Modeling of Software Architecture

Ergude Bao

Beijing Jiaotong University

Contents

- Architectural view model
- 4+1 view model
- Unified modeling language (UML)
- Rational’s 4+1 view model

Architectural View Model

Definition of Model

- Model is description or analogy to help visualize something that cannot be directly observed
  - Model is a simplification of reality
  - Model provides the blueprint of a system
  - Model may be structural, emphasizing the organization of the system, or it may be behavioral, emphasizing the dynamics of the system

Function of Model

- We build models so that we can better understand the system we are creating
- Through modeling, we achieve four things:
  - Models help us to visualize a system as it is or as we want it to be
  - Models give us a template that guides us in constructing a system
  - Models permit us to specify the structure or behavior of a system
  - Models document the decisions we have made

Architectural View Model

- Architectural view model is a simplified description (an abstraction) of a system with diagram, from a particular perspective or vantage point, covering particular concerns, and omitting entities that are not relevant to this perspective

4+1 View Model

About the Kruchten's Paper

- Philippe Kruchten
  - Over 16 years of experience as the leader of RUP development team in Rational corp. (now owned by IBM)
  - Valuable experiences in industry (Telecom, Air traffic control system) which he used for confirmation of his model
- The "4+1 View Model" paper:
  - 60 citations according to ACM portal site

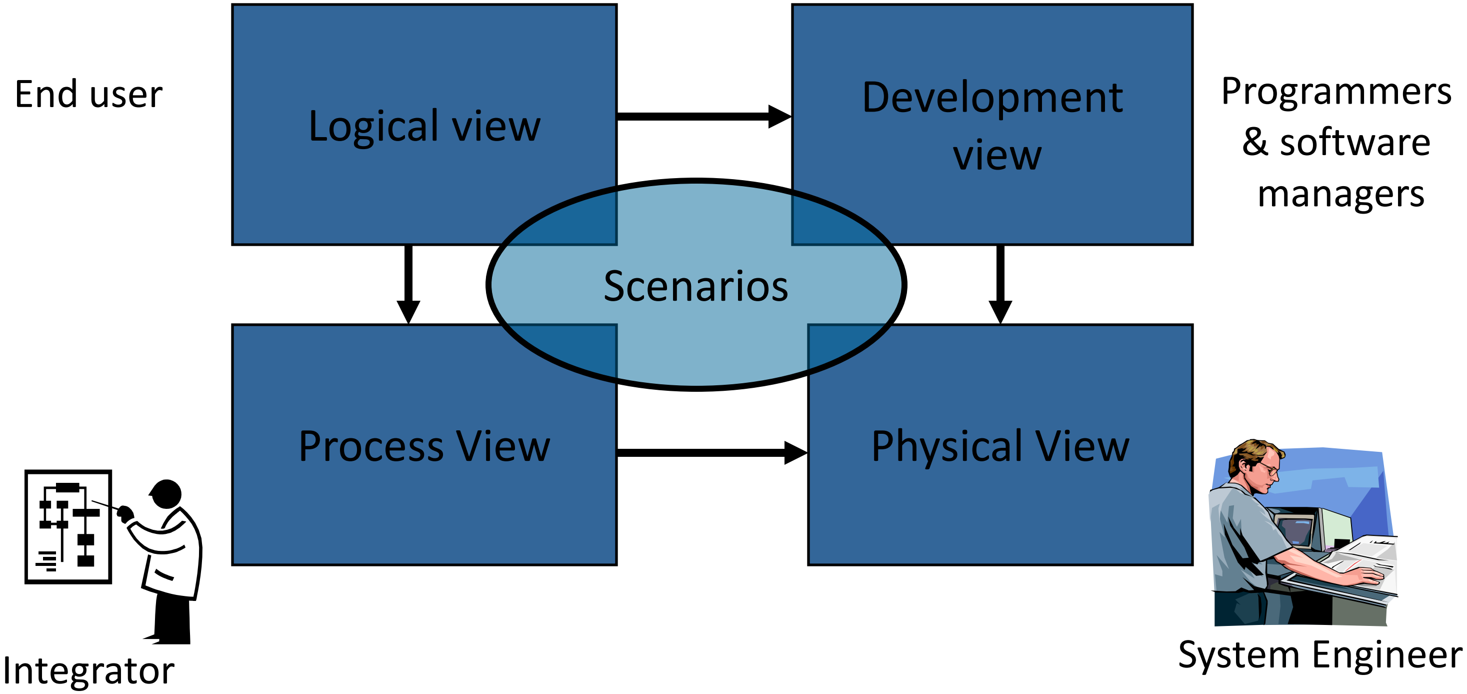
Problem

- Architecture documents over-emphasize an aspect of development (i.e. team organization) or do not address the concerns of all stakeholders
  - Various stakeholders of software system: end-user, developers, system engineers, project managers
- Software engineers struggle to represent more on one blueprint, and so architecture documents contain complex diagrams

Solution

- Using several concurrent views or perspectives, with different notations each one addressing one specific set for concerns
- 4+1 view model presented to address large and challenging architectures

4+1 View Model



End user

Logical view

Development view

Programmers & software managers

Scenarios

Process View

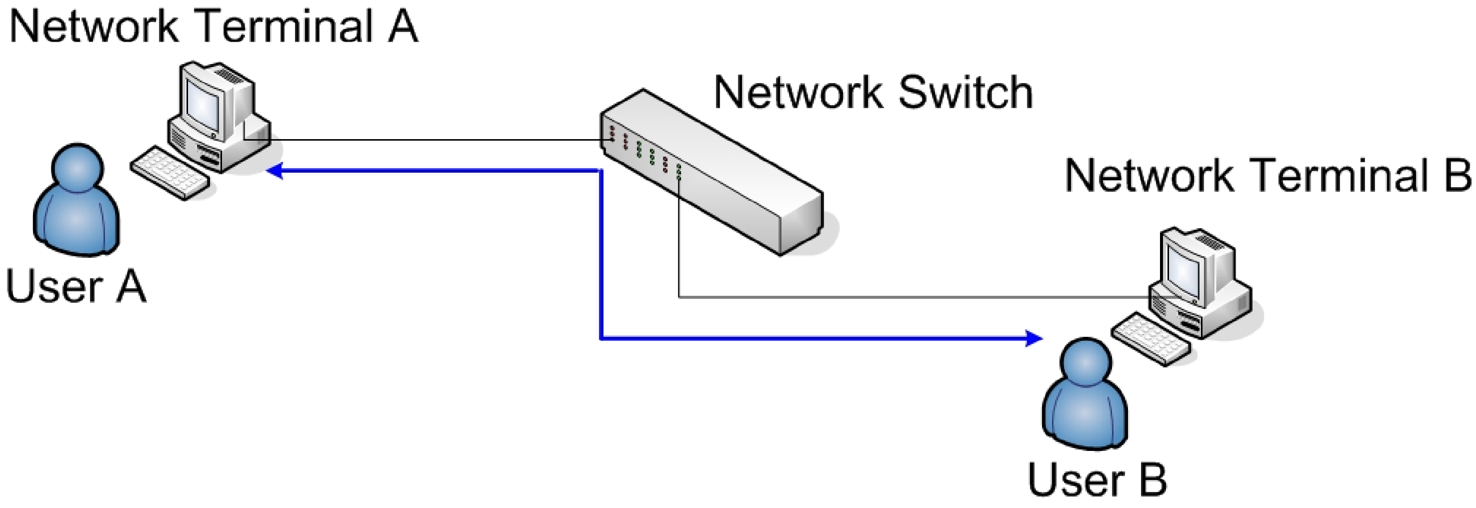
Physical View

Integrator

System Engineer

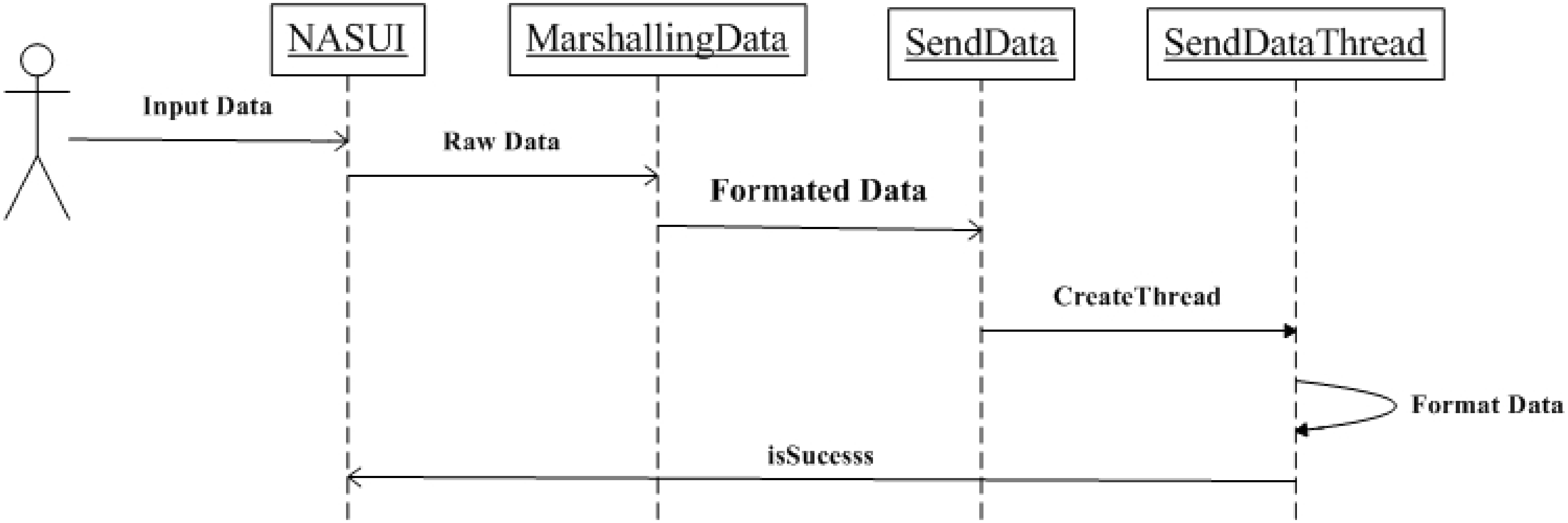
4+1 Views Model: Example

- Software Architecture of a Network Application System (NAS)
  - Terminals receive the input data from users
  - Terminal A formats the input data, and sends the formatted data to Terminal B by network
  - Terminal B parses the formatted data, and represent them to users in the screen



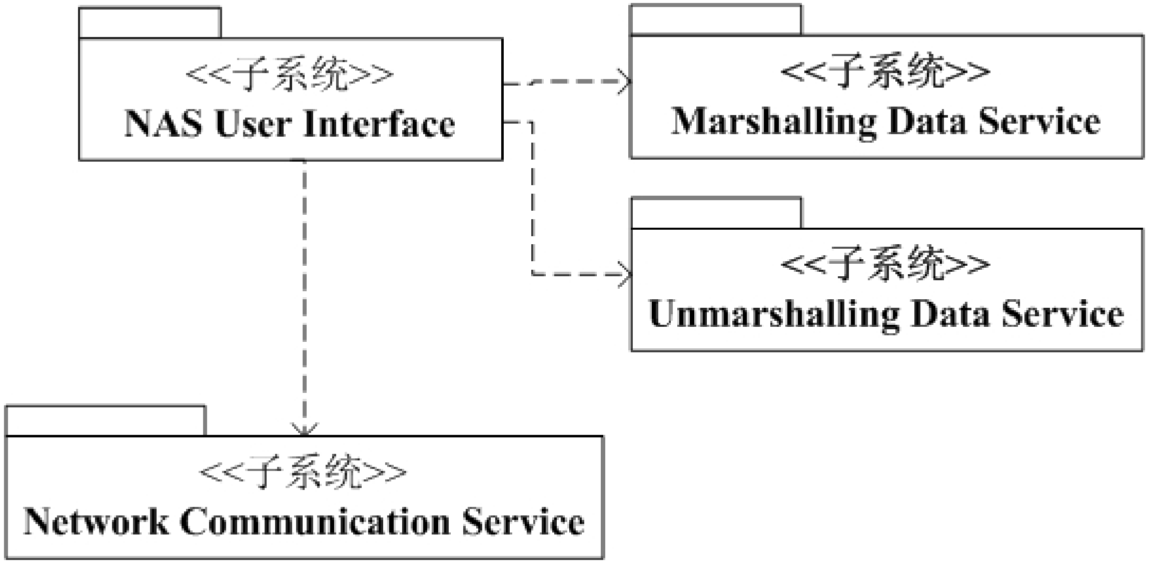
4+1 Views Model: Example

- Scenarios view
  - It is for describing the important system use cases



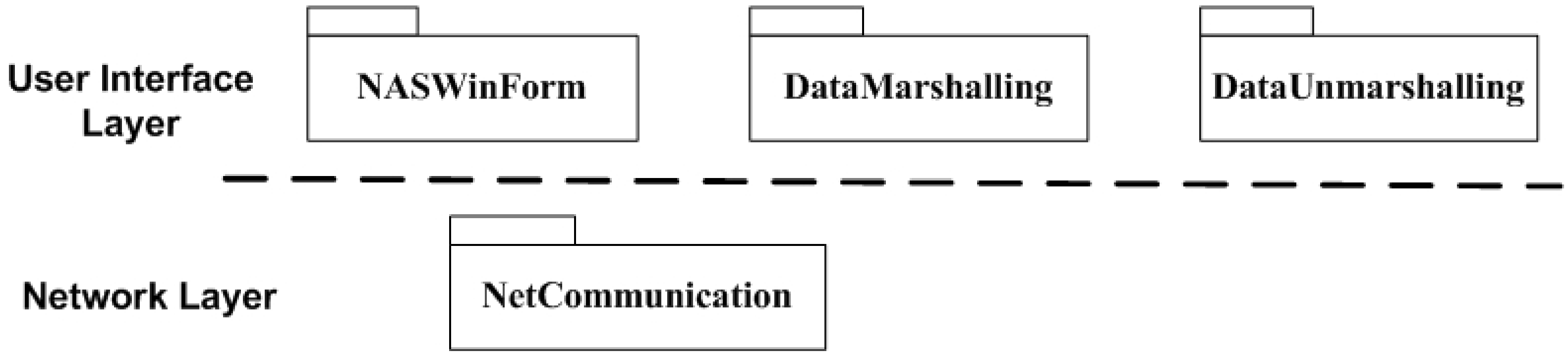
4+1 Views Model: Example

- Logic view
  - The functional abstraction of system. It mainly focused on dividing the system into several functional components and describe their functional relationships



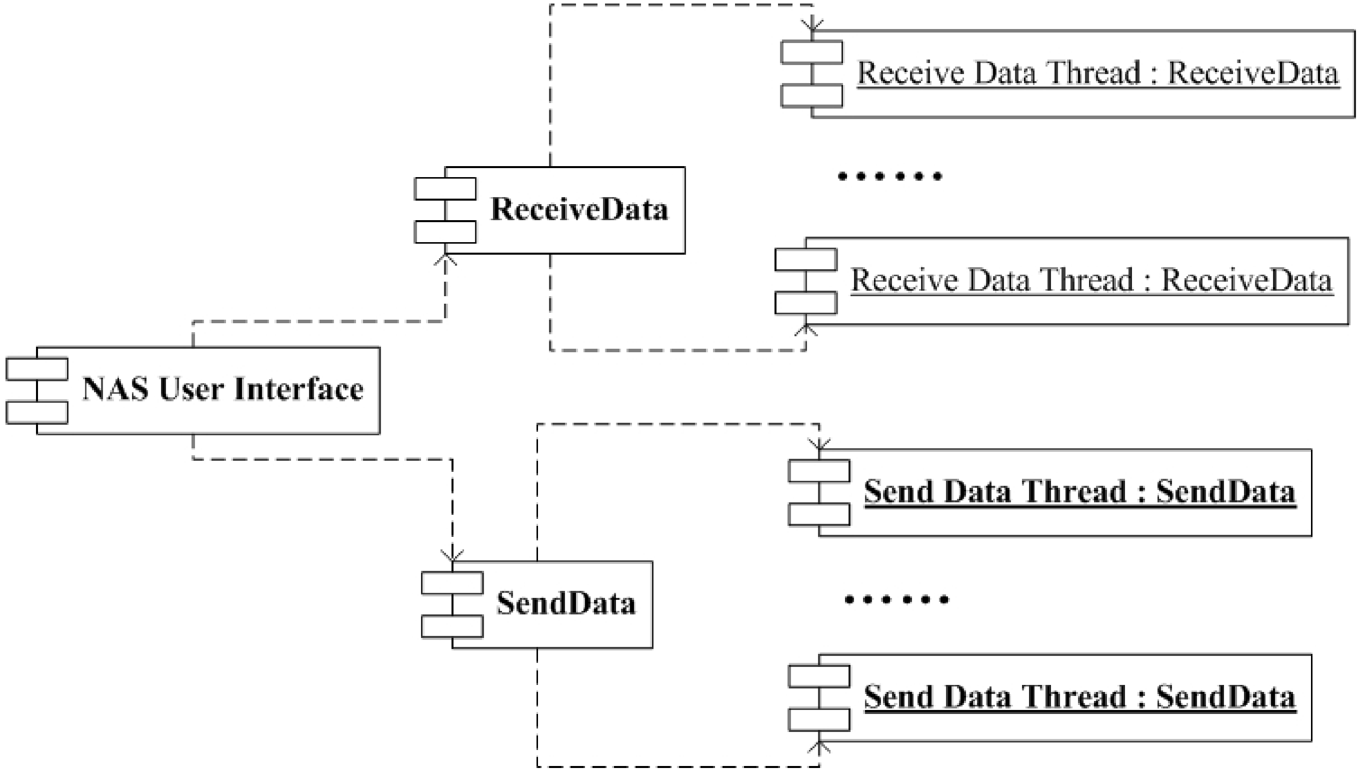
4+1 Views Model: Example

- Development view
  - – The detailed design and construction abstraction of system. It mainly gives a general structure of system for the detailed design and construction



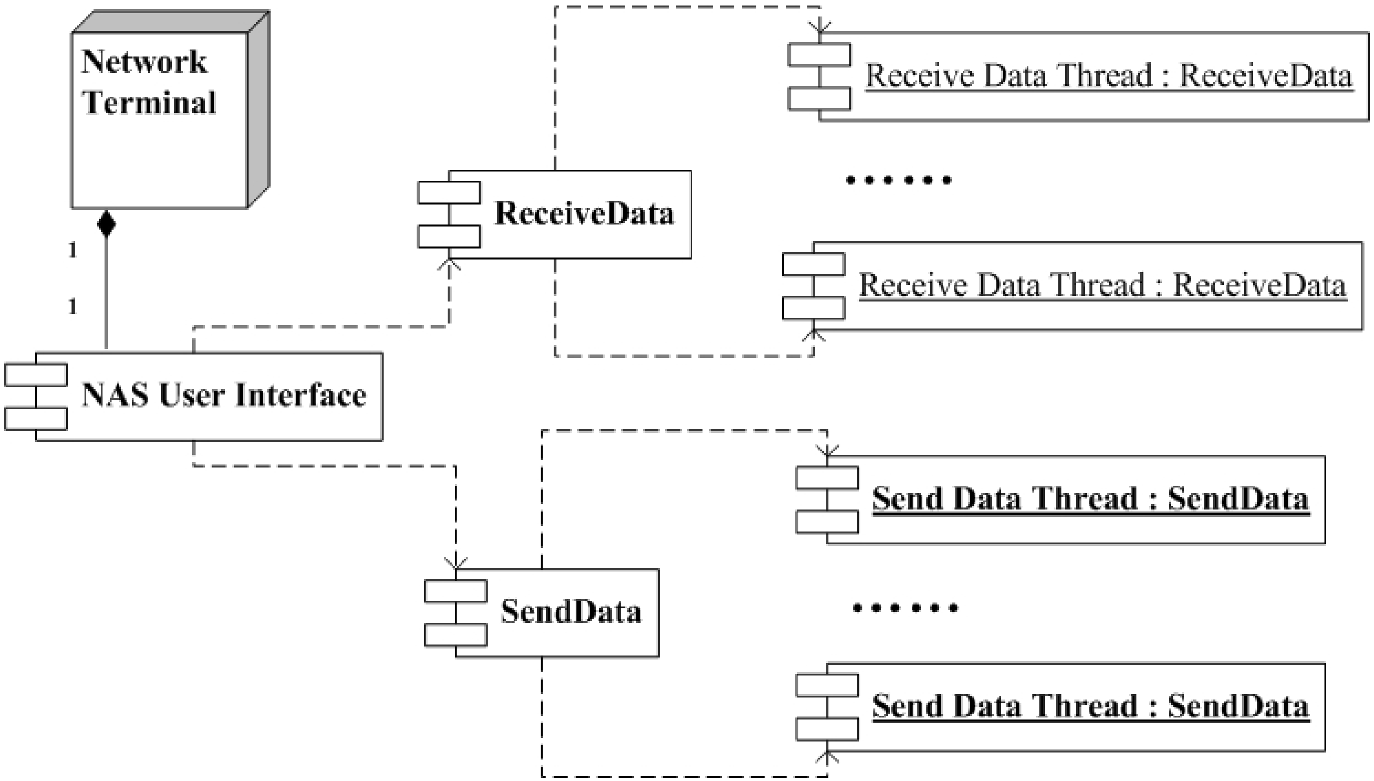
4+1 Views Model: Example

- Process view
  - It is mainly for the non-functional properties and the runtime characters of system


Receive Data Thread : ReceiveData
ReceiveData
......
Receive Data Thread : ReceiveData
NAS User Interface
Send Data Thread : SendData
SendData
......
Send Data Thread : SendData

4+1 Views Model: Example

- Physical view
  - The mapping relationship to the physical deployment environments of system


Network Terminal
1
1
NAS User Interface
ReceiveData
SendData
Receive Data Thread : ReceiveData
......
Receive Data Thread : ReceiveData
Send Data Thread : SendData
......
Send Data Thread : SendData

Unified Modeling Language (UML)

Object-Oriented Modeling Languages

- Object-oriented modeling languages started to appear sometime between the mid-1970s and the late 1980s as methodologists
- The number of object-oriented modeling methods increased from less than 10 to more than 50 during the period between 1989 and 1994
  - Grady Booch’s Booch method, Rational Software Corp.: particularly expressive during the design and construction phases of projects
  - Ivar Jacobson's Object-Oriented Software Engineering (OOSE) , Objectory Corp.: excellent support for business engineering and requirements
  - James Rumbaugh's Object Modeling Technique (OMT), General Electric Corp.: expressive for analysis of data-intensive information systems

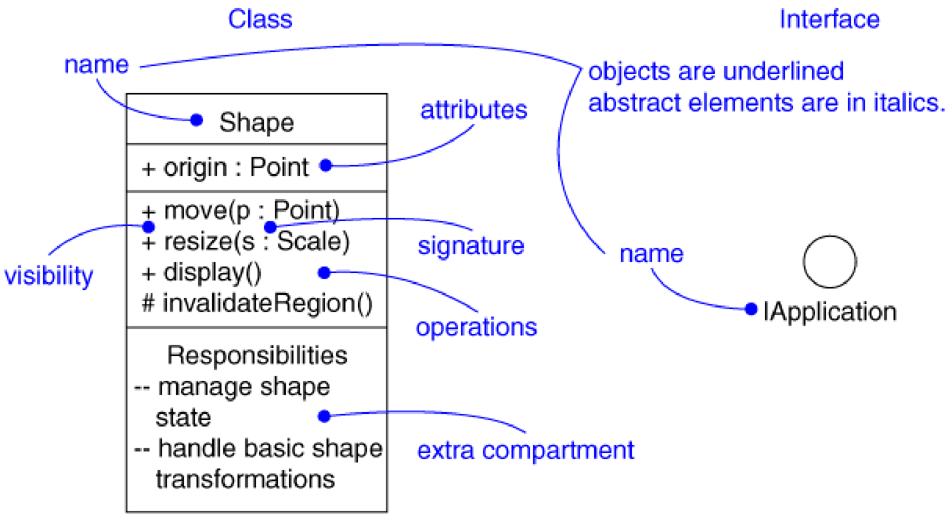
Unified Modeling Language (UML)

- Unified Modeling Language (UML; from Rational Corp.) is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

  the artifacts of a software-intensive system

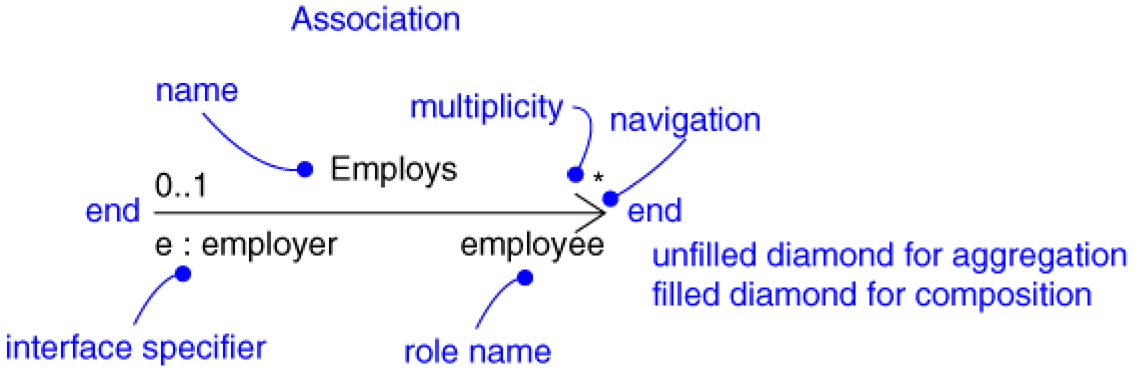
UML: Modeling Elements

- Structural elements
  - Class, interface, collaboration, use case, active class, component, node
- Behavioral elements
  - Interaction, state machine
- Grouping elements
  - Package, subsystem
- Other elements
  - Note



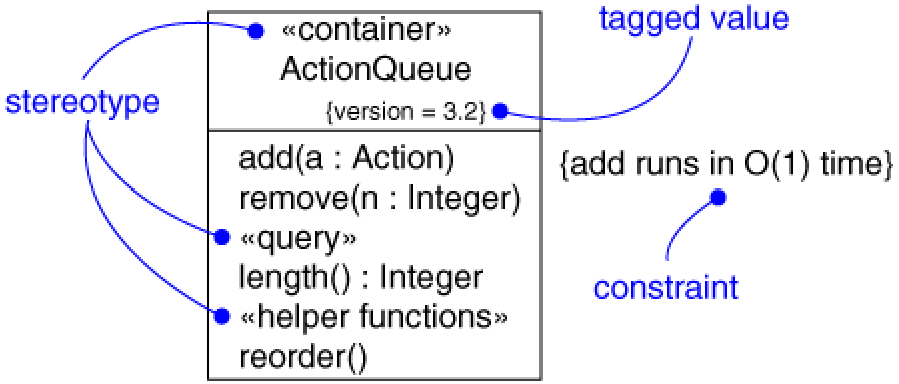
UML: Relationships

- Generalization
- Realization
- Association
  - Aggregation
  - Composition
- Dependency



UML: Extensibility Mechanisms

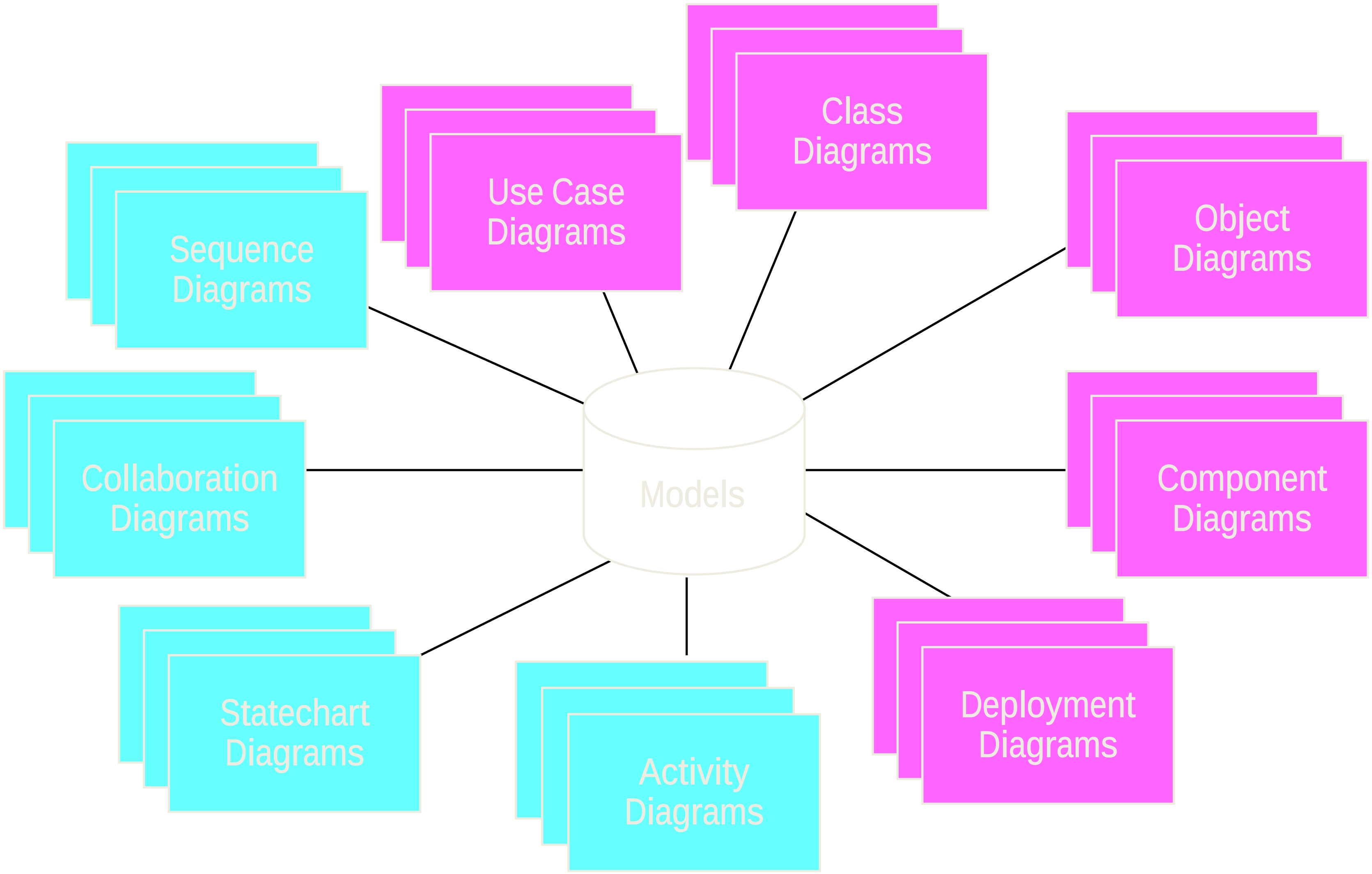
- Stereotype
- Tagged value
- Constraint



Diagrams

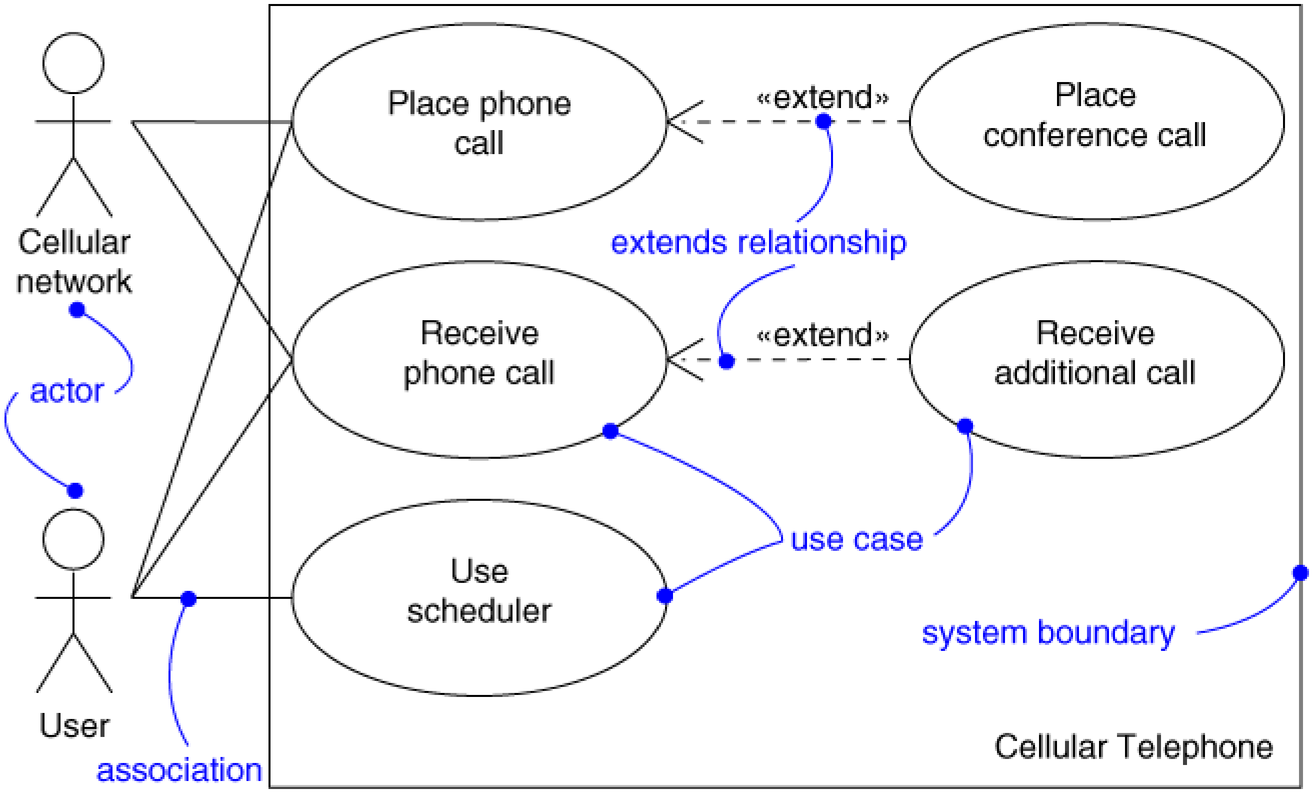
- A diagram is a view model
  - Presented from the aspect of a particular stakeholder
  - Provides a partial representation of the system
  - Semantically consistent with other views
- In UML, there are 13 standard diagrams
  - Static views: use case, class, object, component, deployment, package, composite structure
  - Dynamic views: sequence, collaboration, statechart, activity, timing, interaction

Diagrams


Class Diagrams
Use Case Diagrams
Object Diagrams
Sequence Diagrams
Models
Collaboration Diagrams
Component Diagrams
Statechart Diagrams
Activity Diagrams
Deployment Diagrams

Use Case Diagram

- Captures system functionality seen by users

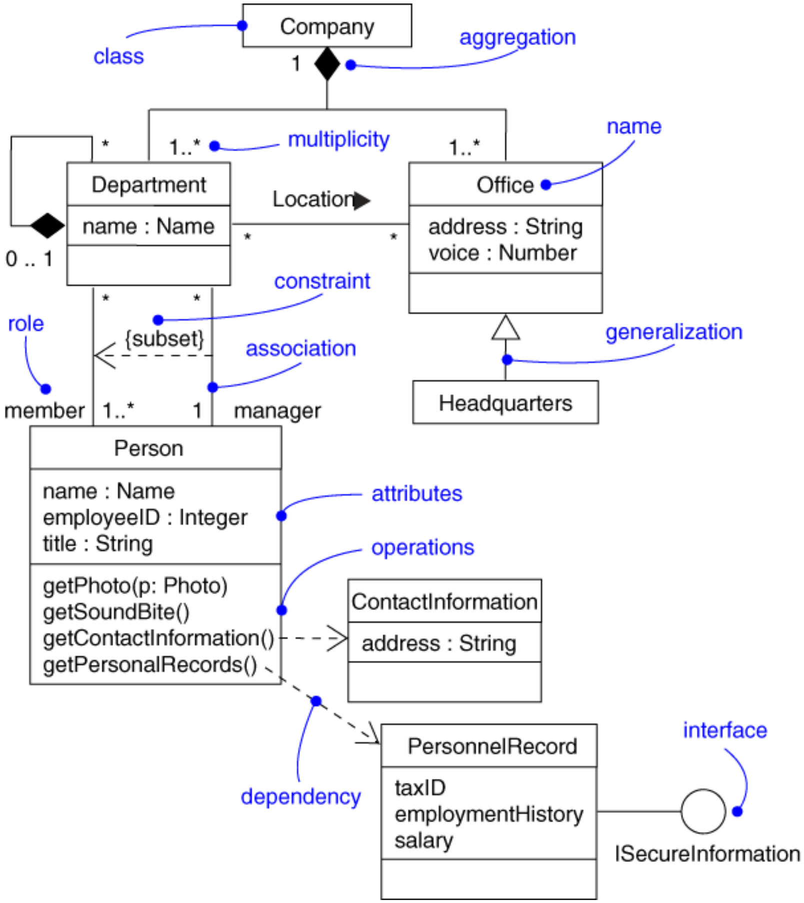


Use Case Diagram

- Captures system functionality 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

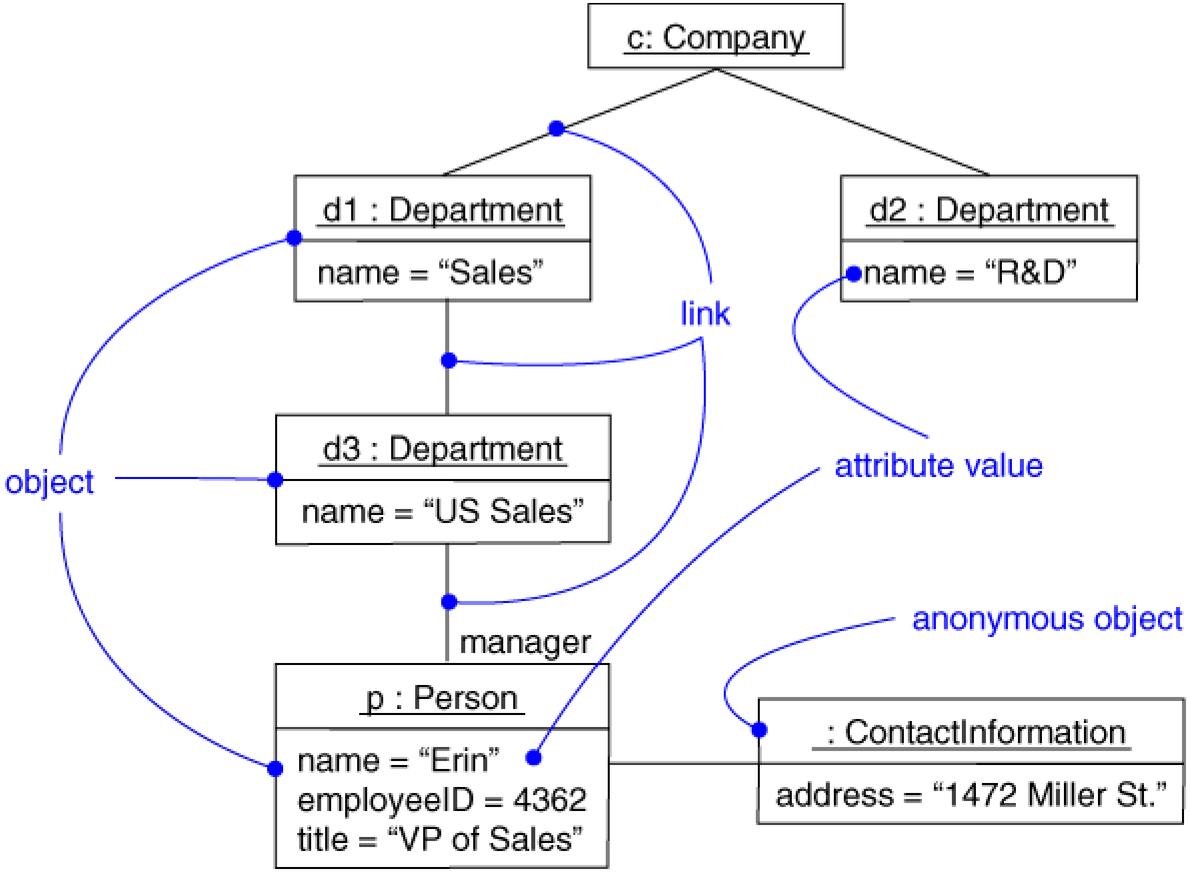


Class Diagram

- 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

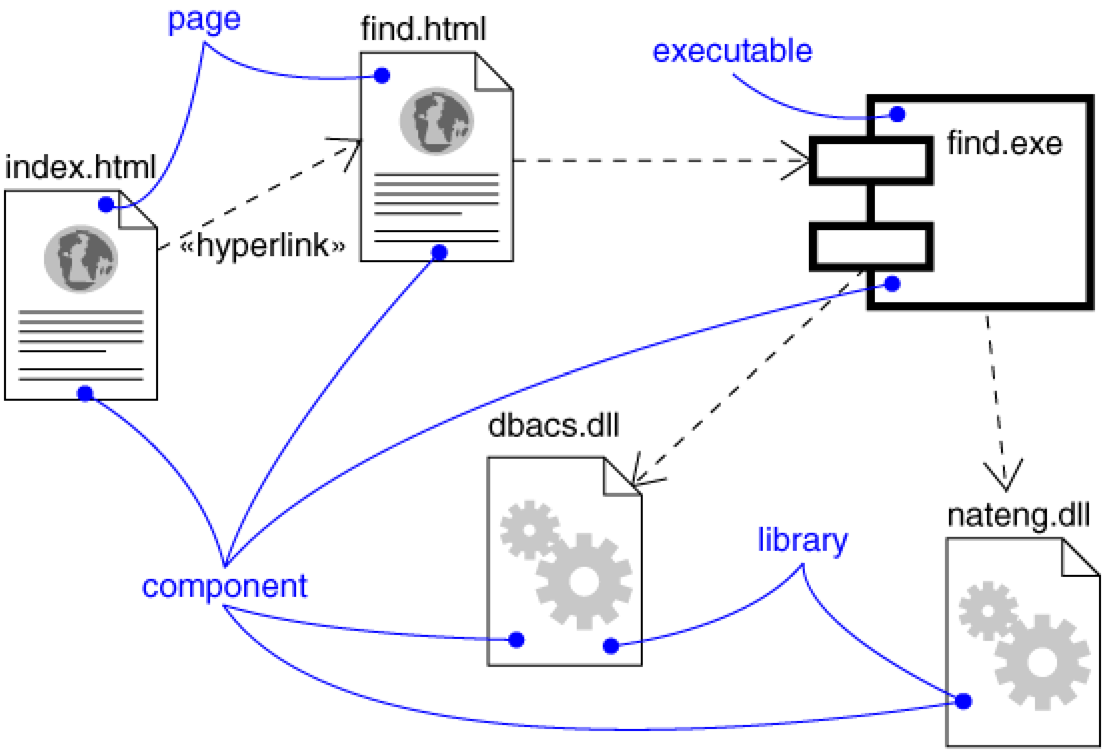


Object Diagram

- Captures 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

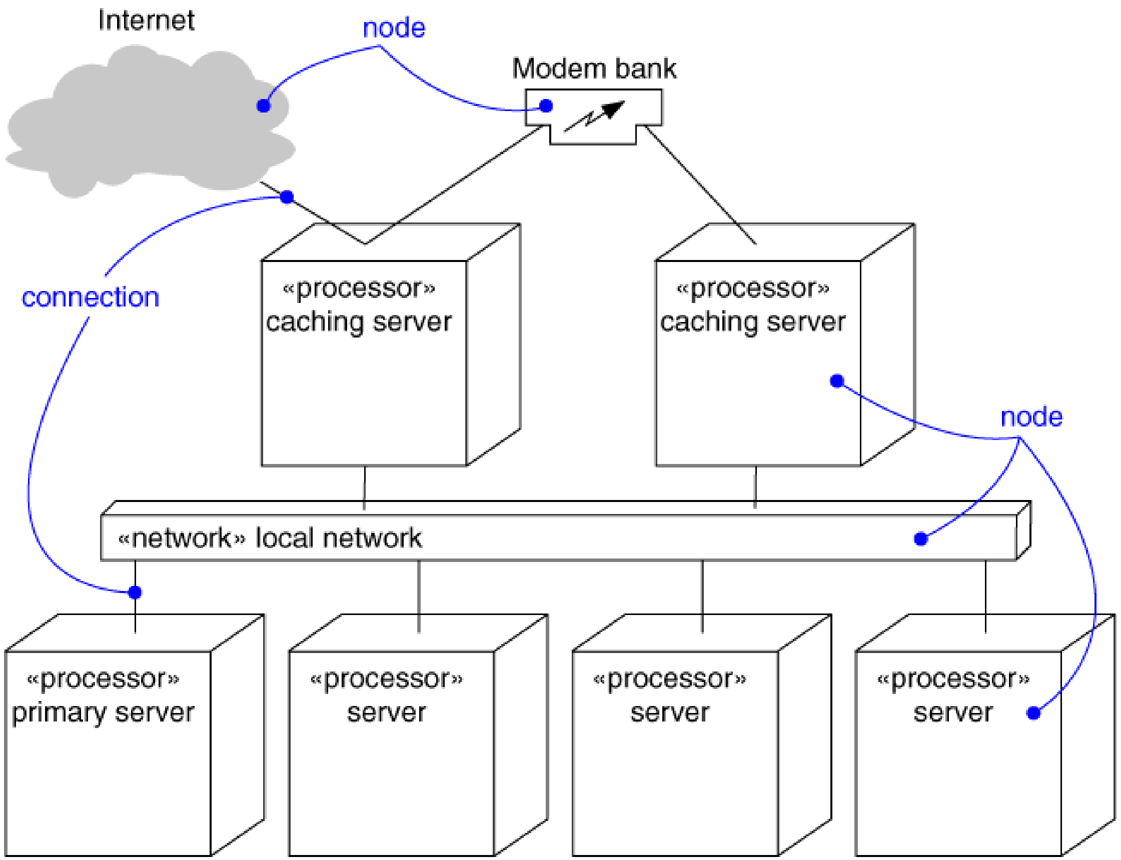


Component Diagram

- 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

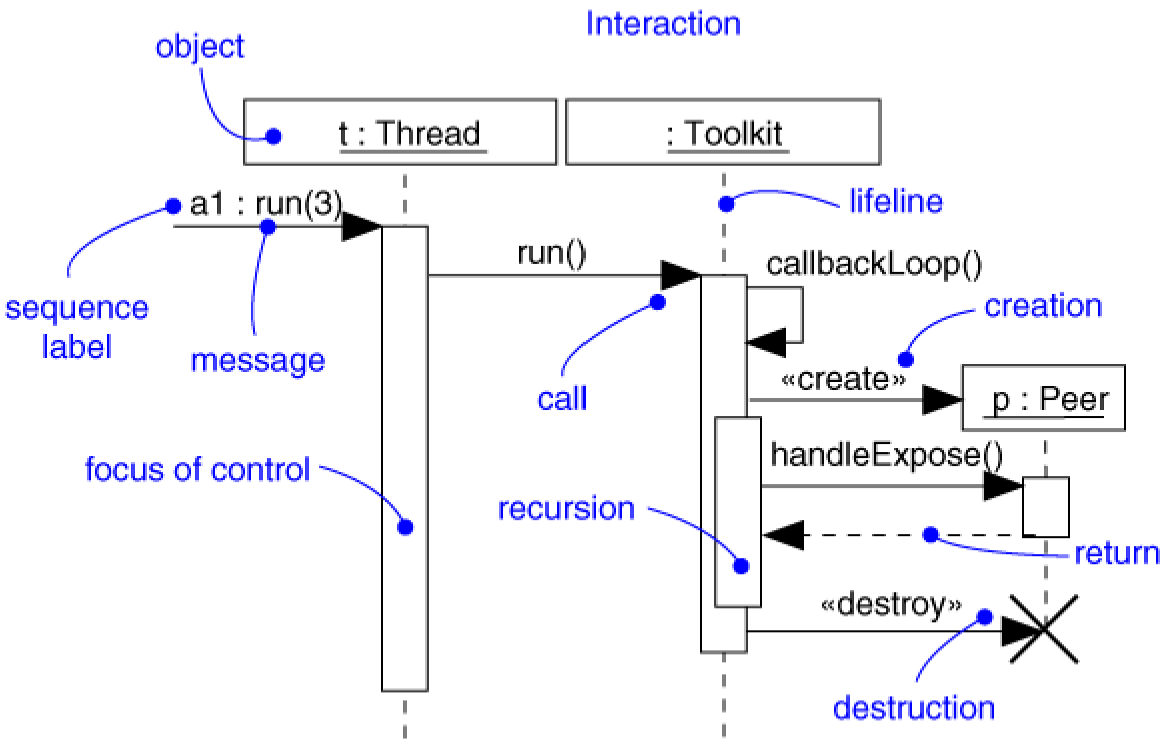


Deployment Diagram

- 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)

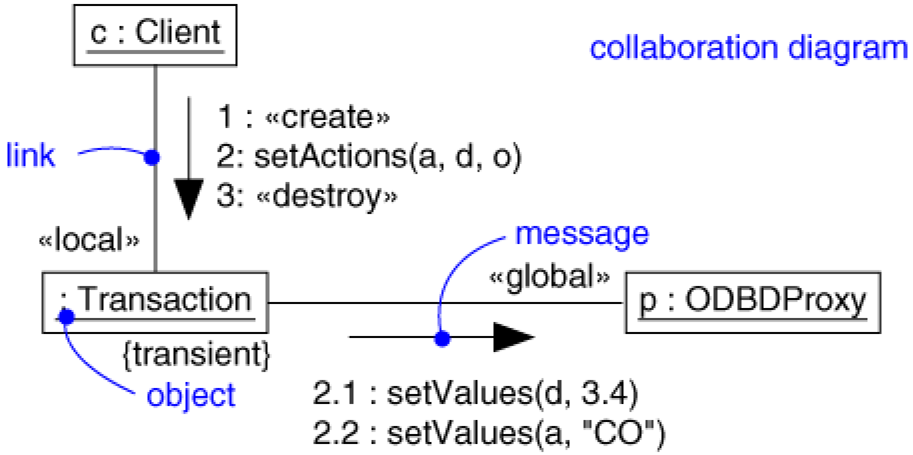


Sequence Diagram

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

Collaboration Diagram

- Captures dynamic behavior (message-oriented)

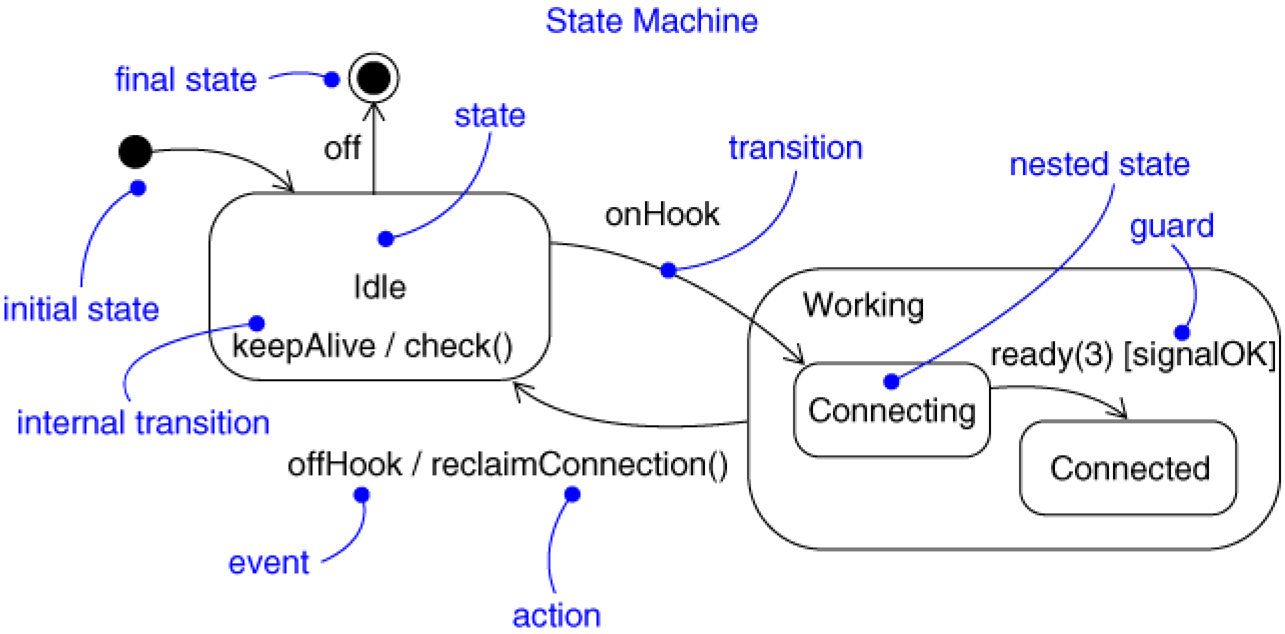


Collaboration Diagram

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

Statechart Diagram

- Captures dynamic behavior (event-oriented)

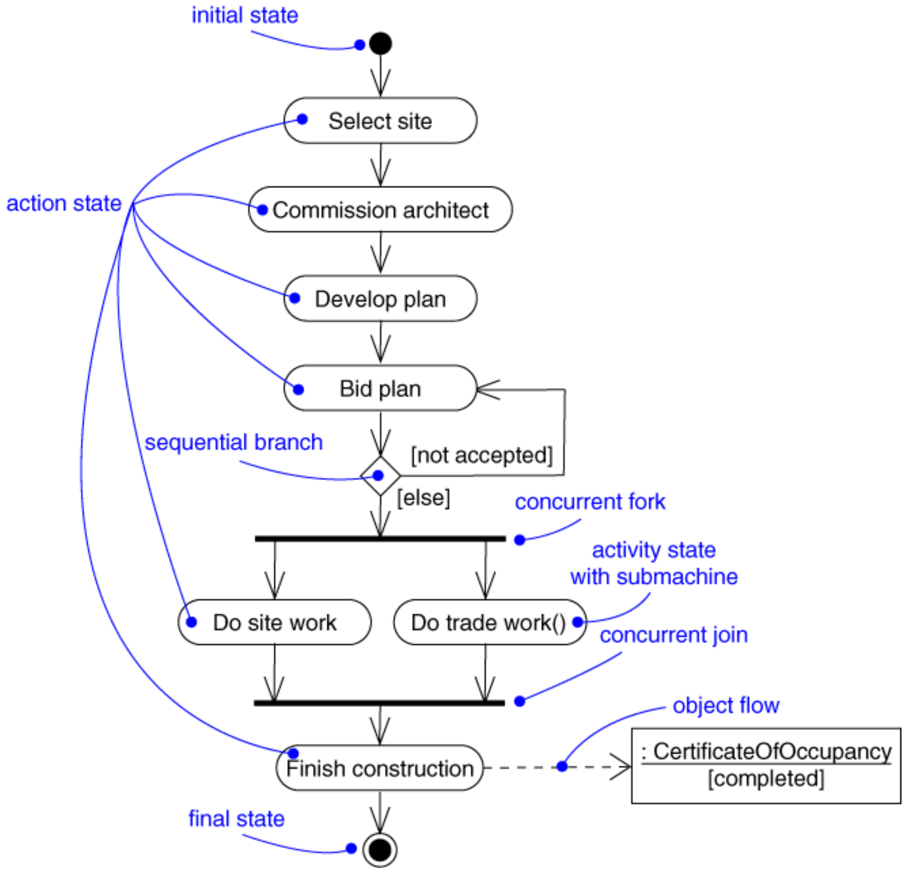


Statechart Diagram

- Captures dynamic behavior (event-oriented)
- Purpose
  - Model object lifecycle
  - Model reactive objects (user interfaces, devices, etc.)

Activity Diagram

- Captures dynamic behavior (activity-oriented)

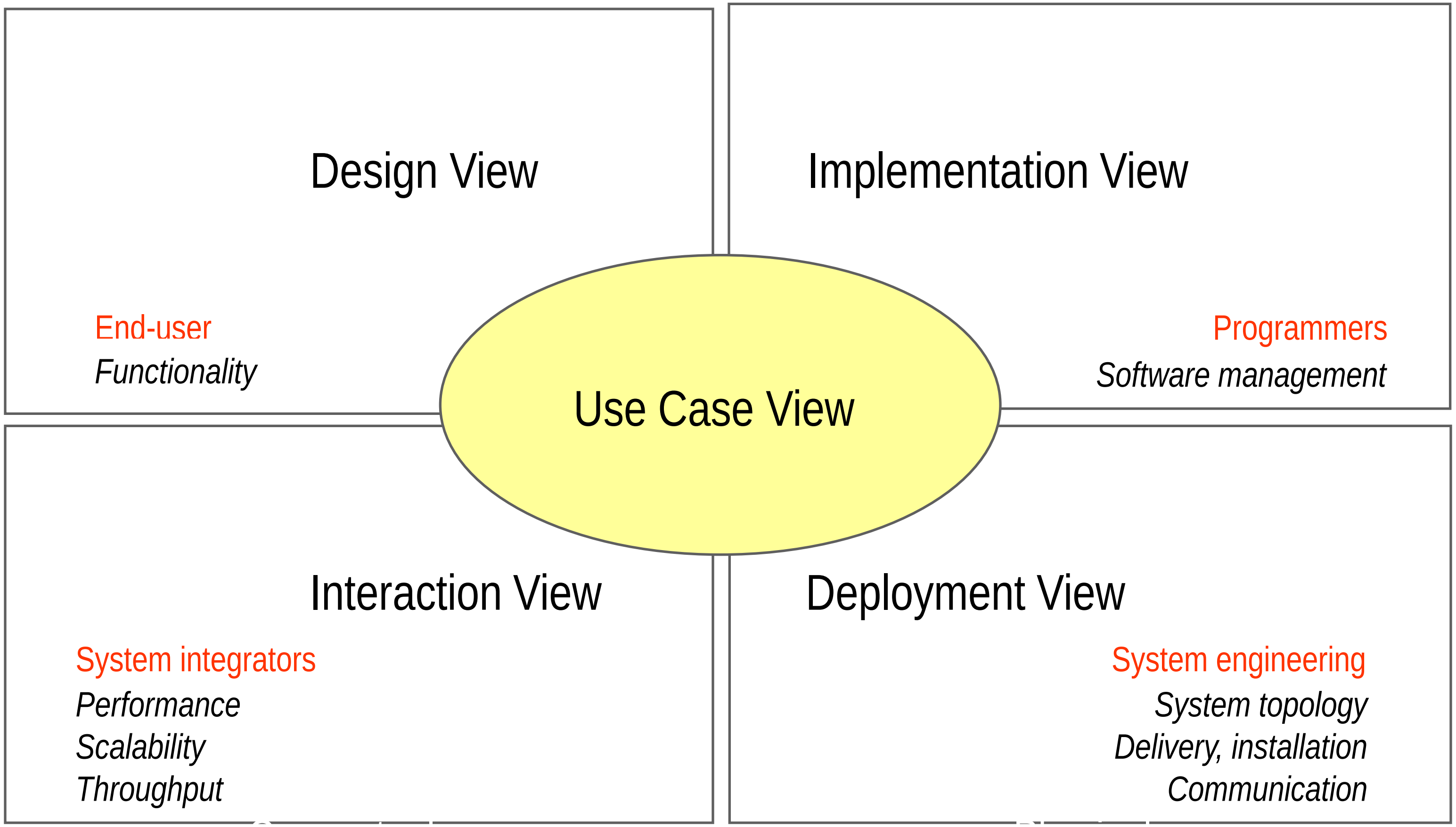


Activity Diagram

- Captures dynamic behavior (activity-oriented)
- Purpose
  - Model business workflows
  - Model operations

Rational’s 4+1 View Model

Rational's 4+1 View Model



Design View

End-user

Functionality

Implementation View

Programmers

Software management

Use Case View

Interaction View

System integrators

Performance

Scalability

Throughput

Deployment View

System engineering

System topology

Delivery, installation

Communication

Use Case View

- Encompasses the use cases that describe the behavior of the system as seen by its end users, analysts, and testers
- Exists to specify the forces that shape the system’s architecture
- Static aspects are captured in use case diagrams
- Dynamic aspects of this view are captured in collaboration diagrams, statechart diagrams, and activity diagrams

Design View

- Encompasses the classes, interfaces, and collaborations that form the vocabulary of the problem and its solution
- Primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users
- Static aspects of are captured in class diagrams and object diagrams
- Dynamic aspects are captured in collaboration diagrams, statechart diagrams, and activity diagrams

Interaction View

- Shows the flow of control among its various parts, including possible concurrency and synchronization mechanisms
- Primarily addresses the performance, scalability, and throughput of the system
- Both static and dynamic aspects are captured in the same kinds of diagrams as the design view but with a focus on the active classes that control the system and the message that flow between them

Implementation View

- Encompasses the artifacts that are used to assemble and release the physical system
- Primarily addresses the configuration management of the system's releases, made up of somewhat independent components that can be assembled in various ways to produce a running system
- Static aspects are captured in artifact diagrams
- Dynamic aspects are captured in collaboration diagrams, statechart diagrams, and activity diagrams

Deployment View

- Encompasses the nodes that form the system's hardware topology, upon which the system executes
- Primarily addresses the distribution, delivery, and installation of the parts that make up the physical system
- Static aspects are captured in deployment diagrams
- Dynamic aspects are captured in collaboration diagrams, statechart diagrams, and activity diagrams

Rational's 4+1 View Model

- Not all systems require all views
  - Single processor: drop deployment view
  - Single process: drop interaction view
  - Very small program: drop implementation view
- Adding views
  - Data view, security view