COMP3027J
Software Architecture
Software Architecture Description

DENG, YONGJIAN

Faculty of Computer Science, BJUT Data

Mining & Security Lab (DMS Lab)

Outline

1. Software Architecture Description

2. Software Architecture Modelling

Outline

1. Software Architecture Description

What is Software Architecture Description

Software architecture consists of a certain form of structural elements, i.e., a collection of components. Including processing components, data components and connection components.

- The processing component is responsible for processing data.
- The data component represents the processed information.
- The connection component is responsible for combining and connecting different components.

Rational Rose

北京工業大學 BEIJING UNIVERSITY OF TECHNOLOGY

What is Software Architecture Description

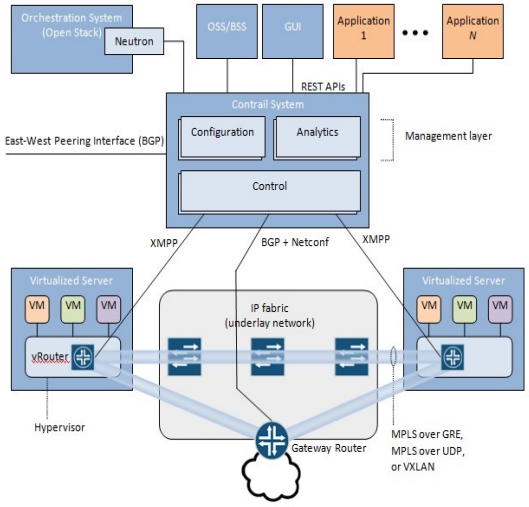
Software architecture is the blueprint of the developing system.

- It defines work distribution.
- It is the primary carrier of quality attributes.
- It is the best tool for early analysis.
- It is key to post-deployment maintenance and mining.
- Describing the software system structure is the most important step in creating an architecture.
- Whether now or 20 years from now, software architecture documents can represent the design ideas of software architecture.

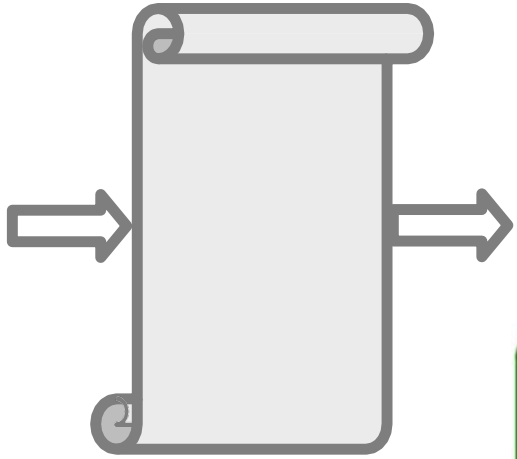
Software Architecture Description - Functions

Describing software architecture is the basis for understanding architecture design ideas and communicating how to use architecture to guide system construction.

- How to use architecture if you can't understand it?
- How to understand if you can't communicate?



Design Decisions



Software Architecture Description

Communication

Software Architecture Description - When

When we browse our company's website, interact with customers, or conduct software architecture evaluation, we need to obtain a description document about the software architecture as a basis for communication.

Web Surfing

Communication

Evaluation

Software Architecture Description - Method

Traditional Description Method used by the industry has certain limitations, and professional software architecture description documents can solve this problem.

In the industrial world, the description method or problem used is generally:

- "Use UML."
- "Frames and lines."
- "What else do I need besides my Rose class diagram?"
- "Natural language isn't very good."
- "How to record changed information?"

Software Architecture Description - Principle

Basic principles for establishing description documents

1. Write from the reader's perspective.
2. Avoid unnecessary duplication.
3. Avoid ambiguity.
4. Use standard organizational structures.
5. Record the reasons.
6. Keep documents current but not updated frequently
7. Review whether the documents meet the requirements

北京工業大學
BEIJING UNIVERSITY OF TECHNOLOGY

Software Architecture Description - Principle

1. Write from the reader's perspective.

- **What do readers hope to know when reading the document?**

  1. Make information easy to find!

  2. Your readers will appreciate your efforts and be more likely to read your document.

- **File marks written for the convenience of authors:**

  1. Stream of consciousness: the sequence of events in the author's mind

  2. Execution flow: the order in which things happen in a computer

Software Architecture Description - Principle

2. Avoid unnecessary duplication.

- Every information should be recorded accurately.
- This makes the document easier to use and change.
- Repetition often causes confusion because information is repeated in slightly different ways and you don't know which one is correct?

Software Architecture Description - Principle

3. Avoid ambiguity.

- **Documents are used to exchange information and ideas. If misunderstandings are caused, the document will be invalid.**
  1. Precisely defined symbols/language help avoid ambiguity.
  2. Building a dictionary helps avoid ambiguity.

- **If the document uses a graphical language**
  1. Need to create a legend
  2. The legend needs to formally define or give the meaning of each symbol.

Software Architecture Description - Principle

4. Use standard organizational structures.

- Establish a standard organizational structure, make sure documents follow it, and make sure readers know what it is.
- Don't leave incomplete sections blank mark them "pending".

5. Record the reasons.

- Why made these design decisions?
- Documenting the fundamentals can save a lot of time.

Software Architecture Description - Principle

6. Keep documents current but not updated frequently.

- Documents that are incomplete, out of date, do not reflect facts, and do not adhere to the requirements of their own organizational structure should not be used.
- Select key points in the development plan when updating documentation
- Follow a release strategy that makes sense for the project.

7. Review whether the documents meet the requirements.

- The information selected to best meet our identified objectives?
- Who are the readers of architecture documents?
- What is their purpose for using the document?

Outline

2. Software Architecture Modelling

Software Architecture Description - View

Definition of view.

- **The concept of "view" provides us with the principles for describing architecture:**
  1. Record relevant opinions
  2. Then add information that applies to multiple views
  3. Thus tying ideas together

- **Architectural elements** – a set of actual elements present in software or hardware

- **View** – a representation of a coherent set of architectural elements written and read by system stakeholders

Software Architecture Description - View

Definition of view.

- A view is a representation of a set of architectural elements and their relationships.

- But not all architectural elements are covered.

- The view binds the element types and relationship types of interest when describing the architecture and displays them.

Software Architecture Description - View

Definition of view.

- Views are a complex management method.

- Each view answers a different question about the architecture.
  1. What are the main execution units and data storage?
  2. How do other software use this software?
  3. How does data flow through the system?
  4. How to deploy software to hardware?

Software Architecture Description - View

Types of view.

- **Exploded View –** Shows the modules associated with the "Associated Submodule" relationship

- **Usage view –** displays modules associated with a "Using" relationship (i.e. one module consumes a service provided by another module)

- **Hierarchical view –** shows groups of modules divided into related and coherent functions. Each group represents a layer in the overall structure.

- **Class/Generalization View –** displays modules called classes, which are related by "inheritance" or "instance" of relationships

Software Architecture Description - View

Types of view.

- **Process view** – shows processes or threads connected through communication, synchronization and exclusion operations

- **Concurrency view** – shows components and connectors, where connectors represent “threads of logic”

- **Shared Data (Repository) view** – shows the components and connectors that create, store, and access persistent data

- **Client-Server View** – shows collaborating clients and servers and the connectors between them (i.e. the protocols and messages they share)

Software Architecture Description - View

Types of view.

- **Deployment view** – shows software elements and their allocation to hardware and communications elements

- **Implementation view** – shows software elements in the development, integration and configuration control environment and their mapping to file structures

- **Work distribution view** – shows modules and how they are assigned to the development teams responsible for implementing and integrating them

Software Architecture Description - View

View-based architecture modeling specification.

- **The specification of a software application's architecture is called an architectural description.**

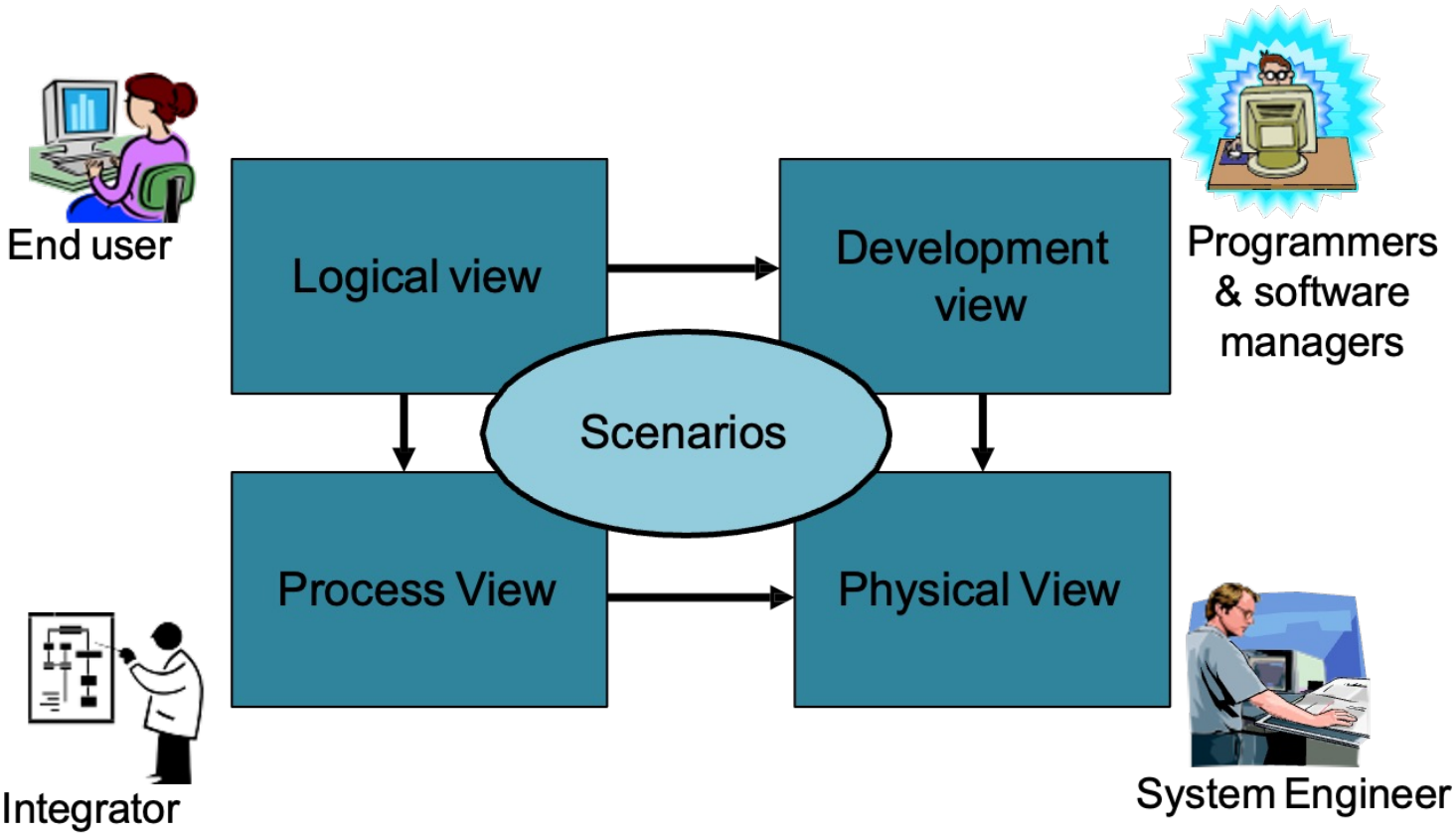
- **In IEEE 1471 terminology, a system has an architecture, which is described by an architecture specification.**

1. The architectural description selects one or more viewpoints.
2. Each viewpoint contains one or more related views
3. A view consists of one or more models and focuses on a single viewpoint.
4. An architecture description consists of one or more views.

Software Architecture Description - View

View-based architecture modeling specification - 4+1 view model.

Kruchten of Rational Company proposed the "4 + 1" model for architecture description in 1995. This model is based on the architecture definition proposed by Perry & Wolf and Berry Boehm.



Software Architecture Description - View

View-based architecture modeling specification - 4+1 view model.

- **Logical view:** supports behavioural requirements. The key abstraction is an object or object class.
- **Process view:** Addressing concurrency and distribution. Map threads to objects.
- **Development view:** organizes software modules, libraries, subsystems, and development units.
- **Physical view:** Maps other elements to processing and communication nodes.
- **Use case view:** illustrates the architecture by mapping other views to important use cases (these use cases are called scenarios), which constitute the fifth view.

Software Architecture Description - View

View-based architecture modeling specification - others.

- **Siemens 4 views:** Conceptual View Module; interconnection view, execution view; code view.

- **Herzum & Sims:** Technology Architecture; Application Architecture; Project Management Architecture; Functional Architecture

- **Views for Reducing Software Costs:** Module exploded view; Process view; Use views.

Software Architecture Description - View

How to choose views?

Many authors and methods specify a standard set of views, but each view has a certain cost and corresponding benefits. Therefore, we need a way to select a view.

- **Create Form.**

  1. Row: List the system architecture related to interests.

  2. Column: Lists a set of styles that can be applied to the documented schema.

  3. If the system architecture x of interest requires view y, check the (x, y) box.

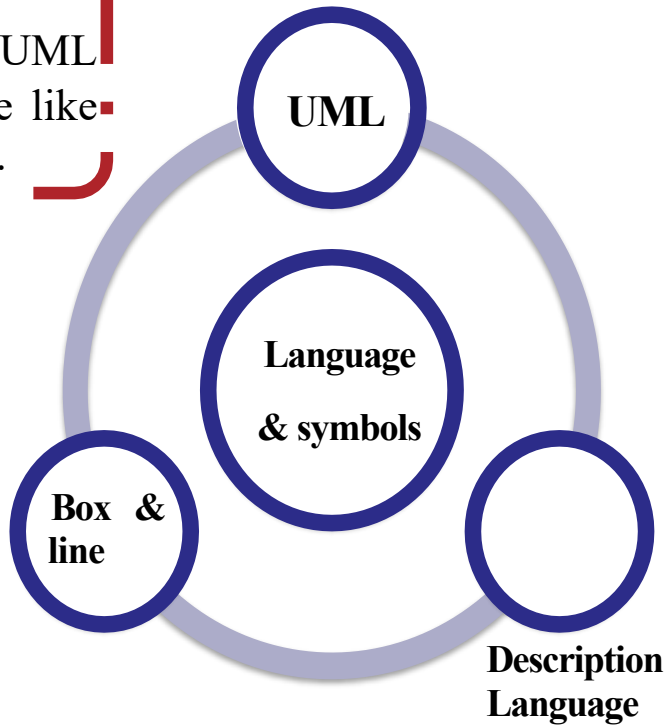
- **Combine views appropriately to reduce the number.**

- **Prioritize views based on needs (some stakeholders may have additional weight).**

Software Architecture Description - UML

UML compared to other methods, advantages?

1. It is not intended to document architectural information.
2. In fact the standard language UML 2.0 is better, it has architecture like "components" and "connectors".



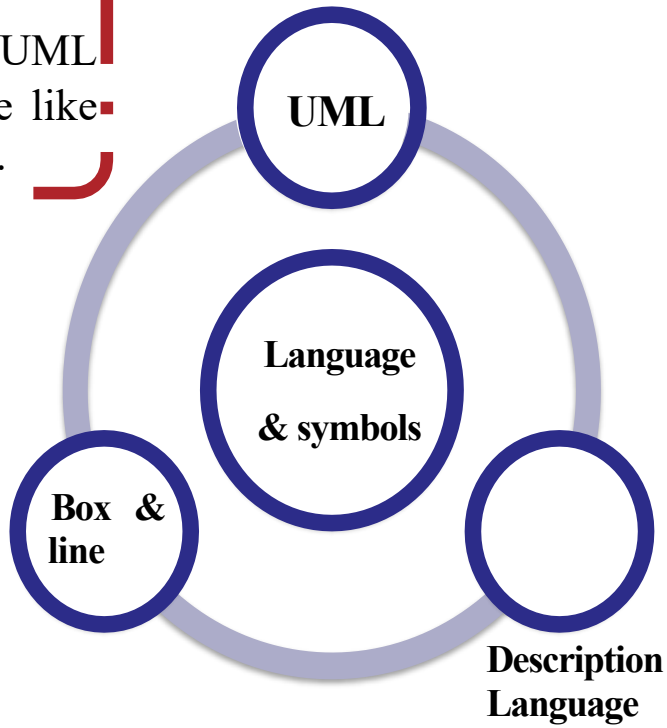
1. Subject of much research in the 1990s; not used much in practice
2. Architecture Description Language (ADL) recently became an IEEE standard.

1. Always use legends
2. Advantages: Flexibility.
3. Disadvantages: vague, no tool support

Software Architecture Description - UML

UML compared to other methods, advantages?

1. It is not intended to document architectural information.
2. In fact the standard language UML 2.0 is better, it has architecture like "components" and "connectors".



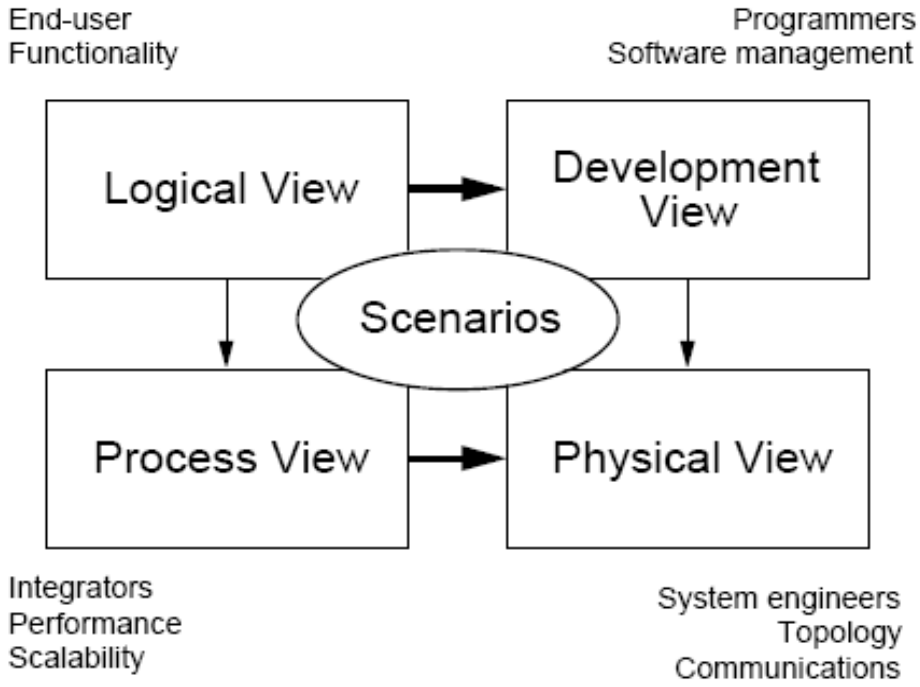
1. Always use legends
2. Advantages: Flexibility.
3. Disadvantages: vague, no tool support

1. Subject of much research in the 1990s; not used much in practice
2. Architecture Description Language (ADL) recently became an IEEE standard.

Software Architecture Description - UML

4 + 1 View Model

The "4+1" view model describes the software architecture from **five different perspectives** including **logical view**, **process view**, **physical view**, **development view** and **scenarios view**. Each view only cares about one aspect of the system, and only five views combined can reflect the entire content of the system's software architecture.



Software Architecture Description - UML

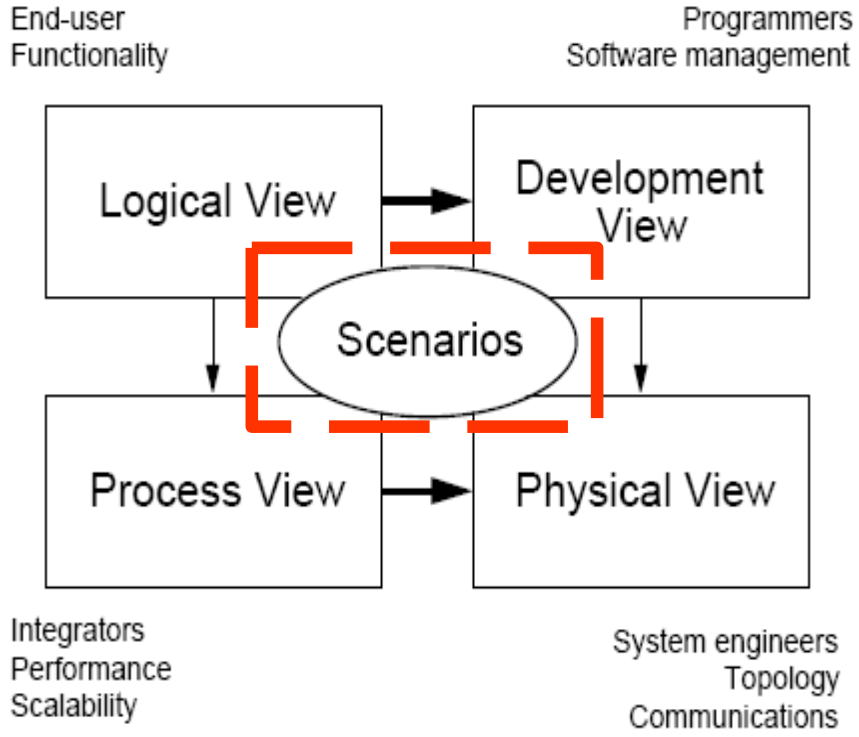
4 + 1 View Model

- **Use case view (scenario)**

1. Describe the functionality of the system being modelled from the perspective of the external world.

2. This view is needed to describe what the system should do. All other views rely on the use case view (scenario) to guide them, which is why the model is called 4+1.

3. This view usually contains use case diagrams, description and overview diagrams.



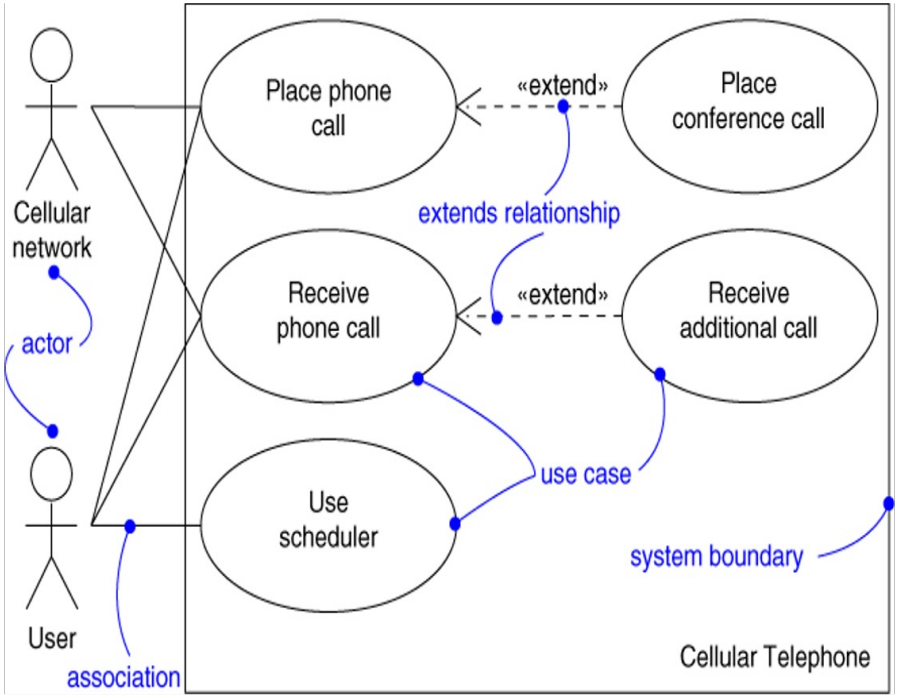
Software Architecture Description - UML

Use Case Diagram

Use cases are a valuable tool in helping to understand the functional requirements of a system and are descriptions of the functionality provided by the system.

1. Used to display several roles and the connection relationships between these roles and use cases provided by the system.

2. It is important to remember that use cases represent an external view of the system; therefore, do not expect any correlation between use cases and classes internal to the system.



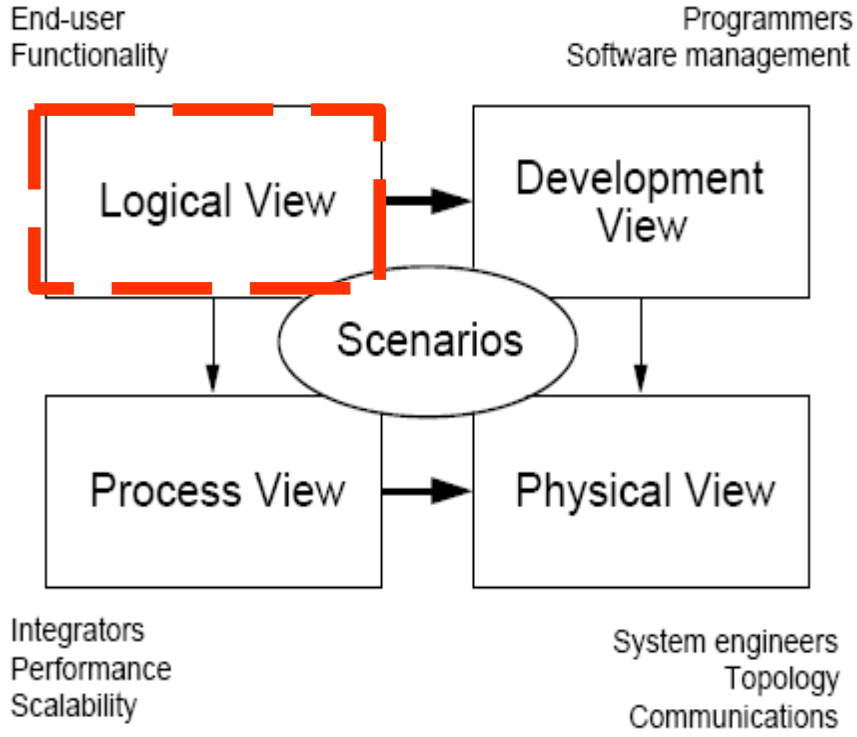
Software Architecture Description - UML

4 + 1 View Model

- **Logical view**

1. An abstract description that describes the parts of the system. Used to model the components of a system and how the components interact with each other.

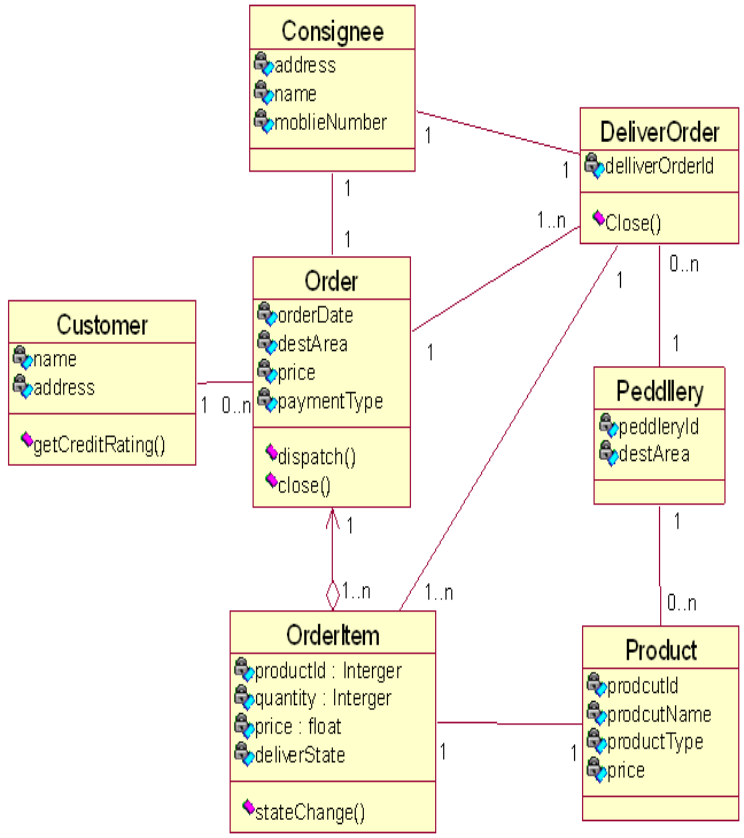
2. Usually include class diagrams, object diagrams, state diagrams and communication diagrams.



Software Architecture Description - UML

Class Diagram

Class diagram represents the classes in the system and the relationship between classes. It is a description of the static structure of the system.



The trouble with class diagrams is that they are extremely diverse, here are some tips for using them:

2. Concept class diagrams are very useful when exploring business languages.

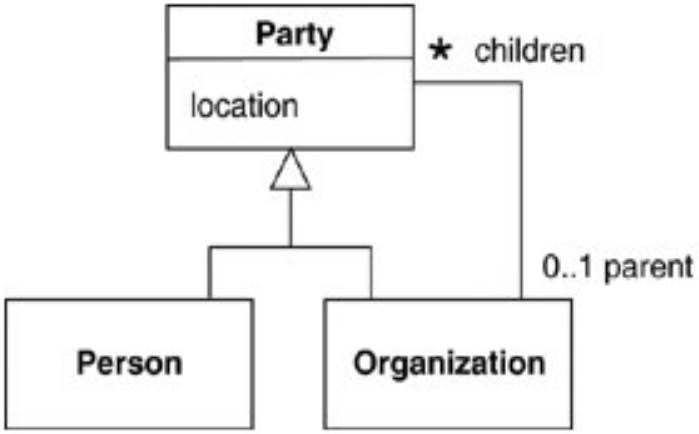
3. Don't draw models for everything; it's better to use a few up-to-date diagrams than many forgotten, outdated models.

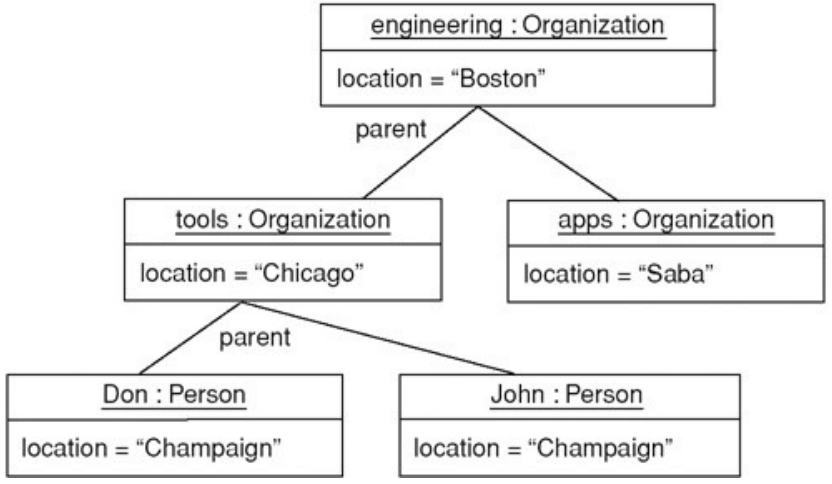
Software Architecture Description - UML

Object Diagram

An object diagram is a snapshot of the objects in the system at a certain point in time. Because it shows instances rather than classes, it is often called an instance diagram.

Object diagrams are useful for showing examples of objects that are connected together. In many cases, the structure can be precisely defined using a class diagram, but the structure is still difficult to understand. In these cases, a few object diagram examples can go a long way.





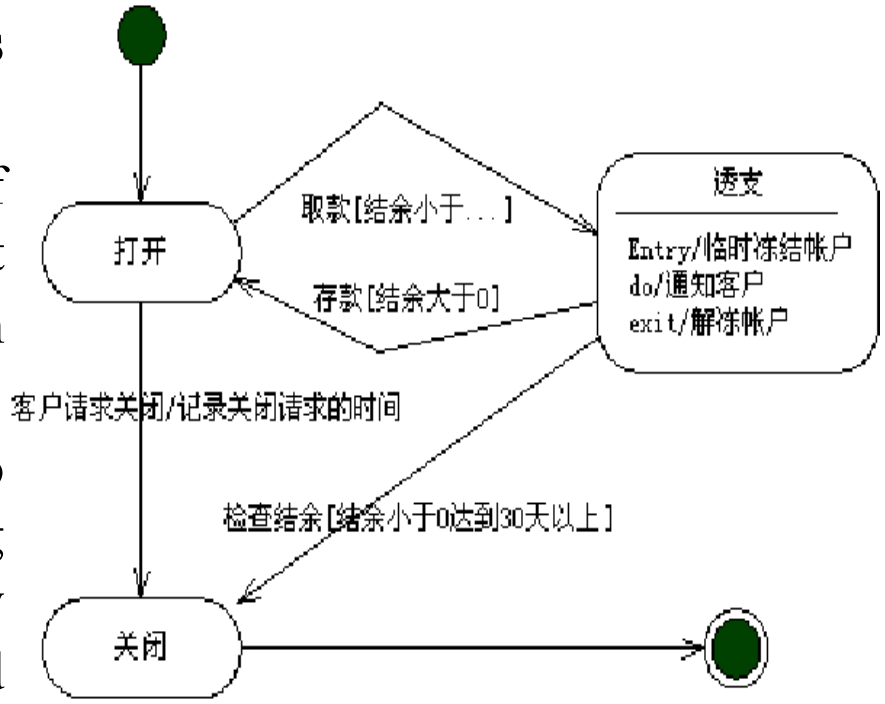
Software Architecture Description - UML

State Diagram

A state diagram describes all possible states of an object of a class and the transition conditions of the state when an event occurs. Typically, state diagrams complement class diagrams.

1. Good at describing the behavior of objects in multiple use cases, less good at describing behavior involving the cooperation of many objects.

2. When using state diagram, don't try to draw it for every object in the system. Using state diagrams only for objects that clearly have special behavior can help you understand what's going on. UI and control objects are by far the most commonly described objects using state diagram because they have many types of behavior.

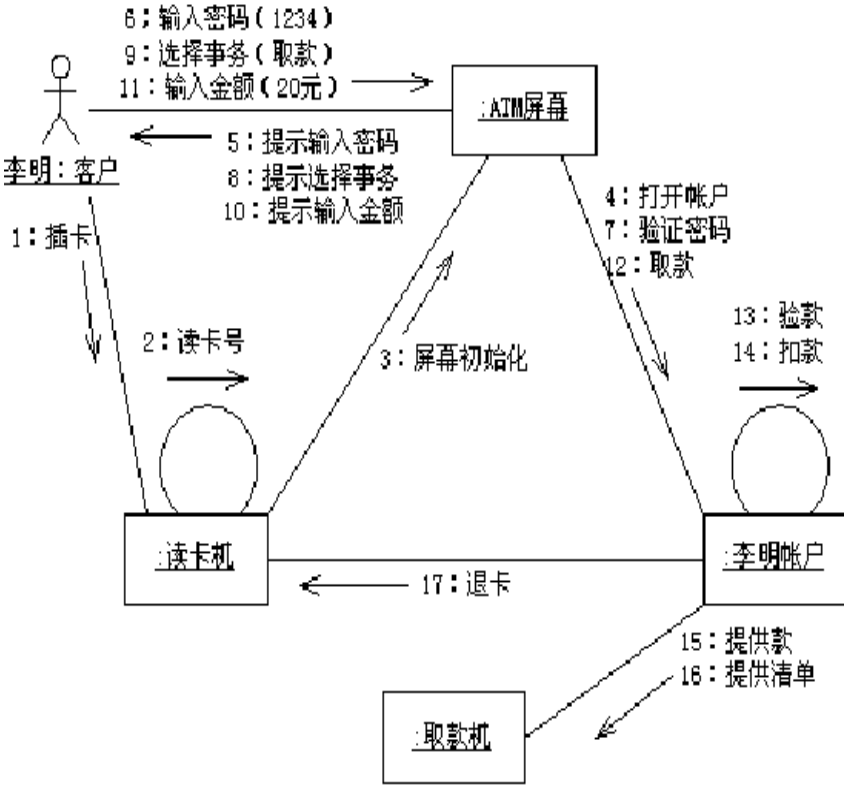


Software Architecture Description - UML

Communication Diagram

A communication diagram is an interaction diagram that emphasizes the organizational structure between objects that send and receive messages. A communication diagram shows a series of objects, the connections between objects, and the messages sent and received between objects.

1. Describing the collaboration relationship between objects, a collaboration diagram (communication diagram) is similar to a sequence diagram, showing the dynamic collaboration relationship between objects..



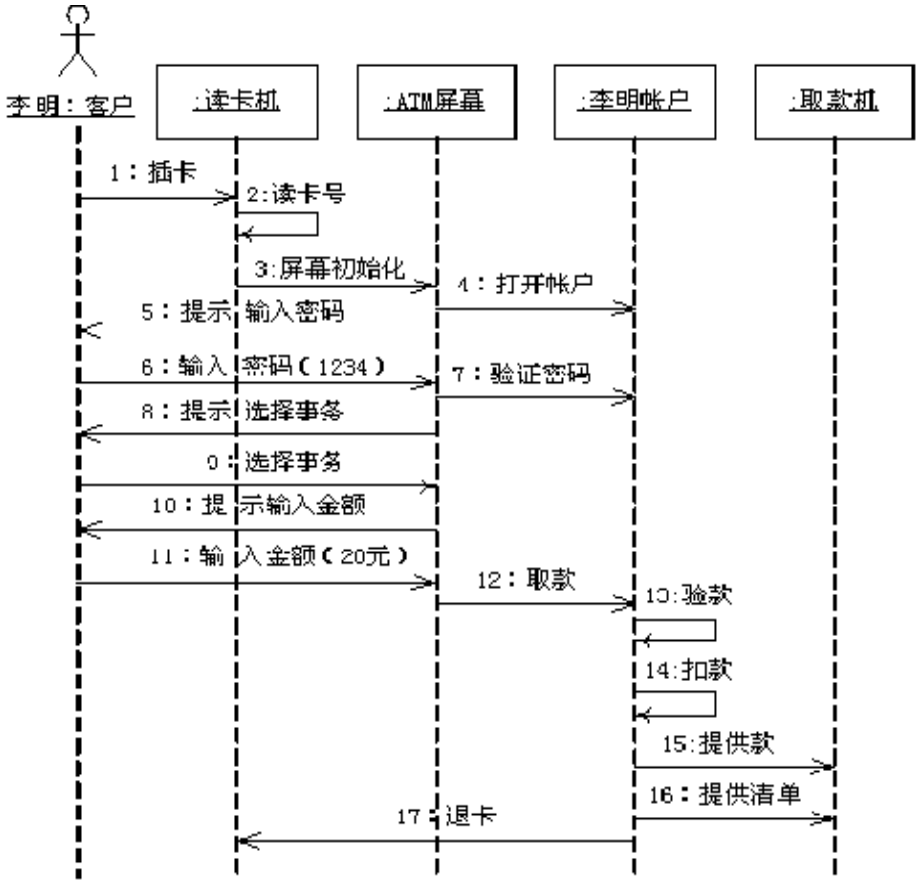
Software Architecture Description - UML

Sequence Diagram

In software engineering, a sequence diagram is a representation of object interaction. Mainly used to display these interactions between objects in a series of sequences in which they occur.

1. It is used to reflect the dynamic collaboration relationship between several objects, that is, how the objects interact with each other over time.

2. Used to view the behavior of multiple objects in a single use case. It is more about describing the cooperation between objects, and is not good at precise definition of their behavior.

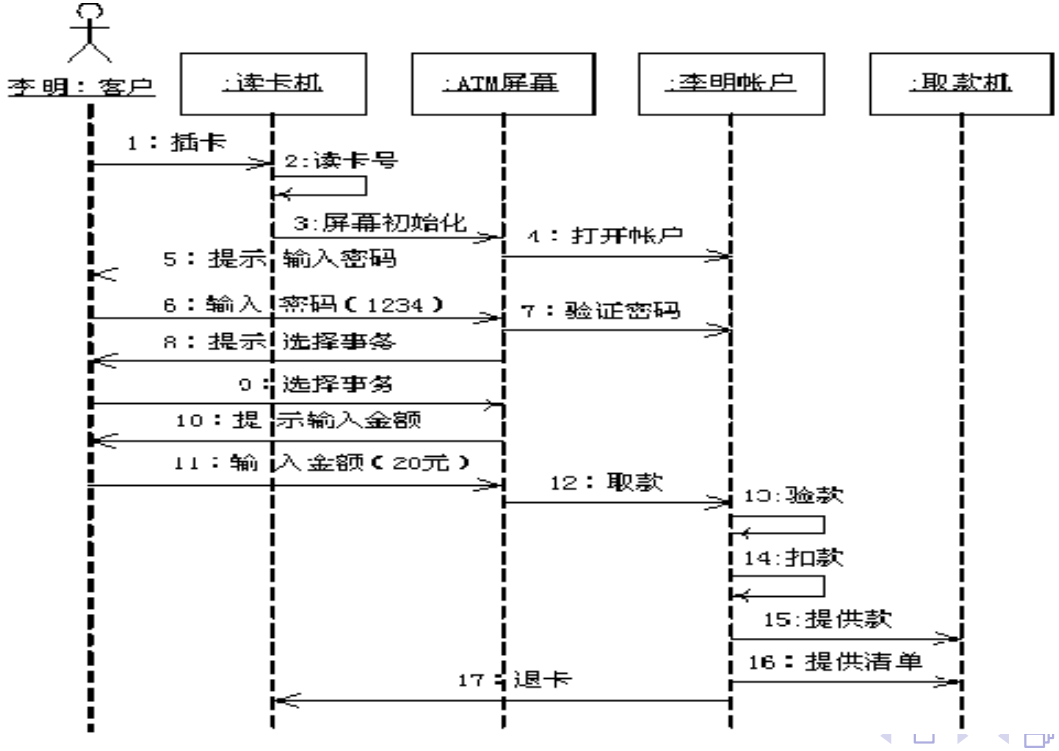


Software Architecture Description - UML

Sequence Diagram

In software engineering, a sequence diagram is a representation of object interaction. Mainly used to display these interactions between objects in a series of sequences in which they occur.

3. State diagram are generally used when viewing the behaviour of a single object across multiple use cases. Generally consider using activity diagrams when viewing behaviour across multiple use cases or multiple threads.

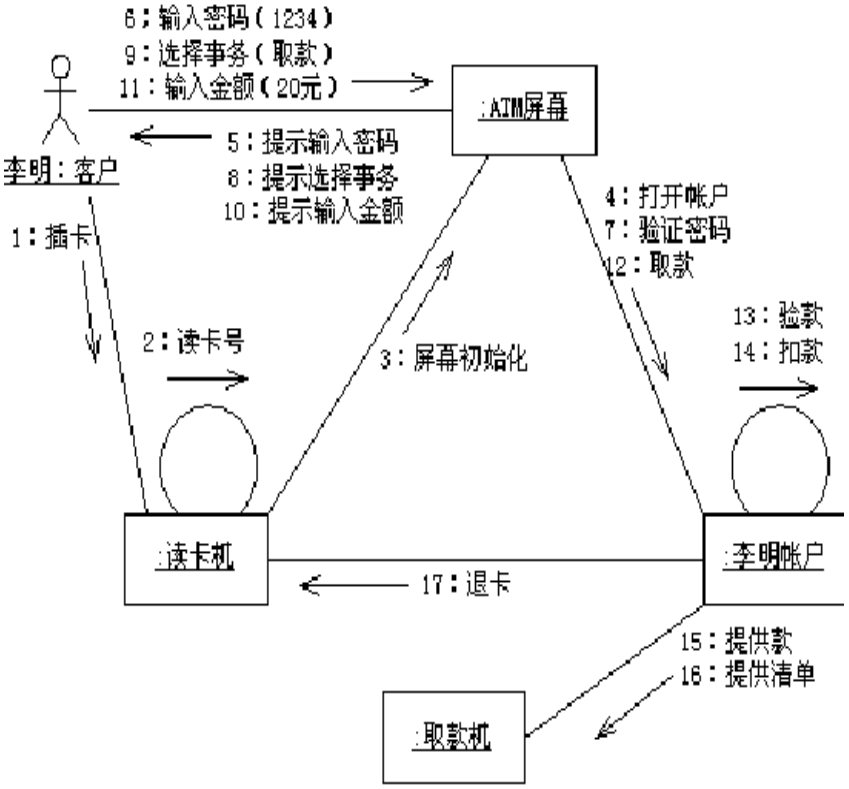


Software Architecture Description - UML

Communication Diagram

1. The main question is when to use communication diagrams instead of the more common sequence diagrams. This largely comes down to personal preference: some people prefer one to the other.

2. When you want to emphasize the call sequence, it is better to choose the sequence diagram; when you want to emphasize the link, it is better to choose the communication diagram. Many people find that communication diagrams are easier to change on a whiteboard and are therefore a good way to explore alternatives.

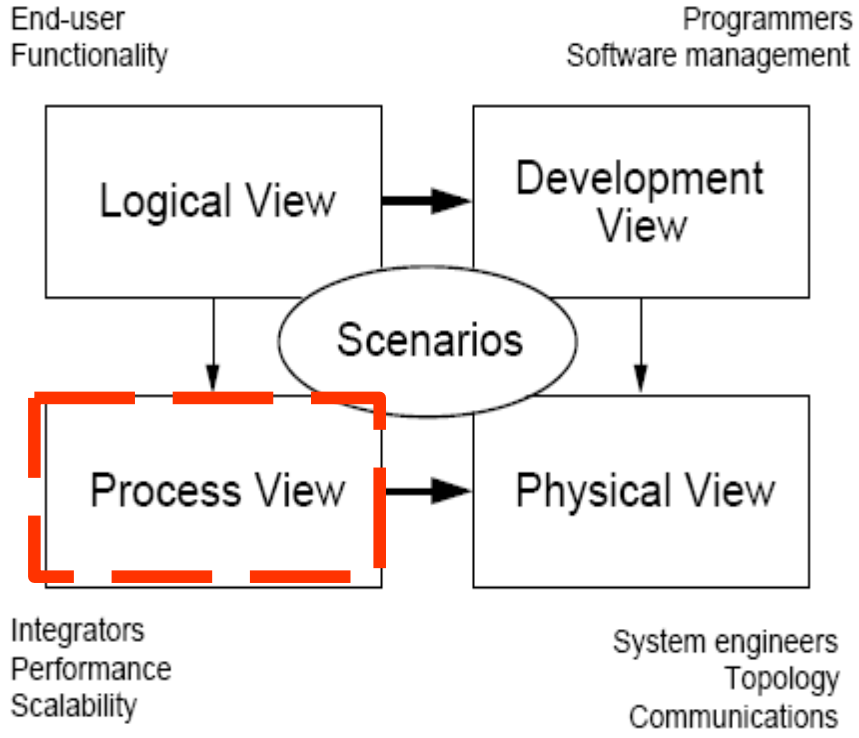


Software Architecture Description - UML

4 + 1 View Model

- **Process view**

1. Describe the processes in the system. This view is particularly useful when visualizing what must be happening in the system.
2. This view usually contains an activity diagram.



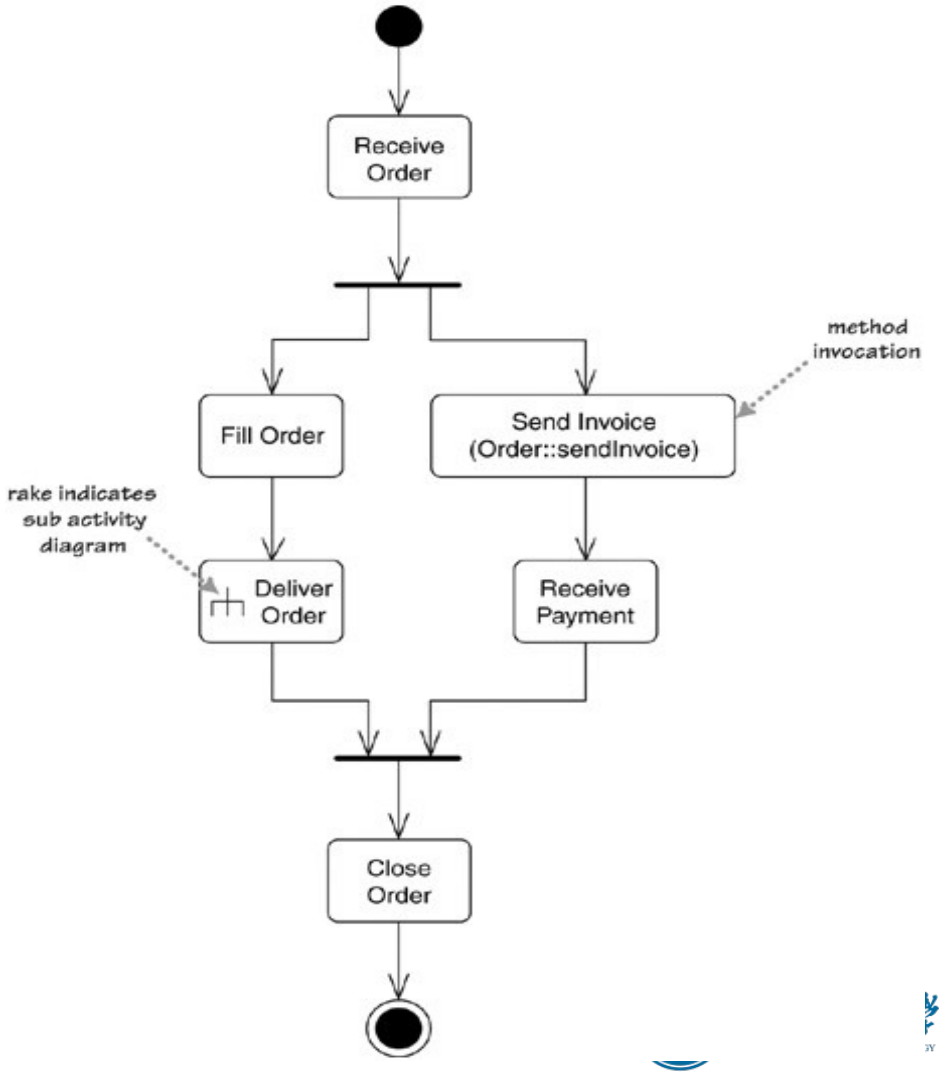
Software Architecture Description - UML

Activity Diagram

Describing the activities to be performed to meet the use case requirements and the constraint relationships between activities is helpful to identify parallel activities.

1. The huge advantage is that parallel behaviour is supported and encouraged. This makes it a great tool for workflow and process modelling, and much of the push in UML 2.0 actually came from people involved in the workflow.

2. In principle, parallel algorithms for concurrent programs can be described using the advantages of forks and joins.



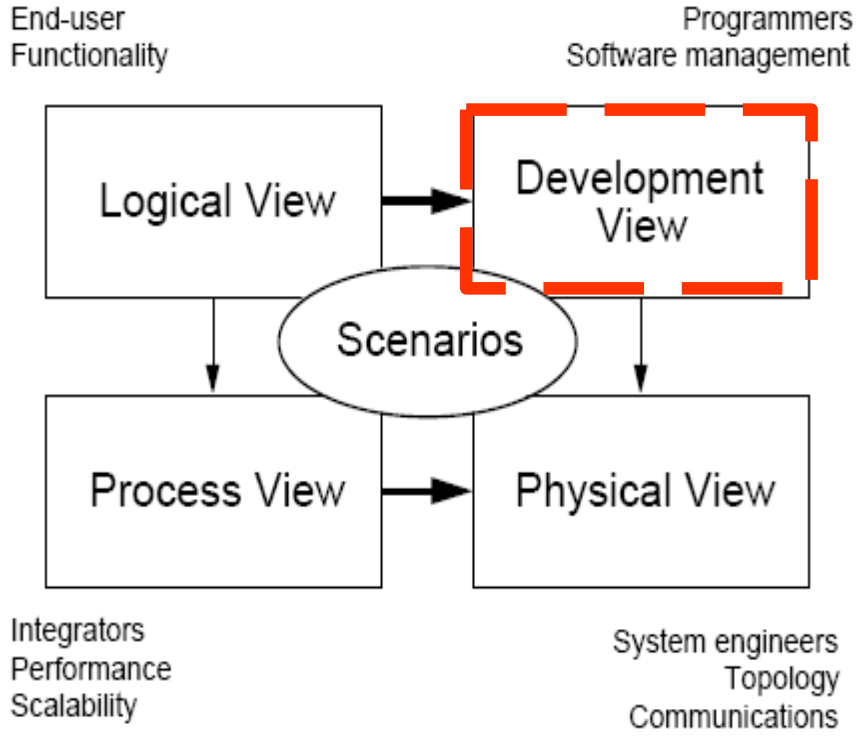
Software Architecture Description - UML

4 + 1 View Model

- **Development view**

1. Describe how parts of the system are organized into modules and components. It is very useful to manage the layers in the system architecture.

2. This view typically contains package diagrams and component diagrams.



Software Architecture Description - UML

Package Diagram

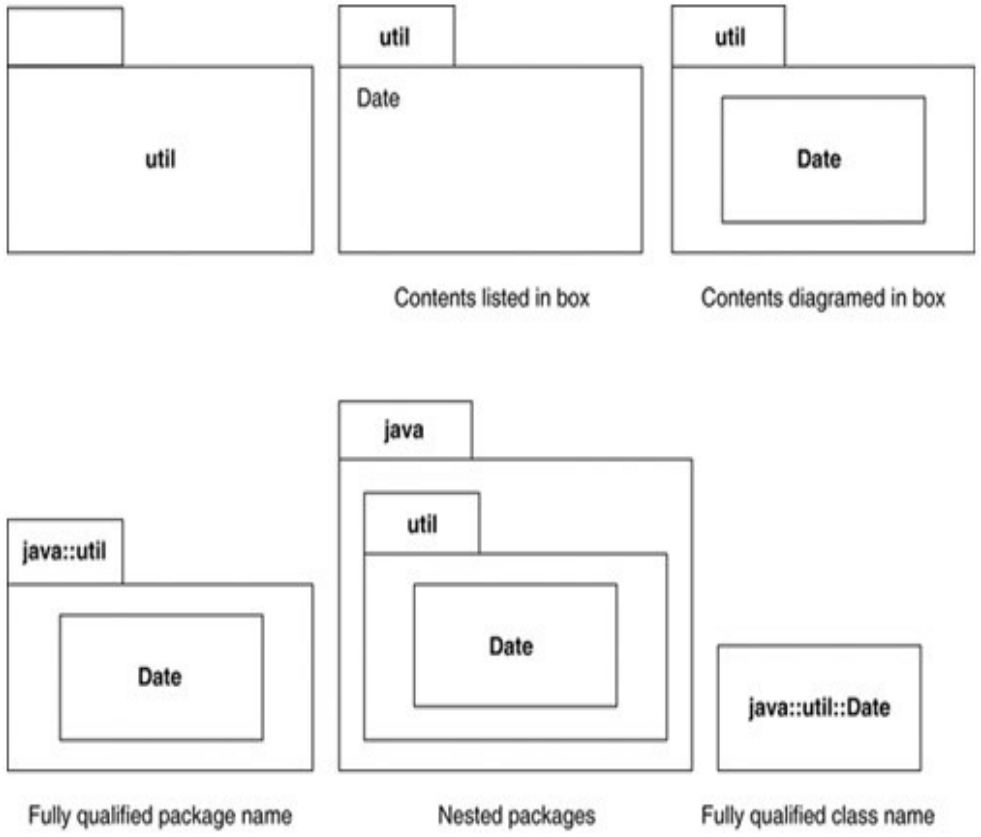
A package is a combination of model elements represented in UML with a folder-like notation, allowing to take any structure from UML and group its elements into higher-level units.

1. Packages can be used in any part of UML.

2. Each element in the system can only be owned by one package, and one package can be nested within another package.

3. Use package diagrams to group related elements into a system

4. A package can contain satellite packages, diagrams, or individual elements



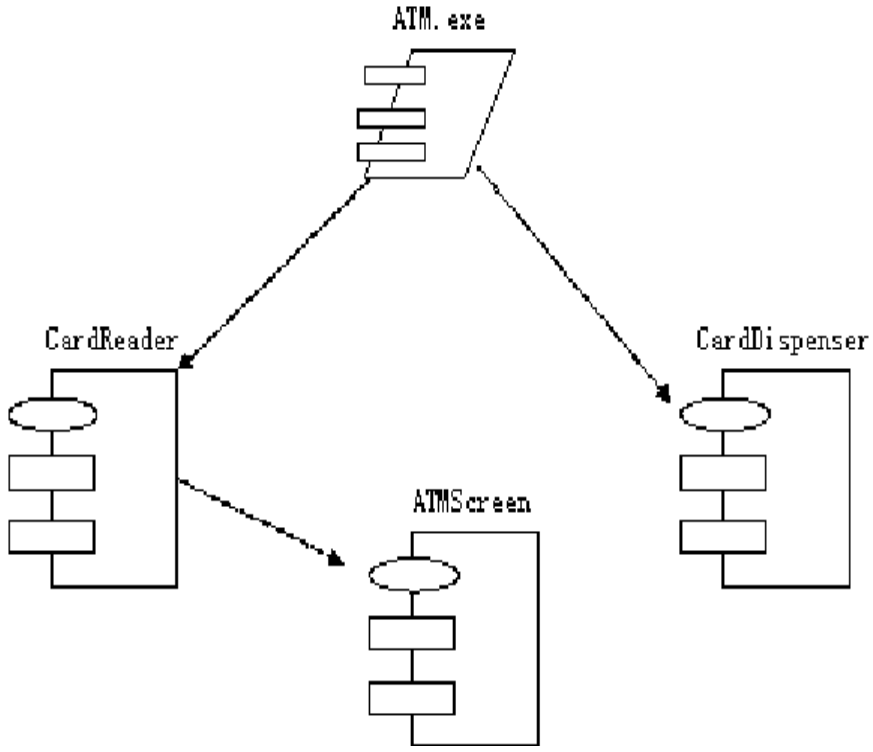
Software Architecture Description - UML

Component Diagram

A component diagram describes the physical structure of code components and the dependencies between components. Divide a system into components and hopefully show their interrelationships through interfaces or subdivision of components into lower-level structures.

1. The main purpose is to show the structural relationships between system components.

2. Useful communication tool for different groups.

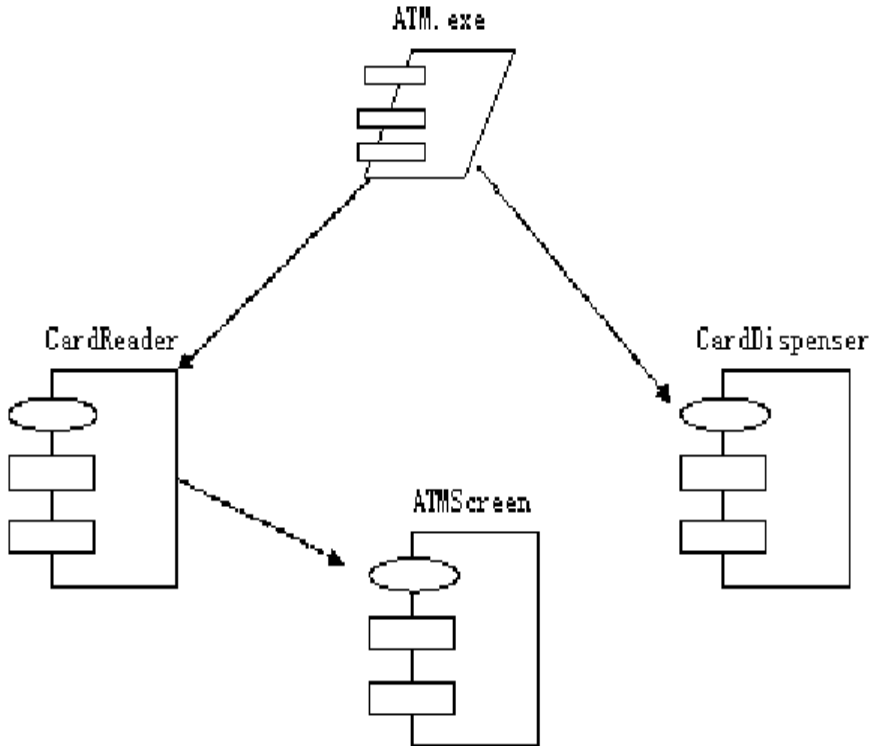


Software Architecture Description - UML

Component Diagram

A component diagram describes the physical structure of code components and the dependencies between components. Divide a system into components and hopefully show their interrelationships through interfaces or subdivision of components into lower-level structures.

3. In component-based development (CBD), component diagrams provide architects with a natural form to start modelling a solution. and allows an architect to verify that the required functionality of the system is implemented by the components.

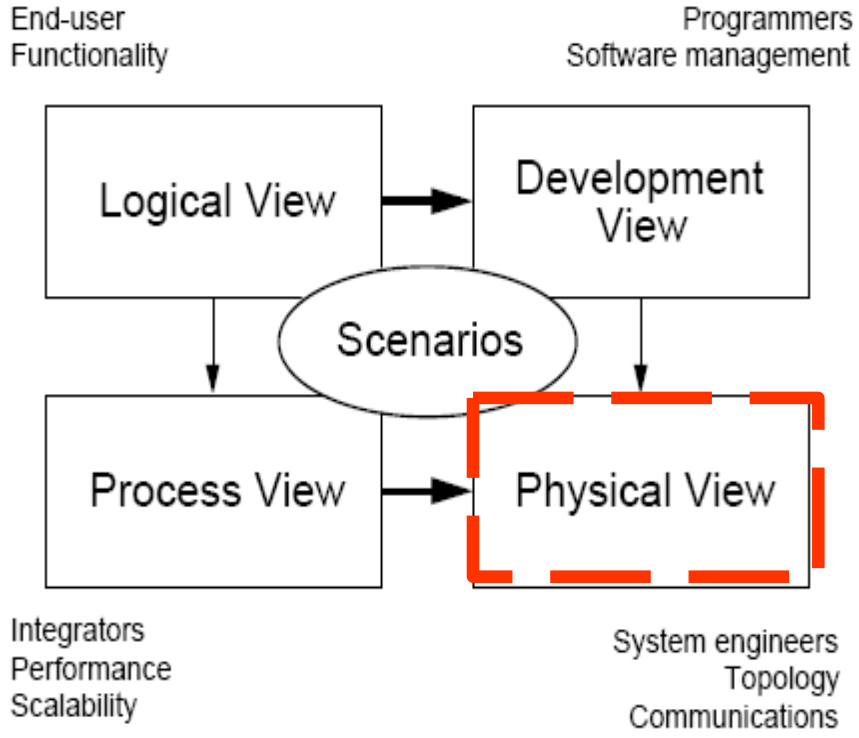


Software Architecture Description - UML

4 + 1 View Model

- **Physical view**

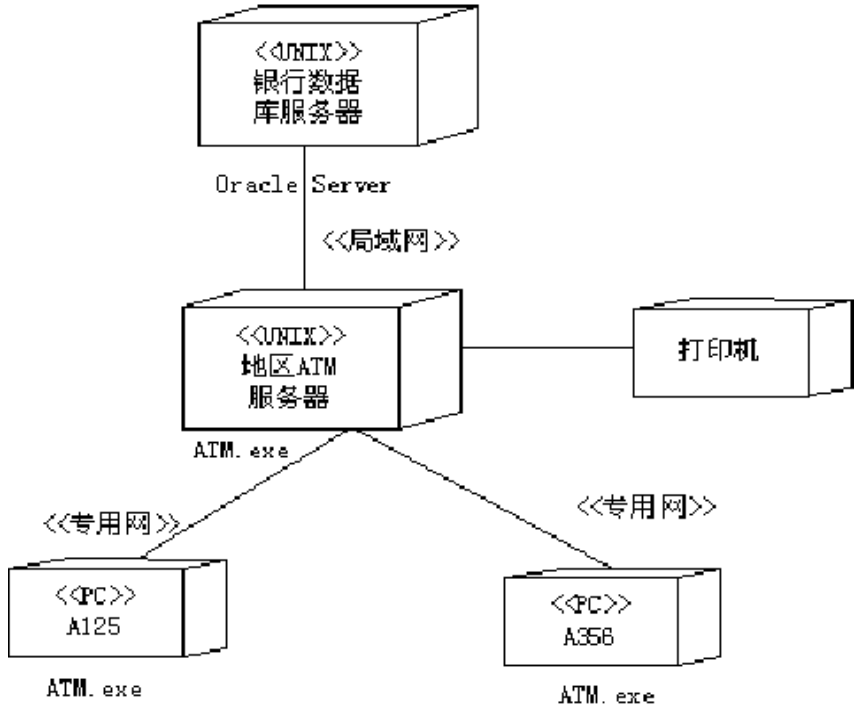
1. Describe how the system design described in the first three views is implemented as a set of real-world entities. The diagram in this view shows how the abstract parts map to the final deployed system.
2. This view typically contains a deployment diagram.



Software Architecture Description - UML

Deployment Diagram

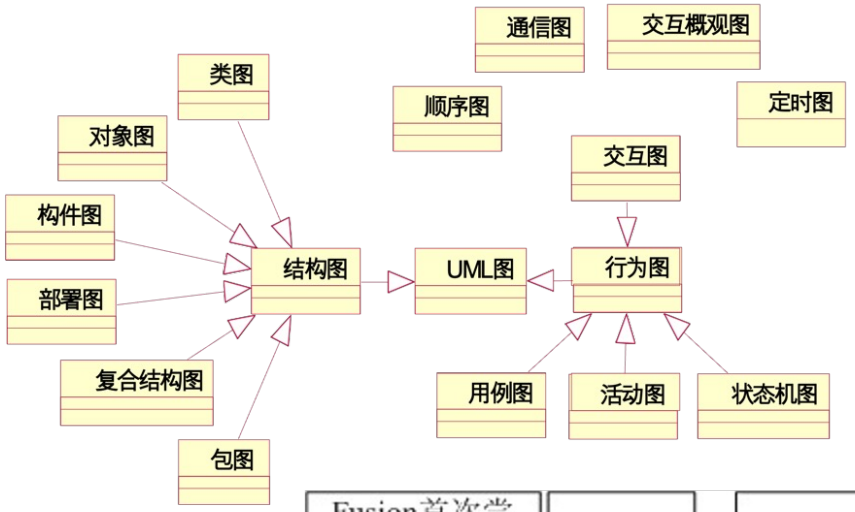
A deployment diagram defines the physical architecture of the software and hardware in the system. Describes a runtime hardware node and a static view of the software components running on these nodes. Shows the system hardware, the software installed on the hardware, and the middleware used to connect heterogeneous machines.

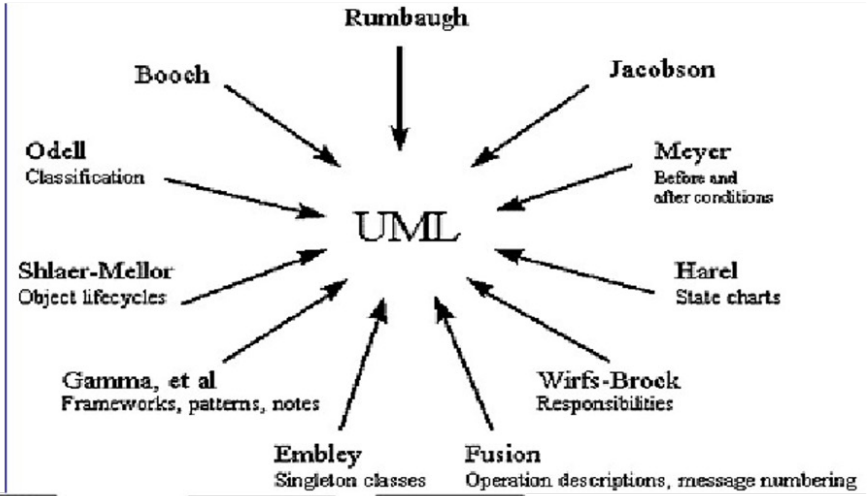


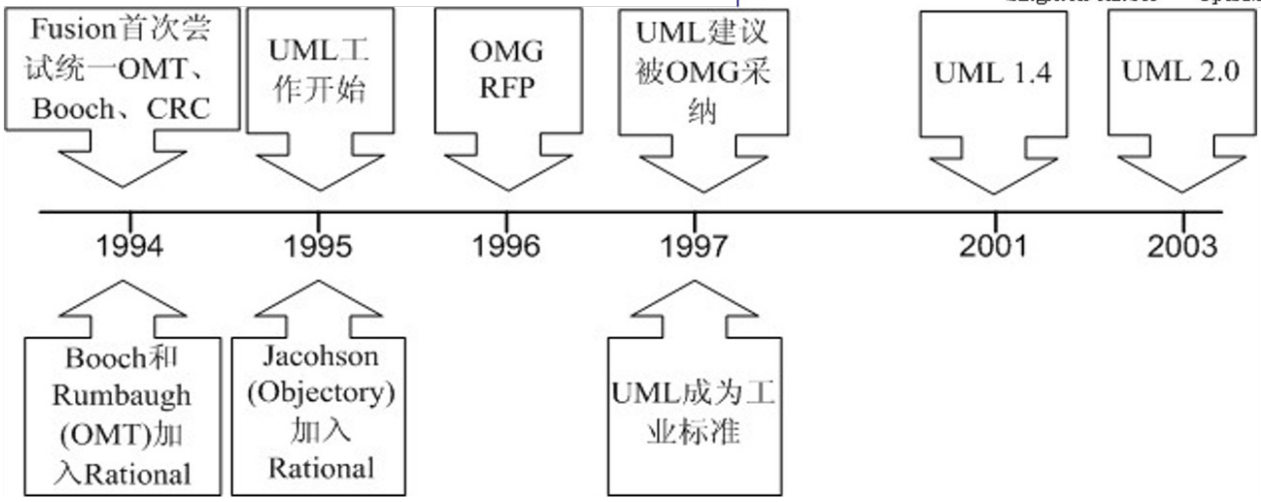
1. From the deployment diagram, you can understand the physical relationships between software and hardware components and the distribution of components across processing nodes.

2. Use a deployment diagram to show the structure of a runtime system while also conveying how the hardware and software elements that make up the application are configured and deployed.

Software Architecture Description - UML





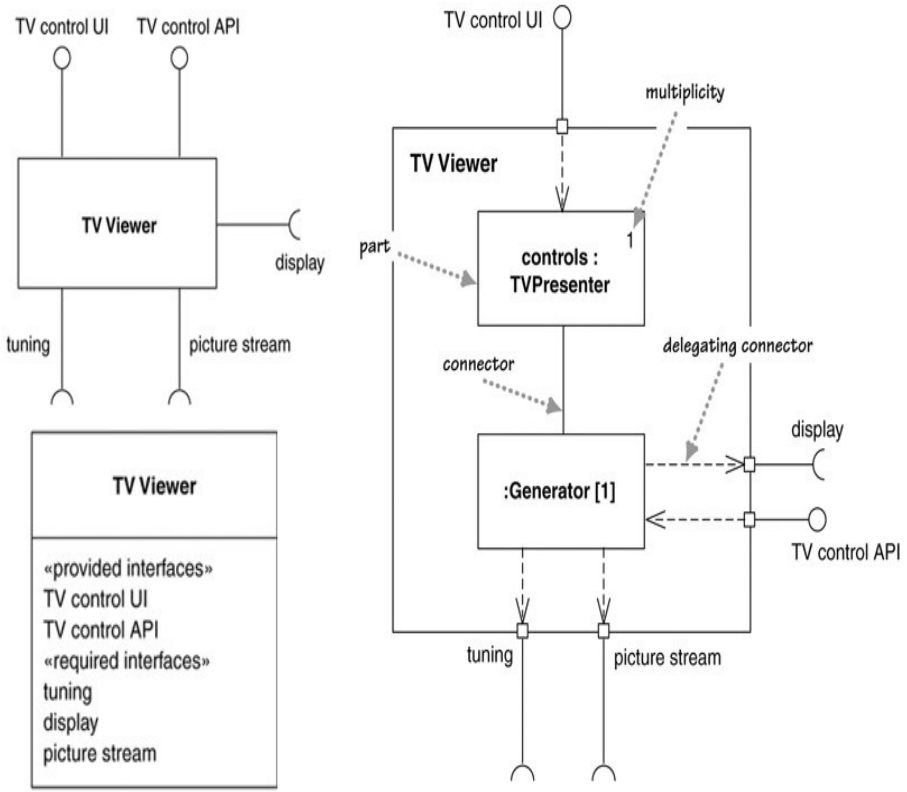


Software Architecture Description - UML

Composite Diagram

Composite structures are a new feature of UML 2. i.e., the ability to hierarchically decompose classes into internal structures. This can break a complex object into parts.

1. A good way to think about the difference between a package and a composite structure is that a package is a compile-time grouping, while a composite structure exhibits a run-time grouping.
2. It is a natural fit for showing components and how they are broken down into parts; therefore, many notations are used in component diagrams.

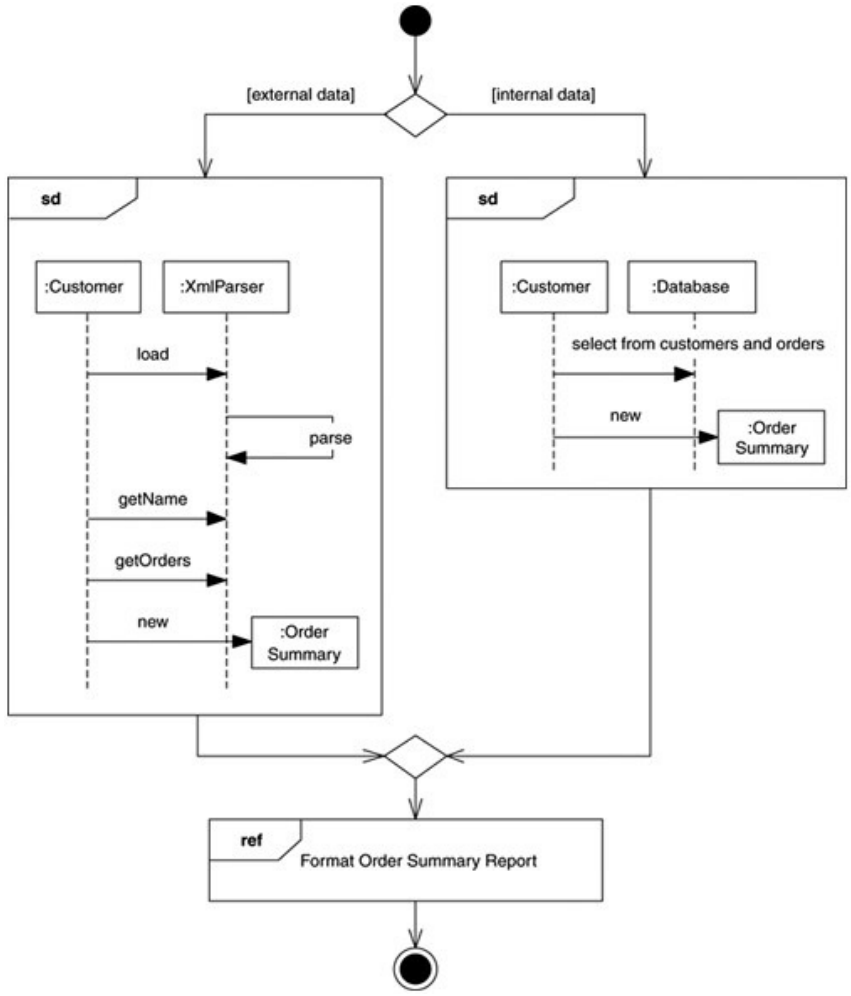


Software Architecture Description - UML

Interaction Diagram

Interaction diagram is a combination of activity diagram and sequence diagram. You can think of it as an activity diagram, where the activities are replaced with small sequence diagrams, or as a sequence diagram broken down with activity diagram notation to show the flow of control.

1. Interaction overview diagrams are more suitable when drawing sketches. First, model the business process through activity diagrams, then refine some key and low-complexity activity nodes, and use sequence diagrams to represent the flow between its objects.

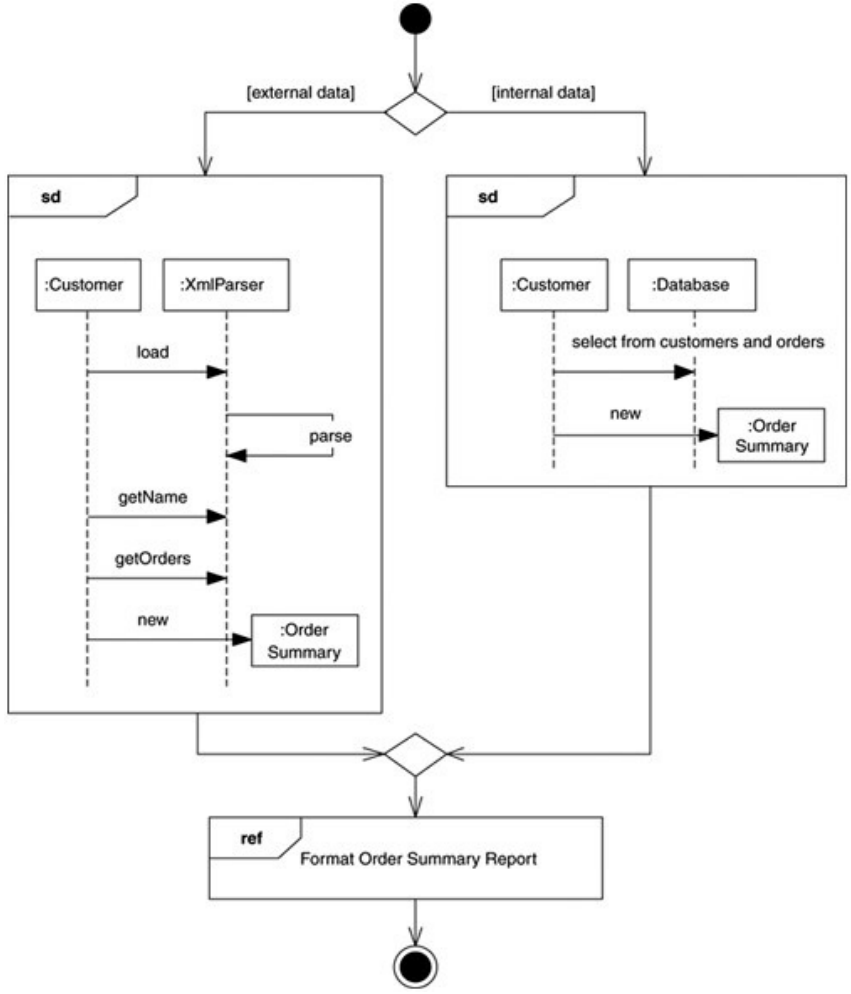


Software Architecture Description - UML

Interaction Diagram

Interaction diagram is a combination of activity diagram and sequence diagram. You can think of it as an activity diagram, where the activities are replaced with small sequence diagrams, or as a sequence diagram broken down with activity diagram notation to show the flow of control.

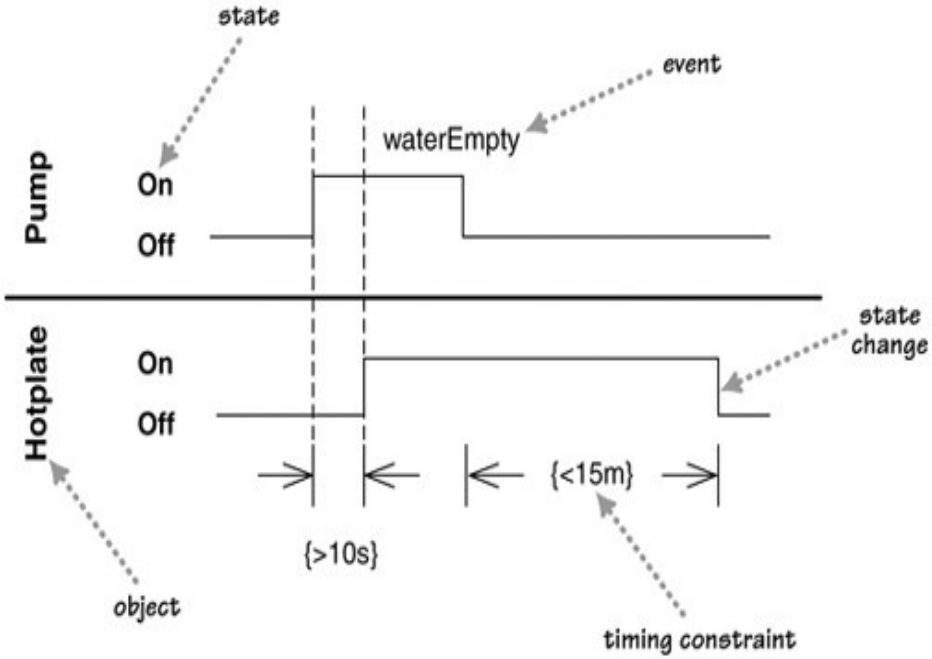
2. Don't blindly use interactive overview diagrams. It is not a good choice for larger scenes and will greatly reduce the readability of the model.

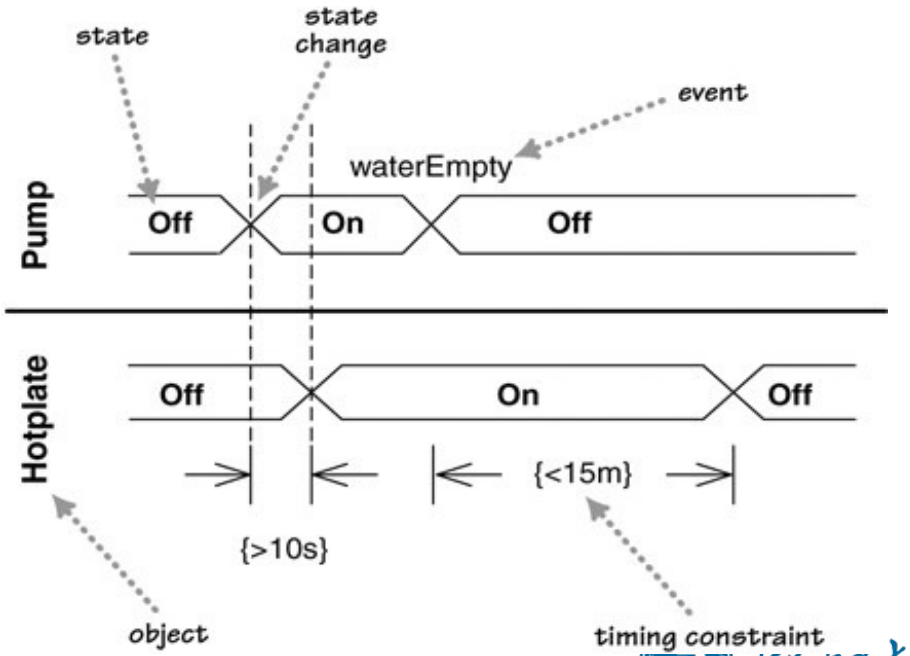


Software Architecture Description - UML

Timing Diagram

Sequence diagrams are another form of interaction diagrams where the focus is on timing constraints. Generally described for a single object or multiple objects. Useful for showing timing constraints between state changes of different objects.





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

Object-oriented modeling languages

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 sys- tems.

Overview of the UML

Definition

The UML is a language for ***visualizing, specifying, constructing, doc- umenting*** 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, component, node.

Behavioral elements

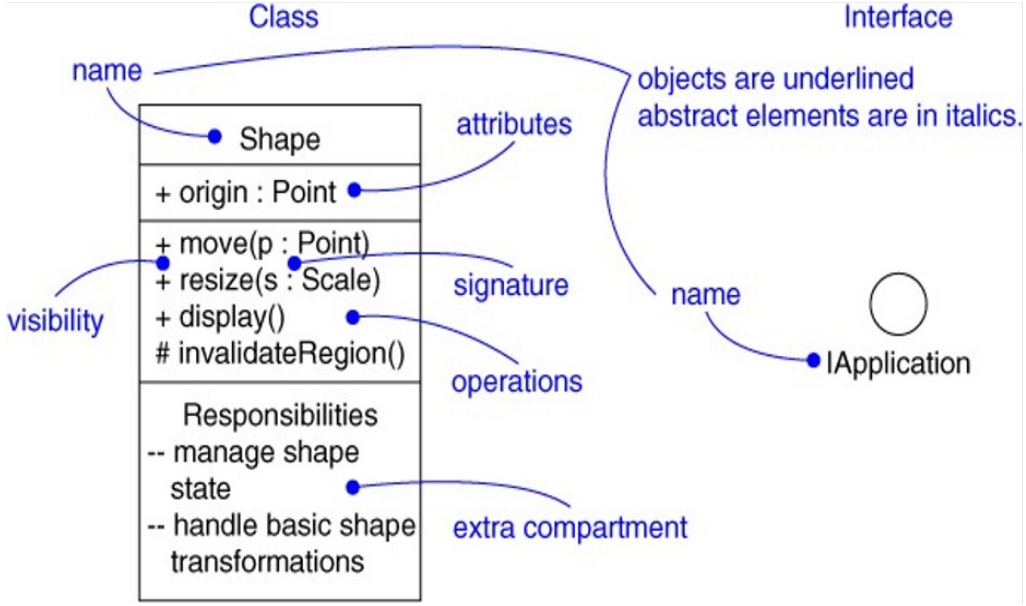
- Interaction
- State machine.

Grouping elements

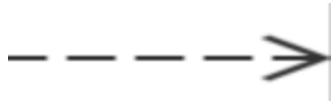
- Package, subsystem.

Other elements

- Note.



Relationships

Dependency(依赖) 

Association(关联) 

Aggregation(聚合) 

Composition(组合) 

Generalization(泛化) 

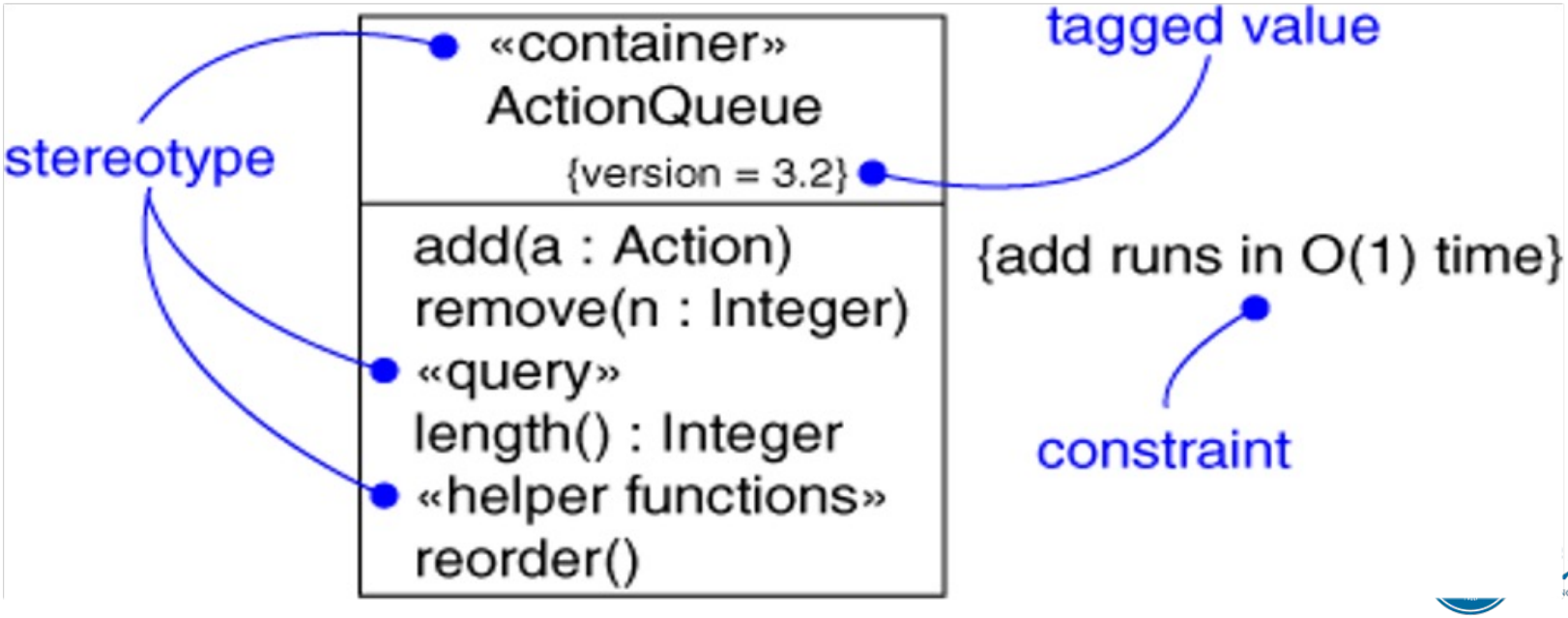
Realization(实现) 

Extensibility Mechanisms

Stereotype （构造性）

Tagged value （标记值）

Constraint （约束）



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.

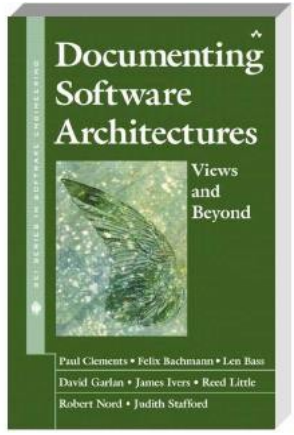
- Static views: use case, class, object, component, deployment, package, composite structure
- Dynamic views: sequence, communication, state machine, activity, timing, interaction overview

Thank you!

Documenting Software Architectures: Views and Beyond

Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford

Published by Addison-Wesley in September 2002 as part of the SEI Series in Software Engineering



北京工業大學 BEIJING UNIVERSITY OF TECHNOLOGY