

## Chapter 5: System Modeling

System modeling is the process of creating abstract models of a system<sup>1111</sup>. Each model shows a different view or perspective of that system<sup>2222</sup>.

### Topics Covered

This lecture discusses the following types of models<sup>3</sup>:

- **Context models:** Show the system's environment<sup>4</sup>.
- **Interaction models:** Show how the system and users or other systems interact<sup>5</sup>.
- **Structural models:** Show how the system is organized<sup>6</sup>.
- **Behavioral models:** Show what the system does and how it responds<sup>7</sup>.
- **Model-driven engineering:** Creating software by designing models first<sup>8</sup>.

---

## Generalization

Generalization is a tool we use every day to handle complex information<sup>9999</sup>.

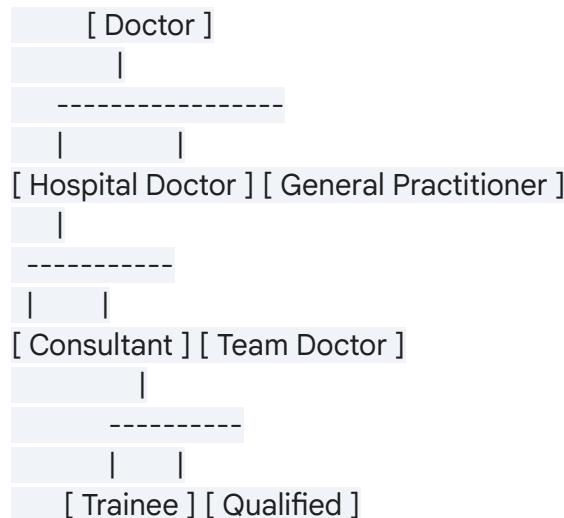
- **How it works:** Instead of learning every single detail about every single thing, we group things into general classes like "animals," "cars," or "houses"<sup>10</sup>.
- **Inference:** This helps us assume that if something belongs to a class, it has certain common traits<sup>11</sup>. For example, squirrels and rats are both rodents<sup>12</sup>.

### Generalization in System Modeling

- When modeling a system, we look for classes that can be generalized<sup>13</sup>.
- **Main Benefit:** If a change is needed, you don't have to check every single class; you only check the relevant general classes<sup>14</sup>.
- **Implementation:** In languages like Java, this is done using **inheritance**<sup>15</sup>.

- **Attributes and Operations:** Higher-level classes (superclasses) share their traits with lower-level classes (subclasses)<sup>16</sup>. Subclasses then add their own specific details<sup>17</sup>.

### 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 runs<sup>262626</sup>. They describe what happens when a system reacts to a "stimulus" from its environment<sup>27</sup>.

### Types of Stimuli

1. **Data:** New information arrives that the system must process<sup>28</sup>.
2. **Events:** Something happens that triggers the system to do work (the event might also

bring data with it)<sup>29</sup>.

## Data-driven Modeling

- Many business systems are data-processing systems<sup>30</sup>.
- These systems are controlled by the data coming in, with very few external events<sup>31</sup>.
- The models show the sequence of actions taken to turn input into output<sup>3232</sup>.
- **Usage:** Great for requirements analysis to see the "end-to-end" process<sup>33</sup>.

## Event-driven Modeling

- Real-time systems are often event-driven<sup>343434</sup>.
- **Example:** A landline phone switch reacts to a 'receiver off hook' event by making a dial tone<sup>35</sup>.
- These models show how a system reacts to internal or external events<sup>36</sup>.
- **Core Idea:** The system has a set number of "states," and events cause it to switch (transition) from one state to another<sup>37</sup>.

---

## Behavioral Perspective

This perspective focuses on the dynamic aspects of the system—how it behaves over time and responds to triggers<sup>38383838</sup>. Both **Activity Diagrams** and **State Diagrams** belong here<sup>39</sup>.

## Activity Diagrams

Activity diagrams show the steps of how a system works to help us understand the flow of control<sup>40404040</sup>.

- **What they show:** The order of activities and whether they happen one after another (sequential) or at the same time (parallel/concurrent)<sup>41</sup>.
- **Flow:** They start at an initial point, follow decision paths, and end at a final point<sup>42</sup>.

- **Usage:** Modeling business workflows, describing use case steps, or showing the logic of a single operation<sup>43</sup>.

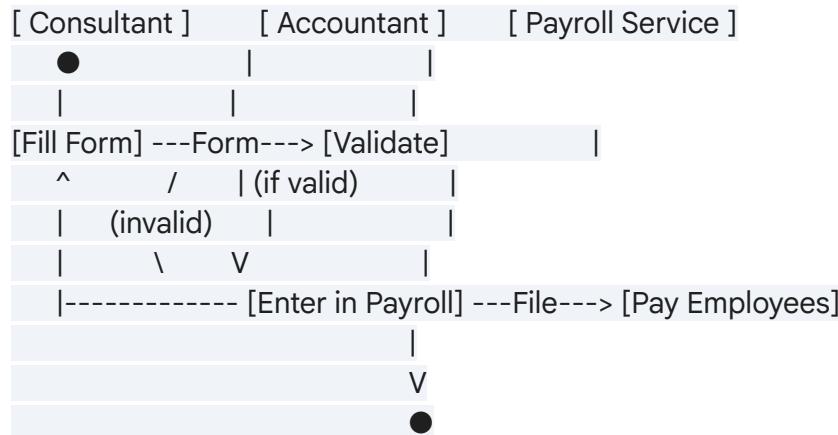
## Activity Diagram Components

### Swimlanes (Partitions)

Swimlanes are vertical or horizontal columns that group activities<sup>46464646</sup>.

- **Purpose:** They show exactly which "actor" (person) or department is responsible for which task<sup>47</sup>.
- **Value:** Incredibly useful for defining clear roles and responsibilities<sup>48</sup>.

### ASCII Example: Staff Expenses Swimlane<sup>49494949</sup>



### 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 settings<sup>50505050</sup>.
- **Banking System:** Shows checking an account and then choosing between a "Withdrawal" or "Deposit" path<sup>51515151</sup>.
- **Process Order:** Shows how "Fill Order" and "Send Invoice" can happen in **parallel** (at the

same time) after an order is received<sup>52525252</sup>.

---

## State Diagrams (State Machine Diagrams)

State diagrams model the behavior of a **single object** over its entire lifetime<sup>53535353</sup>.

- **What they show:** The different "states" an object can be in and the "events" that make it change states<sup>54</sup>.
- **Ideal for:** Reactive systems like user interfaces, network protocols, or embedded systems<sup>55555555</sup>.
- **Simple Example:** A door can be "Open," "Closed," or "Locked." Actions like "Close" or "Lock" change its state<sup>56</sup>.

## State Machine Models

- **Nodes:** Represent the states the system can be in<sup>57</sup>.
- **Arcs (Arrows):** Represent the events that trigger a move from one node to another<sup>58</sup>.
- **Turnstile Example:** Starts **Locked** → Event: Put in coin → Becomes **Unlocked** → Event: Push arm → Returns to **Locked**<sup>59595959</sup>.

## Details inside a State

A state can have specific activities associated with it<sup>60</sup>:

- **Entry:** Action performed immediately upon entering the state<sup>61</sup>.
  - **Do:** Action performed while the object is in the state<sup>62</sup>.
  - **Exit:** Action performed just before leaving the state<sup>63</sup>.
  - **Deferrable Trigger:** An event that doesn't cause a transition now but stays in a "pool" to be used later<sup>64</sup>.
-

# State Machine vs. Activity Diagram

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

## Substates

- **Simple State:** Has no structure inside it<sup>73</sup>.
  - **Composite State:** Contains "nested" states (substates)<sup>74</sup>.
  - **Benefit:** Simplifies complex diagrams by showing that certain states only exist within a specific context<sup>75</sup>.
  - **Example:** In a heater system, the **Cooling** state might have substates like **Startup**, **Ready**, and **Running**.

# Model-Driven Engineering (MDE)

MDE is a software creation method where the **model** is the main product, not the program code<sup>77</sup>.

- **The Blueprint:** Engineers create a very detailed design model<sup>78</sup>.
- **Auto-Generation:** Special tools read the model and automatically write the complex code<sup>79</sup>.
- **Benefit:** Engineers can focus on the "big picture" (what the software does) rather than tiny coding details<sup>80</sup>.

## 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 system<sup>83838383</sup>.

### The 3 Types of MDA Models

1. **Computation Independent Model (CIM):** Focuses on business requirements and "what" the system does, ignoring technology<sup>848484</sup>.
2. **Platform Independent Model (PIM):** Focuses on "how" the system works logically using UML, without choosing a specific platform (like Java or .NET)<sup>858585</sup>.
3. **Platform Specific Model (PSM):** Adds implementation details for a specific platform, making it ready for code generation<sup>868686</sup>.

The Transformation Flow<sup>87878787</sup>:

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

---

## Executable UML (xUML)

The goal of MDE is to have models turn into code automatically<sup>88</sup>. This is possible using a specific subset of UML 2 called **Executable UML**<sup>89</sup>.

Key Model Types in xUML <sup>90</sup>

- **Domain Models:** Use class diagrams to show the main concerns of the system<sup>91</sup>.
- **Class Models:** Define specific classes, attributes, and operations<sup>92</sup>.
- **State Models:** Every class gets its own state diagram to describe its lifecycle<sup>93</sup>.

---

## 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 outputs<sup>94949494</sup>. 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 triggers<sup>95</sup>.

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 them<sup>96</sup>. They are useful because they clearly define roles and responsibilities within a complex process, showing exactly who performs each step<sup>97</sup>.

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 state<sup>98</sup>. 2. **Do Activity:** An action that continues to be performed as long as the object stays in that state<sup>99</sup>. 3. **Exit Activity:** An action performed just before the object leaves the state to transition to another<sup>100</sup>.

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 workflows<sup>101</sup>. 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 components<sup>102</sup>.

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 levels<sup>103</sup>. 1. **CIM** (Computation Independent): Focuses on technology-neutral business requirements<sup>104</sup>. 2. **PIM** (Platform Independent): Models the system's logical structure using UML without implementation details<sup>105</sup>. 3. **PSM** (Platform Specific): Transforms the PIM to include details for a specific platform like Java, ready for code generation<sup>106</sup>.

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