecharts

## Framework for Modeling Concurrent Behaviors

Most real-world systems have behavior that can be ...

**... decomposed into hierarchies and networks of simpler concurrent behaviors.**

**Example.** State machine models for the transmission, heating and lighting systems in an automobile.



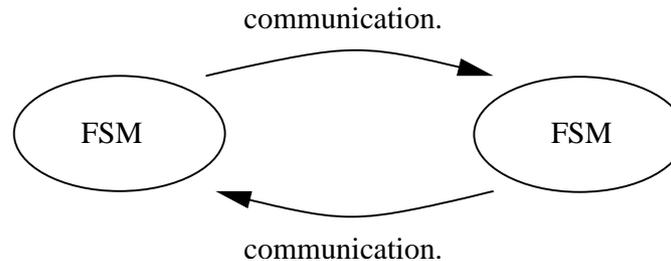
# Statecharts

## Modeling Concurrent Behaviors as Networks of Finite State Machines

Concurrency in engineering systems can be modeled ...

**... using networks of communicating finite state machines (e.g., software systems; digital circuits; control of traffic through intersections).**

Individual processes are represented as FSM.



Embedded software systems are modeled as networks of communicating FSM.

# Statecharts

Statecharts were developed for ...

**... the graphical modeling of control requirements in complex reactive systems,**

and ...

**... to overcome the limitations of basic state machine models.**

## Formal Definition

Formally, statecharts are a higraph-based extension of standard state-transition diagrams, where:

**Statecharts = state transition diagrams + depth + orthogonality + broadcast communication.**

Statecharts incorporate all of the semantics of diagrams for basic finite state machine models.

**References:** Grossman 1997; Harel, 1987; Harel, 1988.

# Statecharts

## Feature 1. Depth

Depth refers to the simplification of models achieved by the hierarchical nesting of states.

Each state encloses a FSM.

- Basic states have no sub-state (bottom of the hierarchy).
- Root states have no parent (top of the hierarchy).

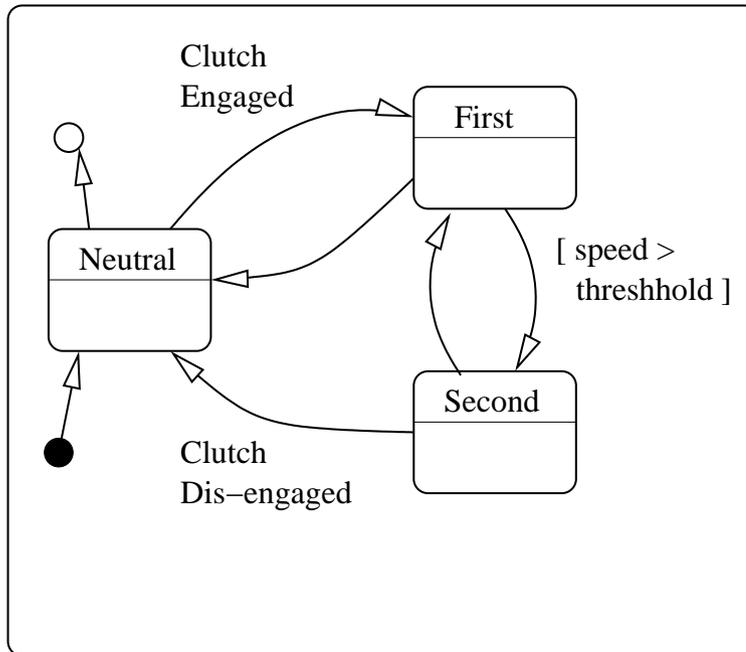
Statecharts can represent ...

**... hierarchies of a single thread (process) or concurrent state state machines.**

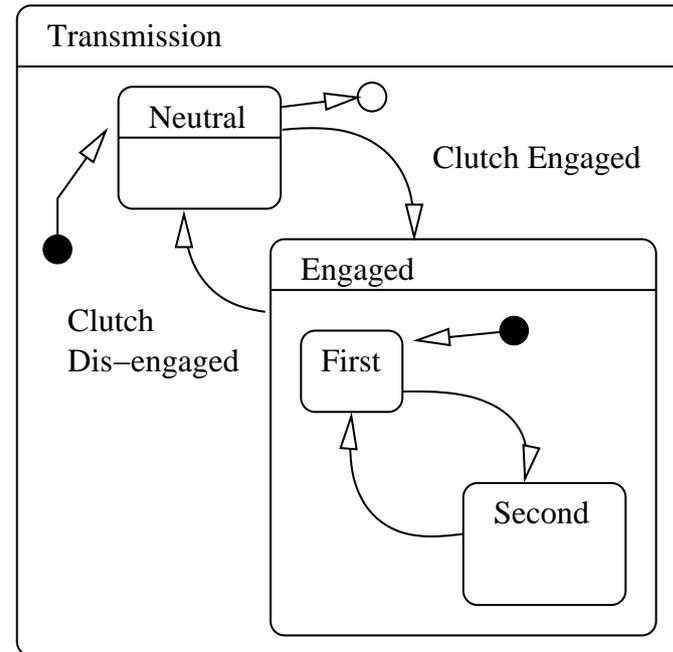
An aggregation of states is called a superstate. The model within an aggregation is a process.

# Statecharts

**Example.** Nested statecharts showing the gear-level view for the transmission system.



Basic FSM Model



Statechart Representation

# Statecharts

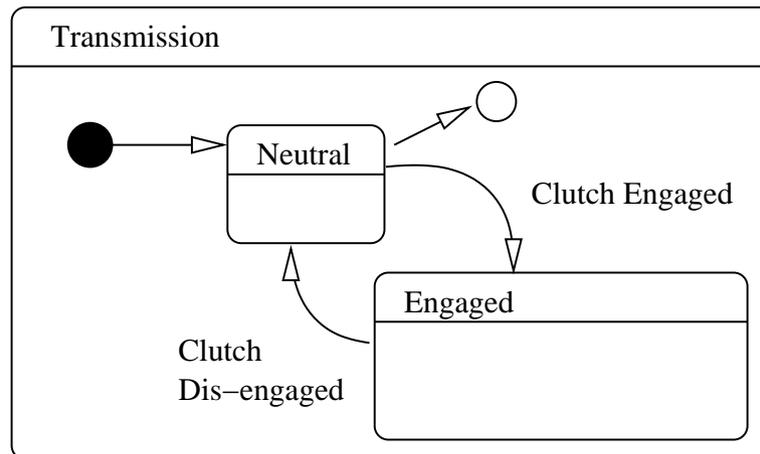
## Points to Note

- The basic FSM model has three states and seven transitions (counting the one from start to neutral).
- The statechart introduces the `Engaged` state to describe the collection of states `first` and `second`. Being in `Engaged` means that an internal FSM is active.
- States enclosed within an undivided box are mutually exclusive, meaning that when the "engaged" state is active, execution must be in either `first` or `second` but not both at the same time.
- The system operation will begin in a `Neutral` state. `First` is the default state when the system is engaged.
- The events `Clutch Engaged` and `Clutch Dis-engaged` trigger transitions from the `Neutral` to `Engaged` and `Engaged` to `Neutral` states, respectively.

# Statecharts

## Points to Note

- The transition from `Engaged` to `Neutral` is shown only once on the diagram, but may be taken from any of the internal states (i.e., `first` or `second`). This form of notation simplifies statechart diagrams.



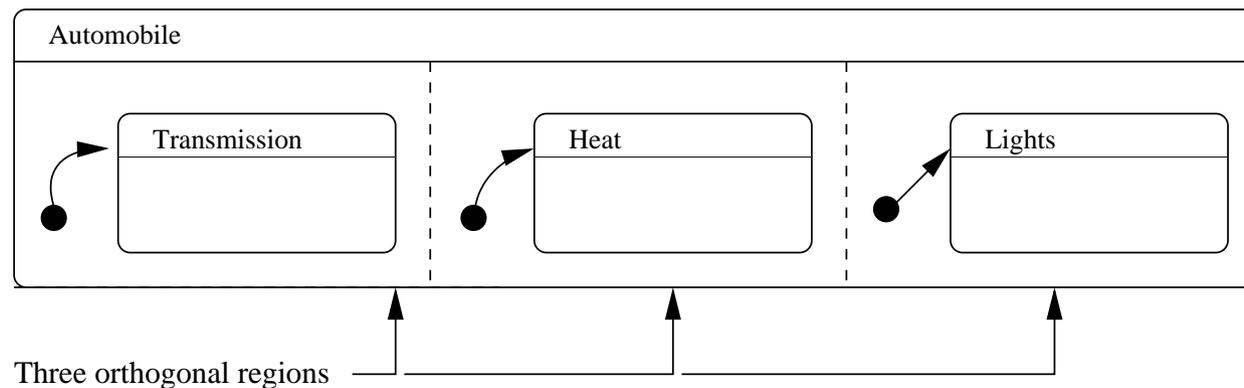
Statechart Representation

# Statecharts

## Feature 2. Orthogonality

Orthogonality refers to the modeling of two or more independent control strategies and/or independent but related processes.

Statecharts represent concurrent (simultaneously active) states by divided superstates.

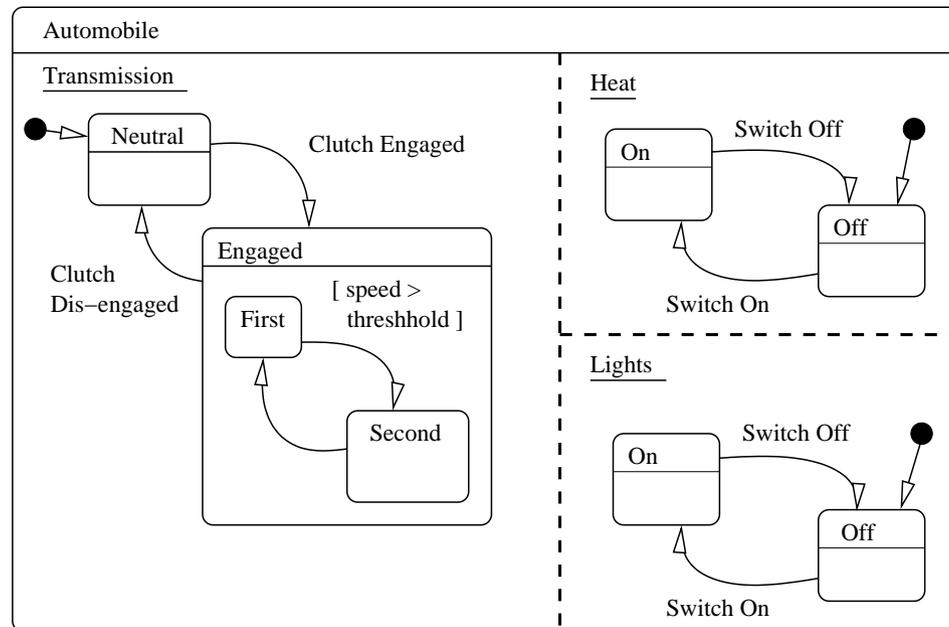


This scenario shows an automobile superstate partitioned into three orthogonal sub-states: transmission, heat, lights.

# Statecharts

## Feature 3. Broadcast Communication

Broadcast means that all machines/processes are visible to other. An output action of any process may be sent to and consumed by any another process.



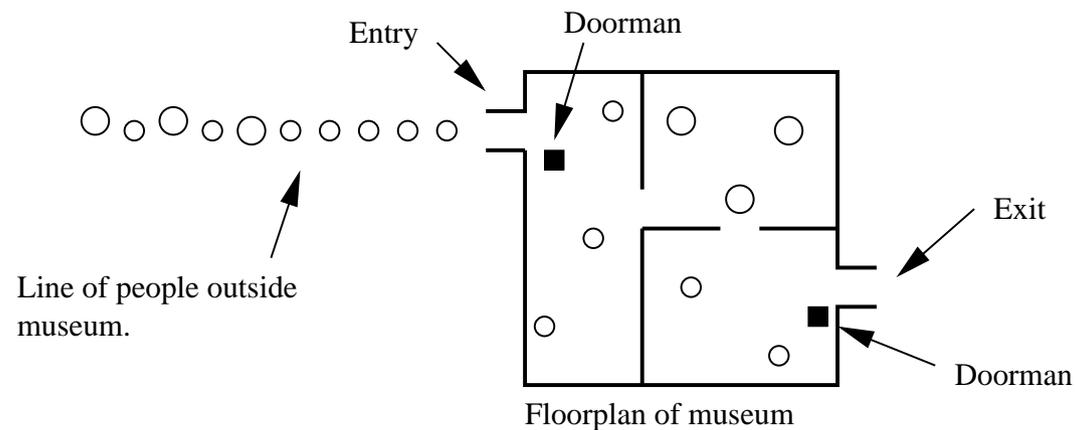
This scenario shows broadcast communication among the transmission, heat, and lighting systems.

# Case Study: Operation of a Museum

## Problem Statement

In this example we ...

**... systematically assemble a simplified systems model of visitor activity and museum occupancy at the Smithsonian Air and Space Museum.**



## Constraints on Behavior

- The museum opens at 10am and closes at 5pm, 7 days a week. When the museum is closed, both the entry and exit doors are locked.

# Case Study: Operation of a Museum

## Constraints on Behavior

The flow of visitors through the museum complies with the following constraints:

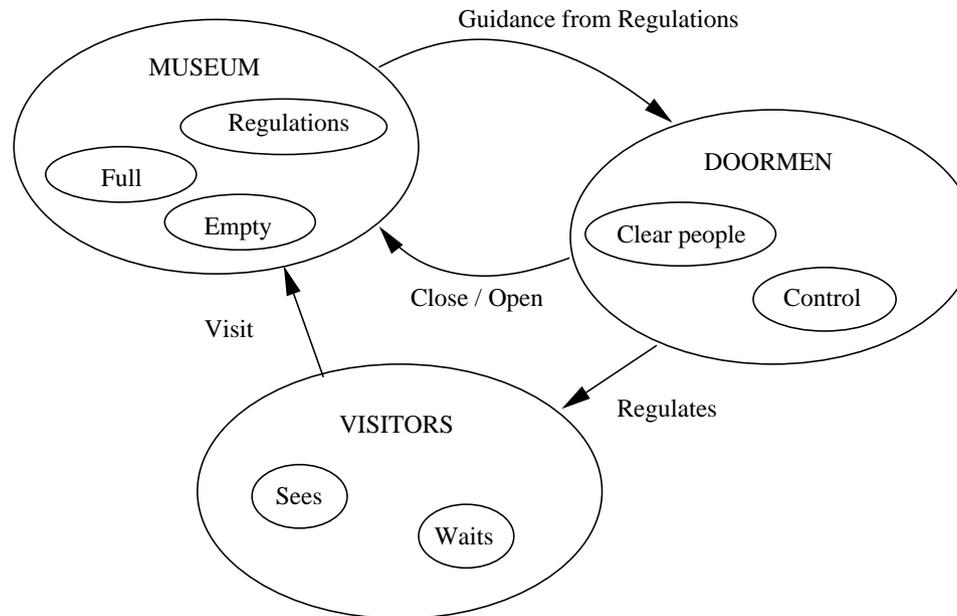
- The doormen are responsible for opening the museum in the morning, controlling the occupancy of the museum during the day, clearing visitors from the museum at 4.55pm, and locking up at precisely 5pm.
- Fire regulations dictate that the capacity of the museum be strictly limited to 1000 people.

More constraints on behavior:

- The museum is "empty" when it opens in the morning. When the museum occupancy is less than 1000, visitors are admitted upon arrival.
- When the museum occupancy equals 1000, it is "full". The doormen will halt admission of new visitors until some of the current visitors have departed.
- During this (hopefully short) period a queue may form outside the museum.

# Case Study: Operation of a Museum

## Systems Framework for the Museum Operation



Point to note:

- Doormen are responsible for controlling the flow of people in and out of the museum.
- Visitors are either "waiting" outside the museum, or inside "seeing" the exhibits.

# Case Study: Operation of a Museum

## Use Cases

Two textual use cases (that are possibly related) for the normal flow of events for a museum visitor.

### Use Case No 1. Normal Operation

#### 1. **Pre-condition:** Museum is Open

**Actors:** Visitor, Doorman

#### **Flow of Events:**

1. Visitor arrives at the museum and doorman lets him/her in.
2. Visitor sees exhibits in the museum.
3. Visitor leaves the museum.

#### 2. **Post-condition:** Visitor has finished seeing the museum and leaves.

# Case Study: Operation of a Museum

## Use Cases (Cont'd)

### Use Case No 2. Museum is Full.

1. **Pre-condition:** Visitor is open and has 1,000 people.

**Actors:** Visitor, Doorman

**Flow of Events:**

1. Visitor arrives.
2. Doorman prohibits visitor from entering museum.
3. Visitor waits in line/queue outside museum.
4. At least one visitor leaves the museum (i.e., the population drops below 1,000).
5. The doorman lets the visitor at the front of the queue in.

2. **Post-condition:** Visitor leaves museum and population drops below 1,000.

# Case Study: Operation of a Museum

## Use Cases (Cont'd)

One textual use case for abnormal flow of events.

**Use Case No 3.** It is 4.55pm.

**1. Pre-condition:** The time is 4.55pm.

**Actors:** Visitor, Doorman

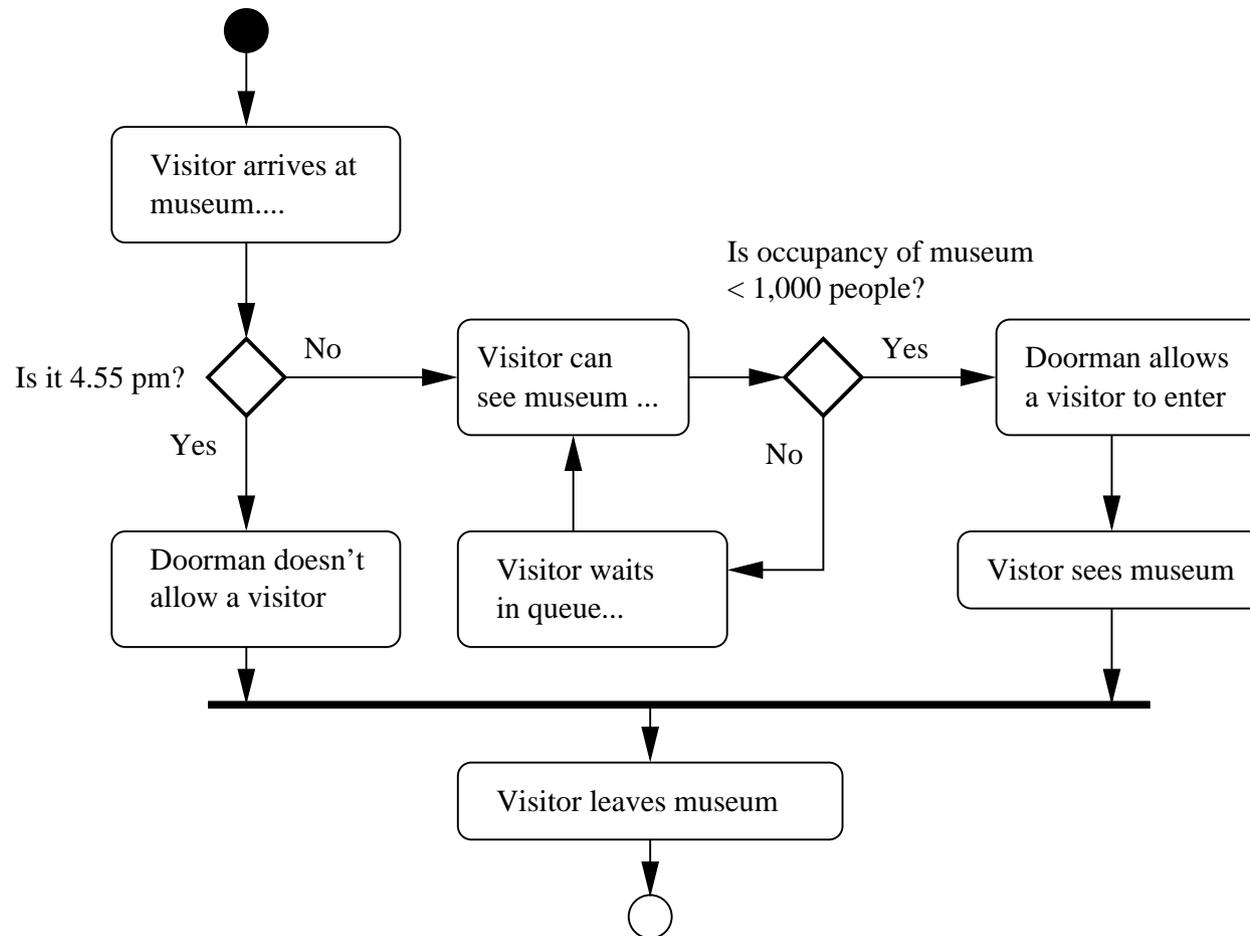
**Flow of Events:**

1. Visitor arrives.
2. Doorman doesn't allow him/her in.
3. Visitor leaves.

**2. Post-condition:** Visitor does not see/visit museum.

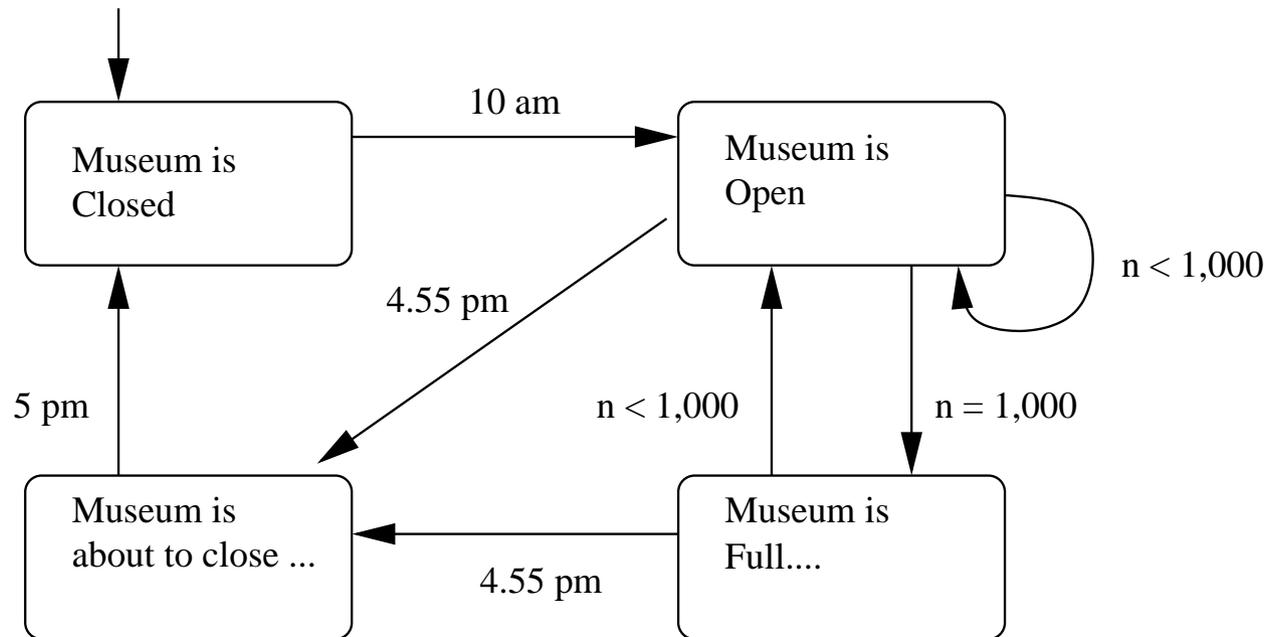
# Case Study: Operation of a Museum

## Activity Diagram for Visitor and Doorman Activity



# Case Study: Operation of a Museum

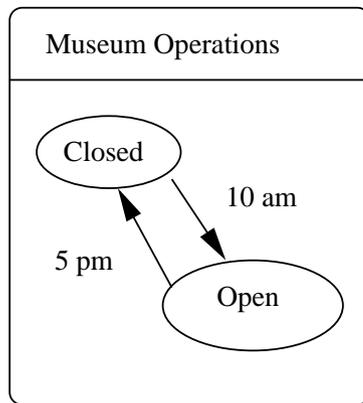
## Finite State Diagram for Museum Occupancy



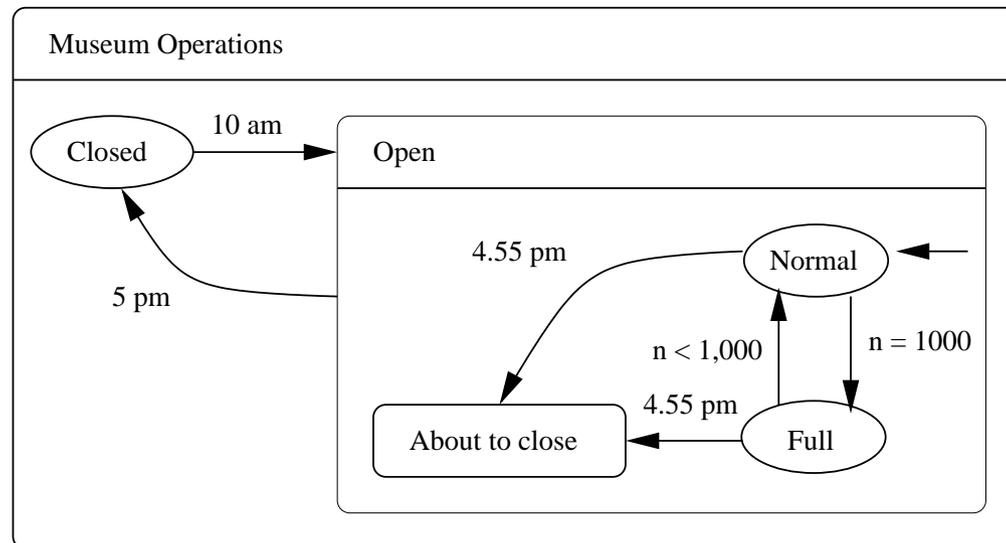
**Note.** The flow of visitors through the museum is controlled by two guard conditions, one on occupancy and a second for time of the day.

# Case Study: Operation of a Museum

## Statechart Diagram for Museum Occupancy



High-Level Model



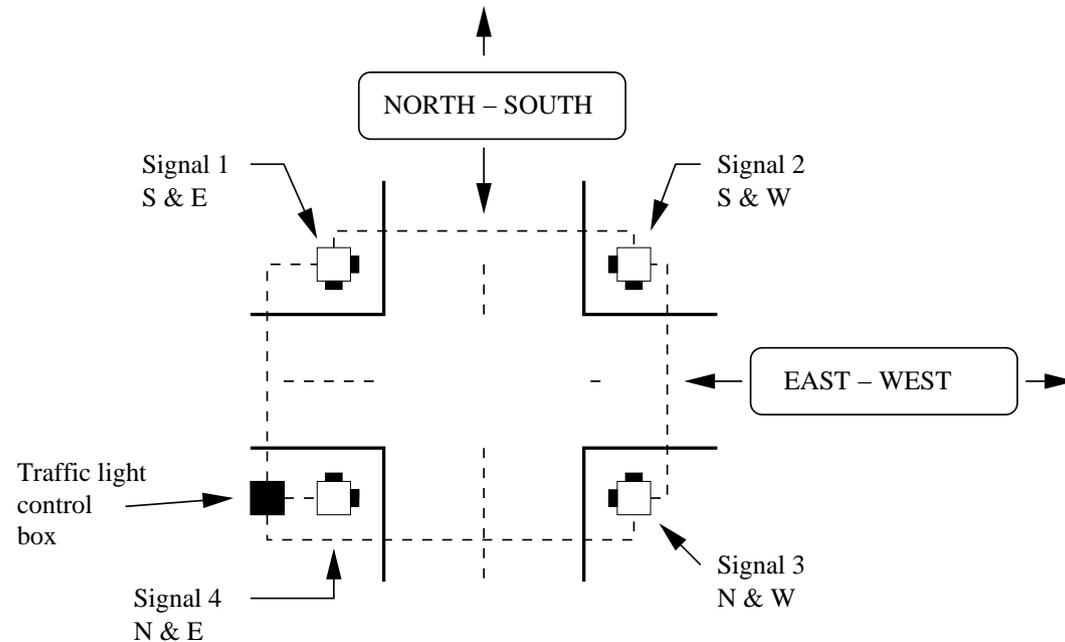
Detailed Model .....

**Note.** Some of the guard conditions are expressions in terms of time. Hence, strictly speaking, this figure is an extended statechart (and not a regular statechart).

# Case Study: Operation of a Traffic Intersection

## Problem Statement

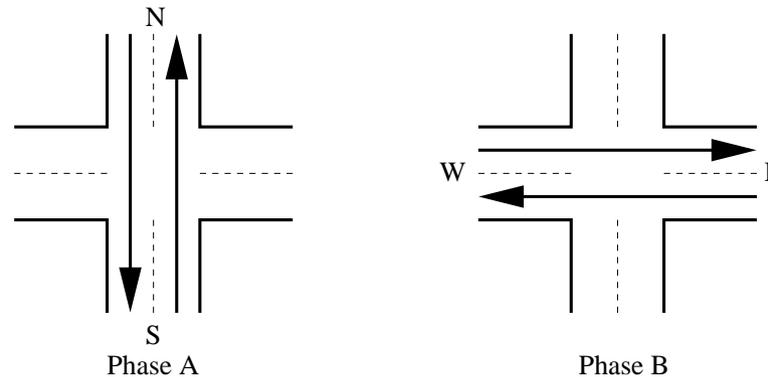
Our goal is to formulate a statchart diagram for behavior at a two-phase traffic intersection.



Each traffic signal will have lights pointing in two directions – for example, traffic signal 1 has lights pointing towards the South (S) and East (E).

# Case Study: Operation of a Traffic Intersection

## Details of Two-Phase Traffic Flow



Two-phase traffic flow implies:

- The lights facing North/South will be the same color at the same time. The same can be said for the lights facing E/W. That is,

Signal 1 S = Signal 4 N

Signal 1 E = Signal 2 W

Signal 2 S = Signal 3 N

Signal 4 E = Signal 3 E

- When one set of lights is either green or yellow, the other set of lights must be red.
- The N/S and E/W sets of lights must follow a regular, systematic pattern of switching colors.

# Case Study: Operation of a Traffic Intersection

## State Table

For "Phase A" (N/S) traffic flow, the light settings are:

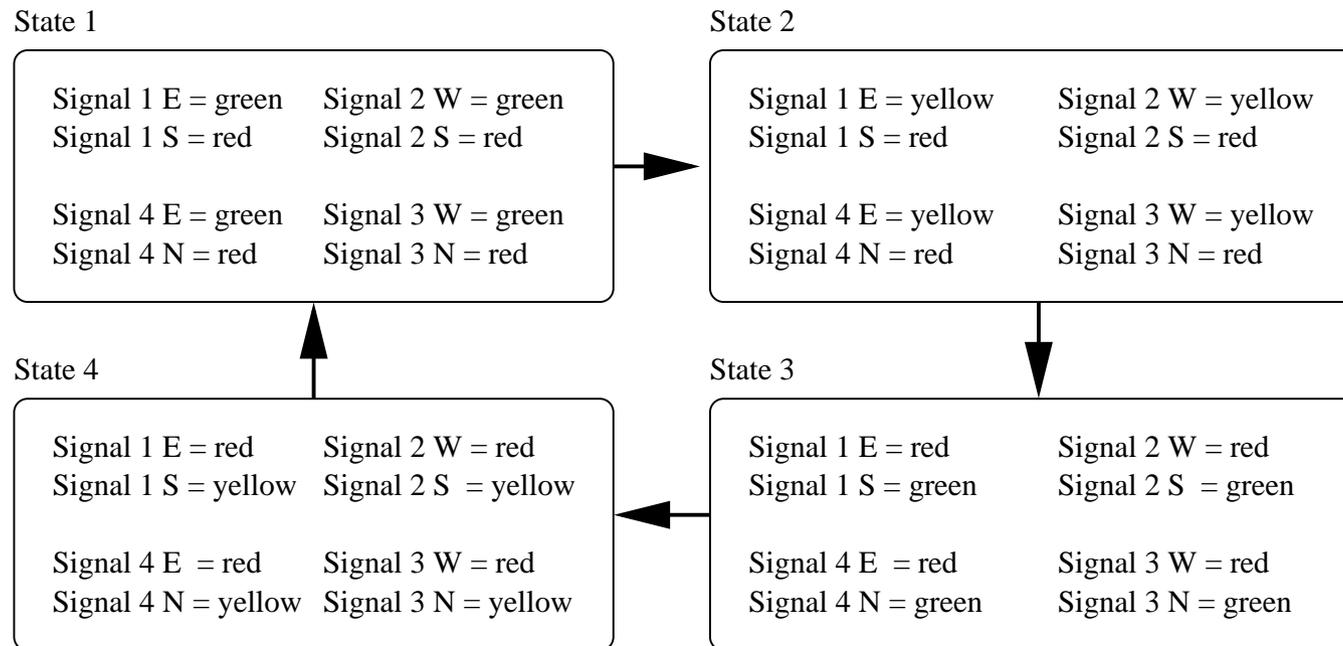
Signal/Direction	North	South	East	West
Signal 1 :	---	Green, Yellow	Red	---
Signal 2 :	---	Green, Yellow	---	Red
Signal 3 :	Green, Yellow	---	---	Red
Signal 4 :	Green, Yellow	---	Red	---

And for "Phase B" (E/W) traffic flow, the light settings are:

Signal/Direction	North	South	East	West
Signal 1 :	---	Red	Green, Yellow	---
Signal 2 :	---	Red	---	Green, Yellow
Signal 3 :	Red	---	---	Green, Yellow
Signal 4 :	Red	---	Green, Yellow	---

# Case Study: Operation of a Traffic Intersection

## Sequences of States for Traffic Light Behavior

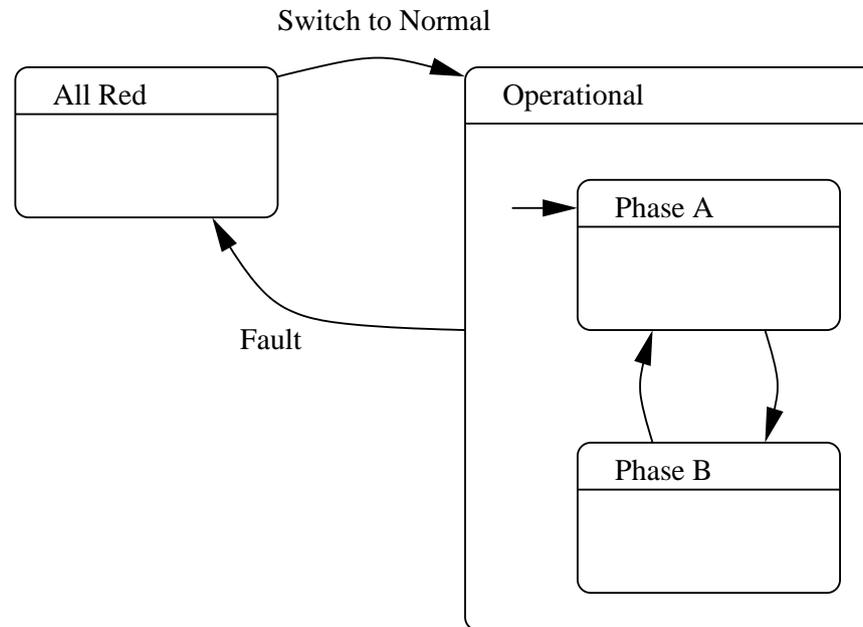


**Note.** States 1 and 2 correspond to Phase A. Phase B corresponds to States 3 and 4.

# Case Study: Operation of a Traffic Intersection

## Statechart Diagram

Now lets expand the behavior model by accounting for an error state – all lights flashing red!!! – and organizing the description of behavior into a hierarchy.



**Note.** Classes for Phase A and Phase B could be expanded to include the detailed light settings described in the state tables.

# References

- Grossman, Ornit. Harel, David, On the Algorithmics of Higraphs, Technical Report CS97-15, The Weizmann Institute of Science, Rehovot, Israel, 1997.
- Harel D., Statecharts: A Visual Formalism for Complex Systems, Science of Computer. Programming, Vol. 8, pp. 231-274, 1987.
- Harel D., On Visual Formalisms, Communications of the ACM, Vol. 31, pp. 514-530, 1988.
- IEEE 1471, Recommended Practice for Architectural Description of Software Intensive Systems, IEEE Std 1471-2000. For details, see [http://standards.ieee.org/reading/ieee/std\\_public/description/se/1471-2000\\_desc.html](http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html) (Accessed April 17, 2010), 2000,
- Rational Software Corporation, Microsoft Software Corporation, UML Summary, Version 1.1., September, 1997. For details, see [http://umlcenter.visual-paradigm.com/umlresources/summ\\_11.pdf](http://umlcenter.visual-paradigm.com/umlresources/summ_11.pdf),
- OMG Systems Modeling Language. See: <http://www.omgsysml.org/>