Representing System Architecture

Logical View
End-user
Functionality

Process View
System integrators
Performance
Scalability
Throughput

Implementation View
Programmers
Software management

Deployment View
System engineering
System topology
Delivery, installation
Communication

Use Case View

Conceptual
Physical
How many views?

- Simplified models to fit the context

- Not all systems require all views:
  - Single processor: drop deployment view
  - Single process: drop process view
  - Very Small program: drop implementation view

- Adding views:
  - Data view, security view
The Value of the UML

- Is an open standard
- Supports the entire software development lifecycle
- Supports diverse applications areas
- Is based on experience and needs of the user community
- Supported by many tools
Creating the UML

- OMG Acceptance, Nov 1997
- Final submission to OMG, Sep ‘97
- First submission to OMG, Jan ‘97
- Public feedback
- UML partners
- Web - June ‘96
- OOPSLA ´95
- Unified Method 0.8
- Other methods
- Booch method
- OMT
- OOSE
- UML 1.0
- UML 1.1
- UML 1.2
- UML 1.3
- UML partners
UML Partners

- Rational Software Corporation
- Hewlett-Packard
- I-Logix
- IBM
- ICON Computing
- Intellicorp
- MCI Systemhouse
- Microsoft
- ObjecTime
- Oracle
- Platinum Technology
- Taskon
- Texas Instruments/Sterling Software
- Unisys
Contributions to the UML

- Meyer
  - Before and after conditions
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Shlaer - Mellor
  - Object lifecycles
- Gamma, et al
  - Statecharts
  - Frameworks and patterns,
- HP Fusion
  - Operation descriptions and message numbering
- Embley
  - Singleton classes and high-level view
- Wirfs-Brock
  - Responsibilities
- Odell
  - Classification
- OMT
- Booch method
- OOSE
- OMT
Overview of the UML

- The UML is a language for
  - visualizing
  - specifying
  - constructing
  - documenting

the artifacts of a software-intensive system
Overview of the UML

- Modeling elements
- Relationships
- Extensibility Mechanisms
- Diagrams
Modeling Elements

- Structural elements
  - class, interface, collaboration, use case, active class, component, node

- Behavioral elements
  - interaction, state machine

- Grouping elements
  - package, subsystem

- Other elements
  - note
Relationships

- Dependency
- Association
- Generalization
- Realization
Extensibility Mechanisms

- Stereotype
- Tagged value
- Constraint

```
ActionQueue
{version = 3.2}

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(a : Action)</td>
<td>{add runs in O(1) time}</td>
</tr>
<tr>
<td>remove(n : Integer)</td>
<td></td>
</tr>
<tr>
<td>«query»</td>
<td></td>
</tr>
<tr>
<td>length() : Integer</td>
<td></td>
</tr>
<tr>
<td>«helper functions»</td>
<td></td>
</tr>
<tr>
<td>reorder()</td>
<td></td>
</tr>
</tbody>
</table>
```
A **model** is a complete description of a system from a particular perspective.
A diagram is a view into a model
- Presented from the aspect of a particular stakeholder
- Provides a partial representation of the system
- Is semantically consistent with other views

In the UML, there are nine standard diagrams
- Static views: use case, class, object, component, deployment
- Dynamic views: sequence, collaboration, statechart, activity
Use Case Diagram

- Captures system functionality as seen by users
Use Case Diagram

- Captures system functionality as seen by users
- Built in early stages of development
- Purpose
  - Specify the context of a system
  - Capture the requirements of a system
  - Validate a system’s architecture
  - Drive implementation and generate test cases
- Developed by analysts and domain experts
Class Diagram

 Captures the vocabulary of a system
Class Diagram

- Captures the vocabulary of a system
- Built and refined throughout development

Purpose
- Name and model concepts in the system
- Specify collaborations
- Specify logical database schemas

- Developed by analysts, designers, and implementers
Object Diagram

- Captures instances and links
Object Diagram

- Shows instances and links
- Built during analysis and design

Purpose
- Illustrate data/object structures
- Specify snapshots

Developed by analysts, designers, and implementers
Component Diagram

- Captures the physical structure of the implementation
Component Diagram

- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database
- Developed by architects and programmers
Deployment Diagram

- Captures the topology of a system’s hardware
Deployment Diagram

- Captures the topology of a system’s hardware
- Built as part of architectural specification
- Purpose
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers
Sequence Diagram

- Captures dynamic behavior (time-oriented)
Sequence Diagram

- Captures dynamic behavior (time-oriented)
- Purpose
  - Model flow of control
  - Illustrate typical scenarios
Collaboration Diagram

- Captures dynamic behavior (message-oriented)

![Collaboration Diagram](image)

1: «create»
2: setActions(a, d, o)
3: «destroy»

2.1: setValues(d, 3.4)
2.2: setValues(a, "CO")
Collaboration Diagram

- Captures dynamic behavior (message-oriented)

Purpose
- Model flow of control
- Illustrate coordination of object structure and control
Captures dynamic behavior (event-oriented)
Statechart Diagram

- Captures dynamic behavior (event-oriented)

- Purpose
  - Model object lifecycle
  - Model reactive objects (user interfaces, devices, etc.)
Activity Diagram

- Captures dynamic behavior (activity-oriented)
Activity Diagram

- Captures dynamic behavior (activity-oriented)

- Purpose
  - Model business workflows
  - Model operations
Architecture and the UML

- Design View: Classes, interfaces, collaborations
- Implementation View: Components
- Process View: Active classes
- Deployment View: Nodes

Use Case View
- Use cases

Organization
- Package, subsystem

Dynamics
- Interaction
- State machine