

Chapter 5: System Modeling

System modeling is the process of creating abstract models of a system¹¹¹¹. Each model shows a different view or perspective of that system²²²².

Topics Covered

This lecture discusses the following types of models³:

- **Context models:** Show the system's environment⁴.
- **Interaction models:** Show how the system and users or other systems interact⁵.
- **Structural models:** Show how the system is organized⁶.
- **Behavioral models:** Show what the system does and how it responds⁷.
- **Model-driven engineering:** Creating software by designing models first⁸.

Generalization

Generalization is a tool we use every day to handle complex information⁹⁹⁹⁹.

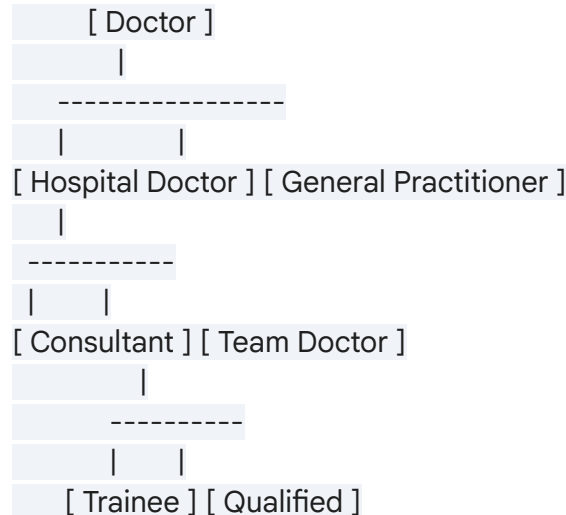
- **How it works:** Instead of learning every single detail about every single thing, we group things into general classes like "animals," "cars," or "houses"¹⁰.
- **Inference:** This helps us assume that if something belongs to a class, it has certain common traits¹¹. For example, squirrels and rats are both rodents¹².

Generalization in System Modeling

- When modeling a system, we look for classes that can be generalized¹³.
- **Main Benefit:** If a change is needed, you don't have to check every single class; you only check the relevant general classes¹⁴.
- **Implementation:** In languages like Java, this is done using **inheritance**¹⁵.

- **Attributes and Operations:** Higher-level classes (superclasses) share their traits with lower-level classes (subclasses)¹⁶. Subclasses then add their own specific details¹⁷.

A Generalization Hierarchy (Doctor Example)



A Detailed Generalization Hierarchy 19

| Class | Attributes | Operations |

| :--- | :--- | :--- |

| Doctor (Superclass) | Name, Phone #, Email 20 | register(), de-register() 21 |

| Hospital doctor (Subclass) | Staff #, Pager # 22 | Inherits Doctor operations 23 |

| General practitioner (Subclass) | Practice, Address 24 | Inherits Doctor operations 25 |

Behavioral Models

Behavioral models show the **dynamic behavior** of a system as it runs²⁶²⁶²⁶. They describe what happens when a system reacts to a "stimulus" from its environment²⁷.

Types of Stimuli

1. **Data:** New information arrives that the system must process²⁸.
2. **Events:** Something happens that triggers the system to do work (the event might also

bring data with it)²⁹.

Data-driven Modeling

- Many business systems are data-processing systems³⁰.
- These systems are controlled by the data coming in, with very few external events³¹.
- The models show the sequence of actions taken to turn input into output³²³².
- **Usage:** Great for requirements analysis to see the "end-to-end" process³³.

Event-driven Modeling

- Real-time systems are often event-driven³⁴³⁴³⁴.
- **Example:** A landline phone switch reacts to a 'receiver off hook' event by making a dial tone³⁵.
- These models show how a system reacts to internal or external events³⁶.
- **Core Idea:** The system has a set number of "states," and events cause it to switch (transition) from one state to another³⁷.

Behavioral Perspective

This perspective focuses on the dynamic aspects of the system—how it behaves over time and responds to triggers³⁸³⁸³⁸³⁸. Both **Activity Diagrams** and **State Diagrams** belong here³⁹.

Activity Diagrams

Activity diagrams show the steps of how a system works to help us understand the flow of control⁴⁰⁴⁰⁴⁰⁴⁰.

- **What they show:** The order of activities and whether they happen one after another (sequential) or at the same time (parallel/concurrent)⁴¹.
- **Flow:** They start at an initial point, follow decision paths, and end at a final point⁴².

- **Usage:** Modeling business workflows, describing use case steps, or showing the logic of a single operation⁴³.

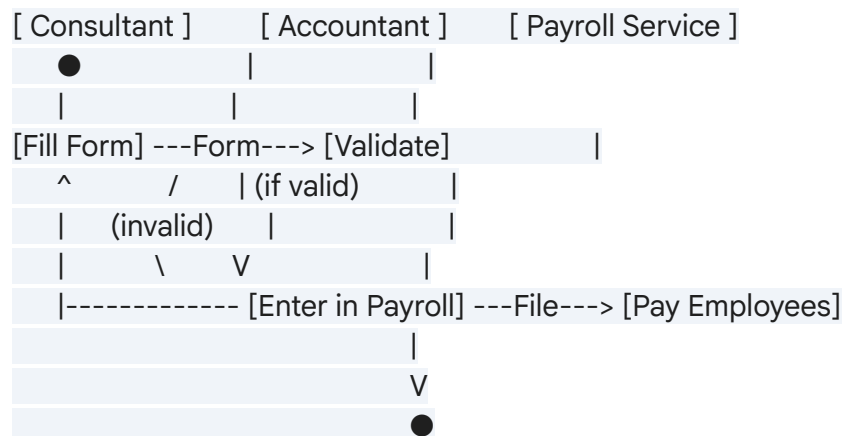
Activity Diagram Components

Swimlanes (Partitions)

Swimlanes are vertical or horizontal columns that group activities⁴⁶⁴⁶⁴⁶⁴⁶.

- **Purpose:** They show exactly which "actor" (person) or department is responsible for which task⁴⁷.
- **Value:** Incredibly useful for defining clear roles and responsibilities⁴⁸.

ASCII Example: Staff Expenses Swimlane⁴⁹⁴⁹⁴⁹⁴⁹



Examples of Activity Diagrams

- **Login Page:** Shows entering a name/password, checking if they are correct (decision node), and either showing an error or displaying user settings⁵⁰⁵⁰⁵⁰⁵⁰.
- **Banking System:** Shows checking an account and then choosing between a "Withdrawal" or "Deposit" path⁵¹⁵¹⁵¹⁵¹.
- **Process Order:** Shows how "Fill Order" and "Send Invoice" can happen in **parallel** (at the

same time) after an order is received⁵²⁵²⁵²⁵².

State Diagrams (State Machine Diagrams)

State diagrams model the behavior of a **single object** over its entire lifetime⁵³⁵³⁵³⁵³.

- **What they show:** The different "states" an object can be in and the "events" that make it change states⁵⁴.
- **Ideal for:** Reactive systems like user interfaces, network protocols, or embedded systems⁵⁵⁵⁵⁵⁵⁵⁵.
- **Simple Example:** A door can be "Open," "Closed," or "Locked." Actions like "Close" or "Lock" change its state⁵⁶.

State Machine Models

- **Nodes:** Represent the states the system can be in⁵⁷.
- **Arcs (Arrows):** Represent the events that trigger a move from one node to another⁵⁸.
- **Turnstile Example:** Starts **Locked** → Event: Put in coin → Becomes **Unlocked** → Event: Push arm → Returns to **Locked**⁵⁹⁵⁹⁵⁹⁵⁹.

Details inside a State

A state can have specific activities associated with it⁶⁰:

- **Entry:** Action performed immediately upon entering the state⁶¹.
 - **Do:** Action performed while the object is in the state⁶².
 - **Exit:** Action performed just before leaving the state⁶³.
 - **Deferrable Trigger:** An event that doesn't cause a transition now but stays in a "pool" to be used later⁶⁴.
-

State Machine vs. Activity Diagram

Aspect	State Machine Diagram	Activity Diagram
Primary Focus	State changes of an object over time ⁶⁵ .	Flow of control or data in a process ⁶⁶ .
Granularity	Focuses on a single object or entity ⁶⁷ .	Focuses on high-level processes with multiple objects ⁶⁸ .
Parallelism	Less emphasis; harder to show ⁶⁹ .	Explicitly shows parallel flows with forks/joins ⁷⁰ .
Usage	Embedded systems, protocols, state-dependent logic ⁷¹ .	Business workflows, software process flows ⁷² .

Substates

- **Simple State:** Has no structure inside it⁷³.
- **Composite State:** Contains "nested" states (substates)⁷⁴.
- **Benefit:** Simplifies complex diagrams by showing that certain states only exist within a specific context⁷⁵.
- **Example:** In a heater system, the **Cooling** state might have substates like **Startup**, **Ready**, and **Running**⁷⁶.

Model-Driven Engineering (MDE)

MDE is a software creation method where the **model** is the main product, not the program code⁷⁷.

- **The Blueprint:** Engineers create a very detailed design model⁷⁸.
- **Auto-Generation:** Special tools read the model and automatically write the complex code⁷⁹.
- **Benefit:** Engineers can focus on the "big picture" (what the software does) rather than tiny coding details⁸⁰.

Pros and Cons of MDE

Pros	Cons
Systems are considered at higher abstraction levels.	Abstraction models aren't always right for final implementation.
Cheaper to move systems to new platforms by re-generating code.	Cost of developing the "translators" might be higher than the savings.

Model-Driven Architecture (MDA)

MDA is a specific type of MDE that uses a subset of UML models to describe a system⁸³⁸³⁸³⁸³.

The 3 Types of MDA Models

1. **Computation Independent Model (CIM):** Focuses on business requirements and "what" the system does, ignoring technology⁸⁴⁸⁴⁸⁴.
2. **Platform Independent Model (PIM):** Focuses on "how" the system works logically using UML, without choosing a specific platform (like Java or .NET)⁸⁵⁸⁵⁸⁵.
3. **Platform Specific Model (PSM):** Adds implementation details for a specific platform, making it ready for code generation⁸⁶⁸⁶⁸⁶.

The Transformation Flow⁸⁷⁸⁷⁸⁷⁸⁷:

CIM → (Translator) → PIM → (Translator) → PSM → (Translator) → Code

Executable UML (xUML)

The goal of MDE is to have models turn into code automatically⁸⁸. This is possible using a specific subset of UML 2 called **Executable UML**⁸⁹.

Key Model Types in xUML⁹⁰

- **Domain Models:** Use class diagrams to show the main concerns of the system⁹¹.
- **Class Models:** Define specific classes, attributes, and operations⁹².
- **State Models:** Every class gets its own state diagram to describe its lifecycle⁹³.

Quick Exam Questions (5 marks level)

1. Explain the difference between Data-driven and Event-driven modeling.

Answer: Data-driven modeling is used for systems (like business processing) primarily controlled by incoming data; it shows the sequence of actions that turn inputs into outputs⁹⁴⁹⁴⁹⁴⁹⁴. Event-driven modeling is used for real-time systems triggered by external or internal events; it shows how the system switches between a finite number of states in response to these triggers⁹⁵.

2. What are "Swimlanes" in an activity diagram, and why are they useful?

Answer: Swimlanes are vertical or horizontal columns that group activities in an activity diagram based on the "actor" or department responsible for them⁹⁶. They are useful because they clearly define roles and responsibilities within a complex process, showing exactly who performs each step⁹⁷.

3. Describe the three types of activities (Entry, Do, Exit) that can occur within a State Machine Diagram state.

Answer: 1. **Entry Activity:** An action performed as soon as the object enters the state⁹⁸. 2. **Do Activity:** An action that continues to be performed as long as the object stays in that state⁹⁹. 3. **Exit Activity:** An action performed just before the object leaves the state to transition to another¹⁰⁰.

4. Compare and contrast Activity Diagrams and State Machine Diagrams.

Answer: Activity Diagrams focus on the flow of control or data through a process involving multiple entities, making them ideal for high-level workflows¹⁰¹. State Machine Diagrams focus on the lifecycle of a single object and how it changes states over time, making them ideal for modeling reactive, state-dependent behavior in individual components¹⁰².

5. Define Model-Driven Architecture (MDA) and explain its three main model types (CIM, PIM, PSM).

Answer: MDA is a model-focused approach where software is designed using UML models at different abstraction levels¹⁰³. 1. **CIM** (Computation Independent): Focuses on technology-neutral business requirements¹⁰⁴. 2. **PIM** (Platform Independent): Models the system's logical structure using UML without implementation details¹⁰⁵. 3. **PSM** (Platform Specific): Transforms the PIM to include details for a specific platform like Java, ready for code generation¹⁰⁶.

Would you like me to draw a State Machine Diagram for the library book scenario mentioned at the end of the lecture?