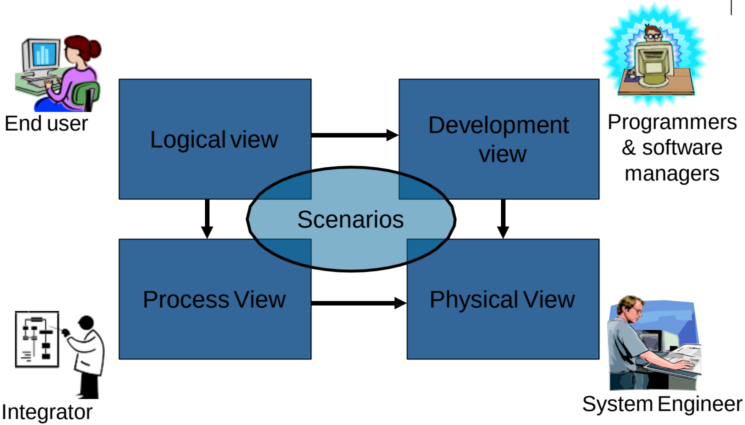
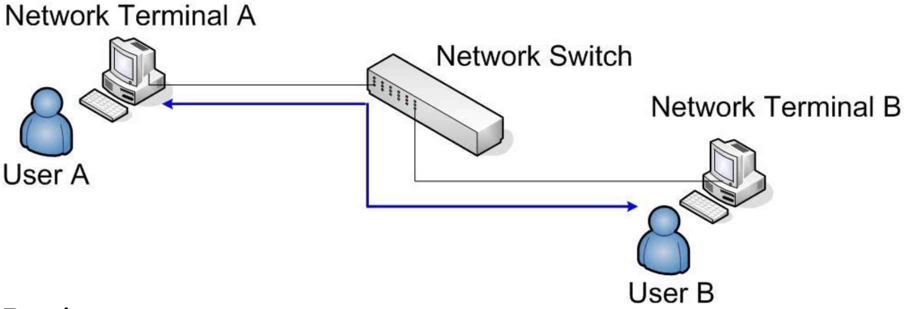
Software Architecture Modeling Tools and Languages

4+1 View Model of Architecture



"4+1 Views" Model

- A Case: Software Architecture of a Network Application System---- NAS

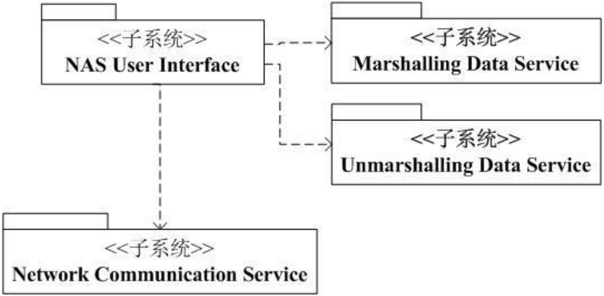


Requirements:
- Terminals receive the input data from users.
- Terminal A formats the input data, and sends the formatted data to Terminal B by network.
- Terminal B parses the formatted data, and represent them to users in the screen.

“4+1 Views” Model

- Logic View

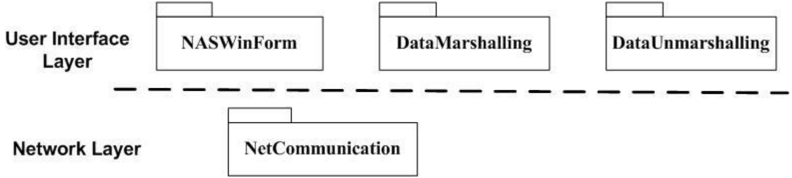
The functional abstraction of system. It mainly focused on dividing the System into several functional components and describe their functional relationships.



"4+1 Views" Model

- Development View

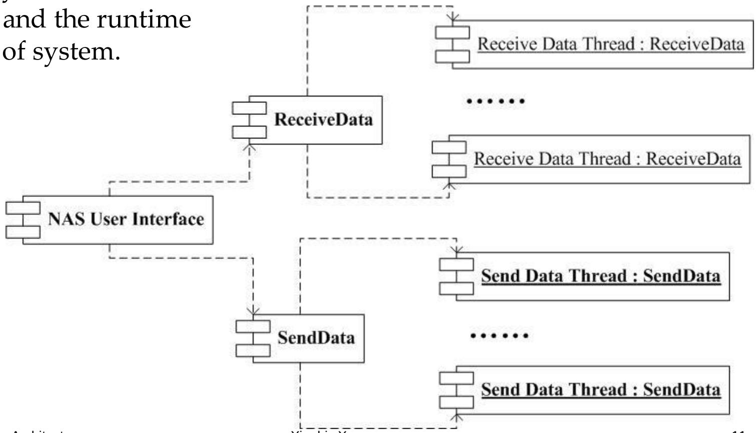
The detailed design and construction abstraction of system. It mainly gives a general structure of system for the detailed design and construction.



"4+1 Views" Model

- Process View

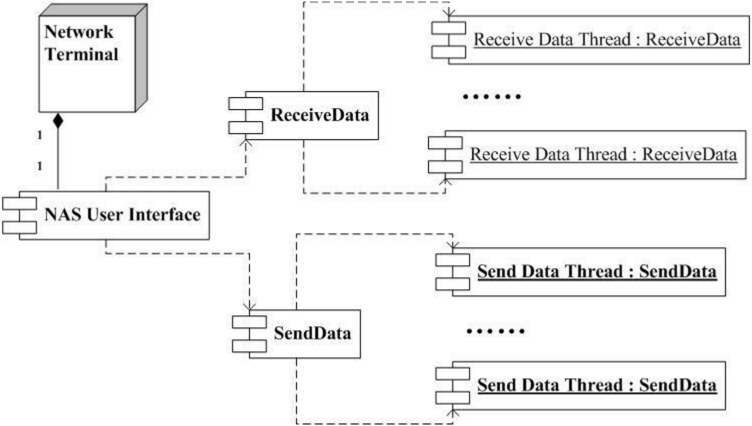
It mainly for the non-functional properties and the runtime characters of system.



"4+1 Views" Model

- Physical View

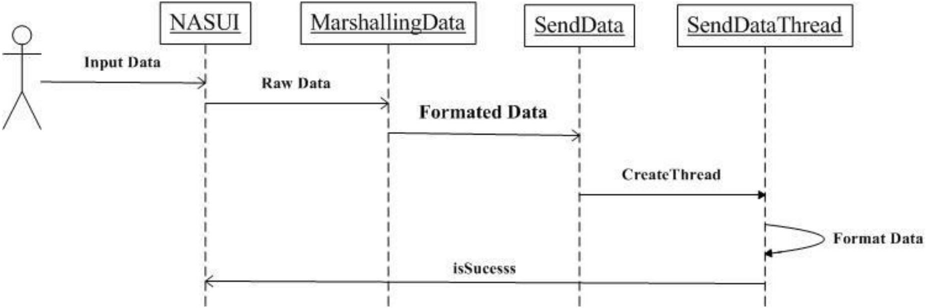
The mapping relationship to the physical deployment environments of system.



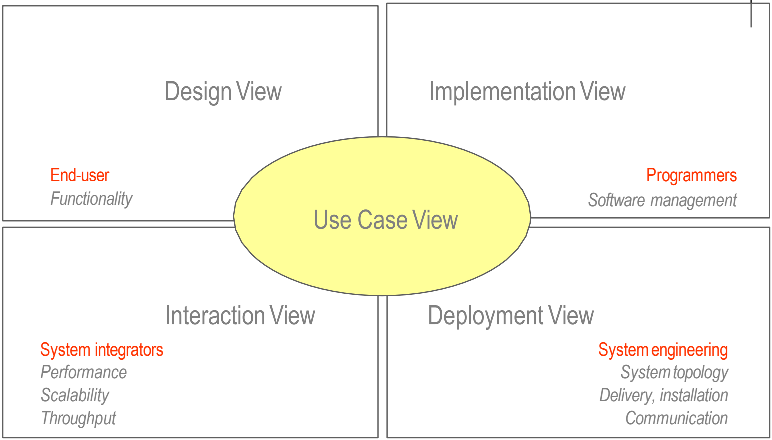
"4+1 Views" Model

It is for describing the important system use cases.

- Scenarios View



Rational's 4+1 View Model



Rational's 4+1 View Model

- 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

Rational's 4+1 View Model

- Use Case View/Scenarios
  - The ***use case view*** of a system encompasses the use cases that describe the behaviour of the system as seen by its end users, analysts, and testers.
  - The view exists to specify the forces that shape the system's architecture.
- With UML:
  - the static aspects of this view are captured in ***use case diagrams***.
  - the dynamic aspects of this view are captured in ***interaction diagrams***, ***state diagrams***, and ***activity diagrams***.

Rational's 4+1 View Model

- Design View/Logic View
  - The ***design view*** of a system encompasses the classes, interfaces, and collaborations that form the vocabulary of the problem and its solution.
  - The view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users.
- With UML:
  - the static aspects of this view are captured in ***class diagrams*** and ***object diagrams***.
  - the dynamic aspects of this view are captured in ***interaction diagrams***, ***state diagrams***, and ***activity diagrams***.

Rational's 4+1 View Model

- Interaction View/Process View
  - The ***interaction view*** of a system shows the flow of control among its various parts, including possible concurrency and synchronization mechanisms.
  - The view primarily addresses the performance, scalability, and throughput of the system.
- With UML, the static and dynamic aspects of this view 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.

Rational's 4+1 View Model

- Implementation View/Development View
  - The ***implementation view*** of a system encompasses the artifacts that are used to assemble and release the physical system.
  - The view 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.
- With UML:
  - the static aspects of this view are captured in ***artifact diagrams***.
  - the dynamic aspects of this view are captured in ***interaction diagrams, state diagrams, and activity diagrams.***

Rational's 4+1 View Model

- Deployment View/Physical View
  - The ***deployment view*** of a system encompasses the nodes that form the system's hardware topology, upon which the system executes.
  - The view primarily addresses the distribution, delivery, and installation of the parts that make up the physical system.
- With UML:
  - the static aspects of this view are captured in ***deployment diagrams***.
  - the dynamic aspects of this view are captured in ***interaction diagrams, statechart diagrams, and activity diagrams***.

Modeling Tools and Languages

- 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 models methods increased from less than 10 to more than 50 during the period between 1989 and 1994.

- Grady Booch's Booch method ---Rational Software Corporation)
- Ivar Jacobson's Object-Oriented Software Engineering (OOSE) ---Objectory
- James Rumbaugh's Object Modeling Technique (OMT) --- General Electric

In simple terms:

●The Booch method was particularly expressive during the design and construction phases of projects.

●OOSE provided excellent support for business engineering and requirements analysis.

● OMT-2 was expressive for analysis of data-intensive information systems.

Overview of the UML

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

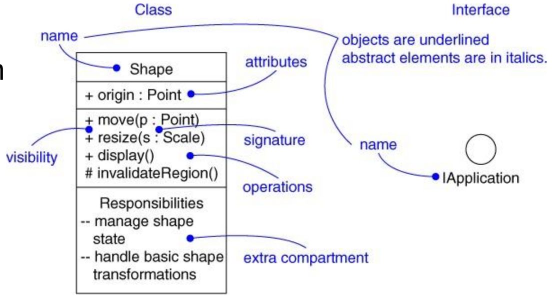
  the artifacts of a software-intensive system

Overview of the UML

- Modeling elements
- Relationships
- Extensibility Mechanisms
- Diagrams

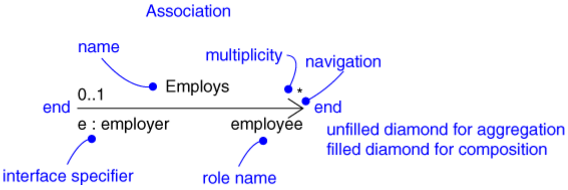
Modeling Elements

- Structural elements
  - class, interface, collaboration, use case, active class, component, node
- Behavioral elements
  - interaction, state machine
- Grouping elements
  - package, subsystem
- Other elements
  - note



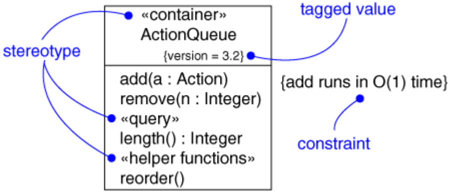
Relationships

- Dependency
- Association
- Generalization
- Realization



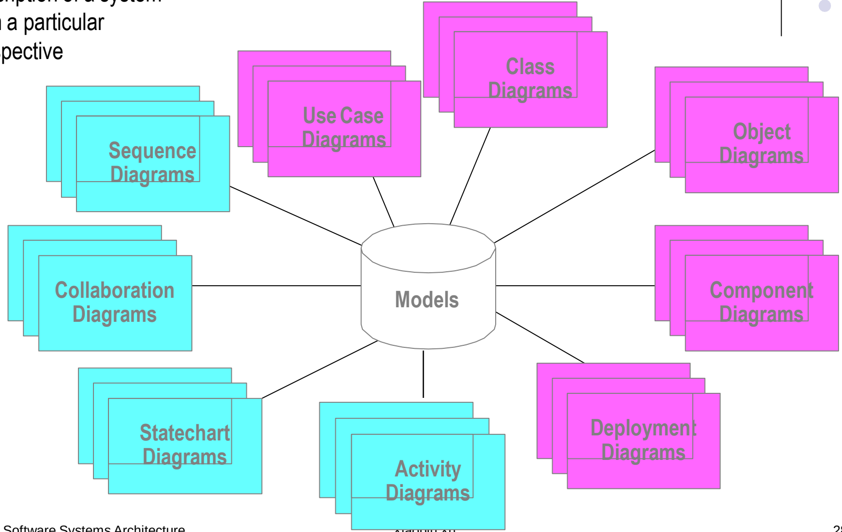
Extensibility Mechanisms

- Stereotype
- Tagged value
- Constraint



Models, Views, and Diagrams

A ***model*** is a complete description of a system from a particular perspective

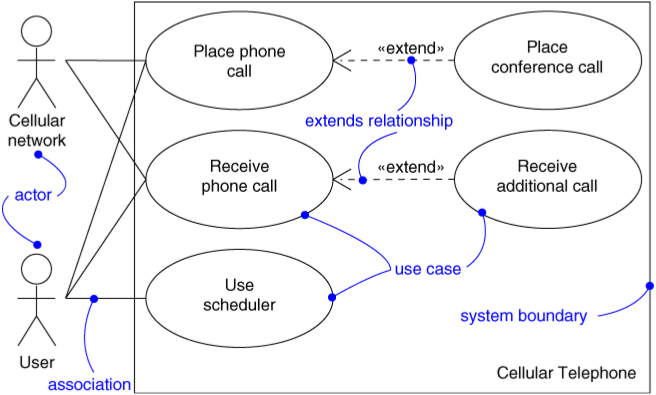


Diagrams

- 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 13 standard diagrams
  - Static views: use case, class, object, component, deployment, package, composite structure
  - Dynamic views: sequence, communication, state machine, activity, timing, interaction overview

Use Case Diagram

- Captures system functionality as seen by users

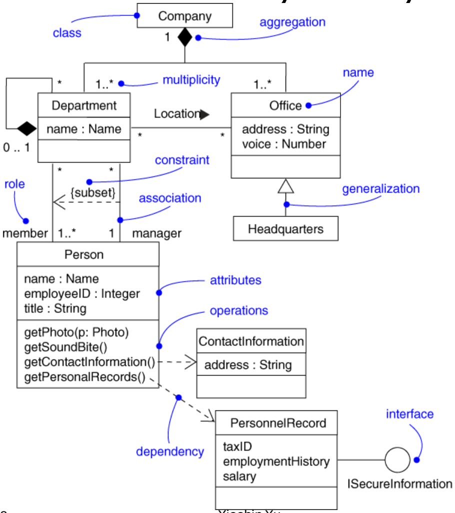


Use Case Diagram

- 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

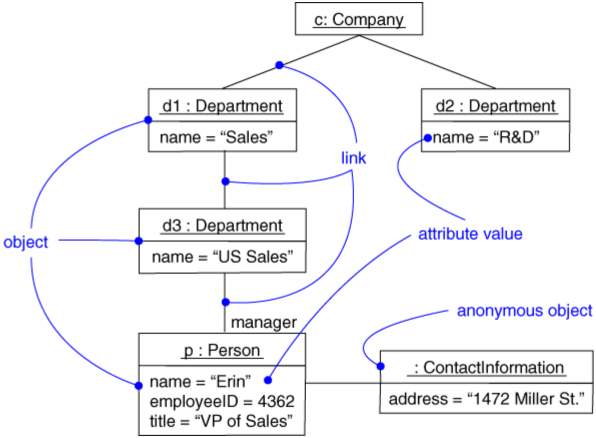


Class Diagram

- 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

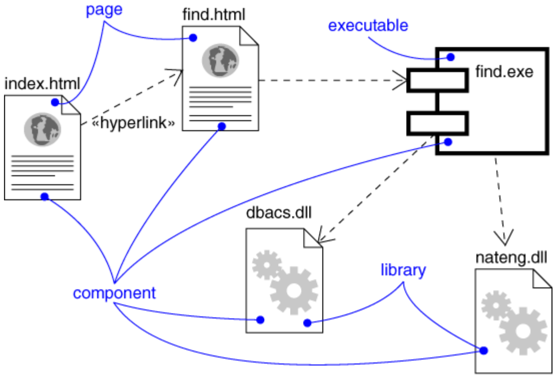


Object Diagram

- 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

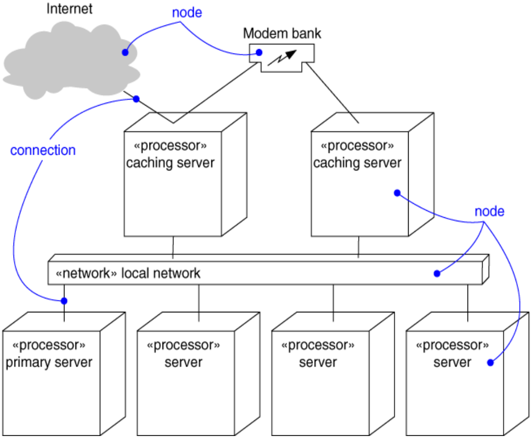


Component Diagram

- 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

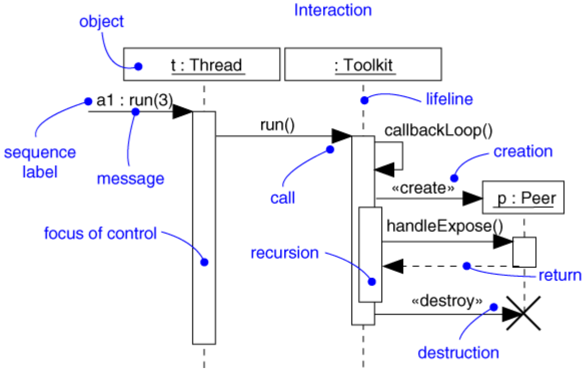


Deployment Diagram

- 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)

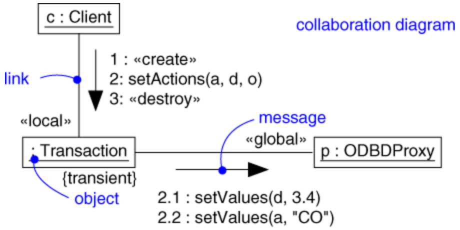


Sequence Diagram

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

Communication Diagram

- Captures dynamic behavior (message-oriented)

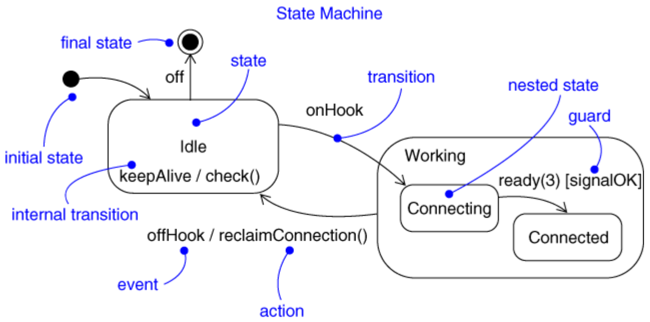


Communication Diagram

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

State Diagram

- Captures dynamic behavior (event-oriented)

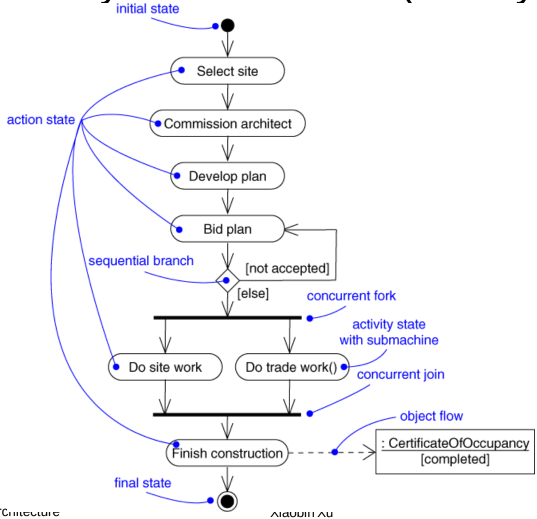


State Diagram

- 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