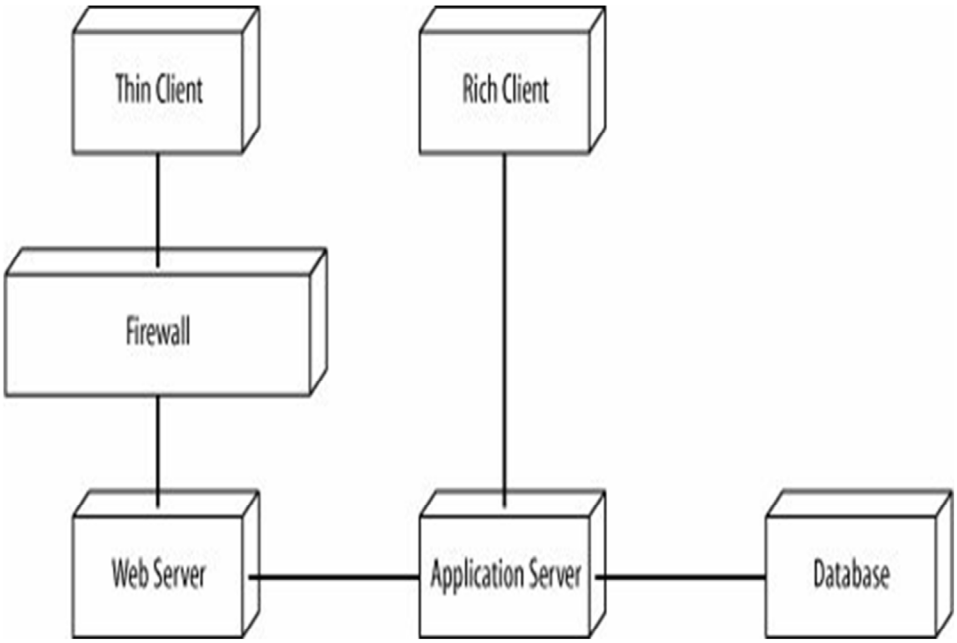
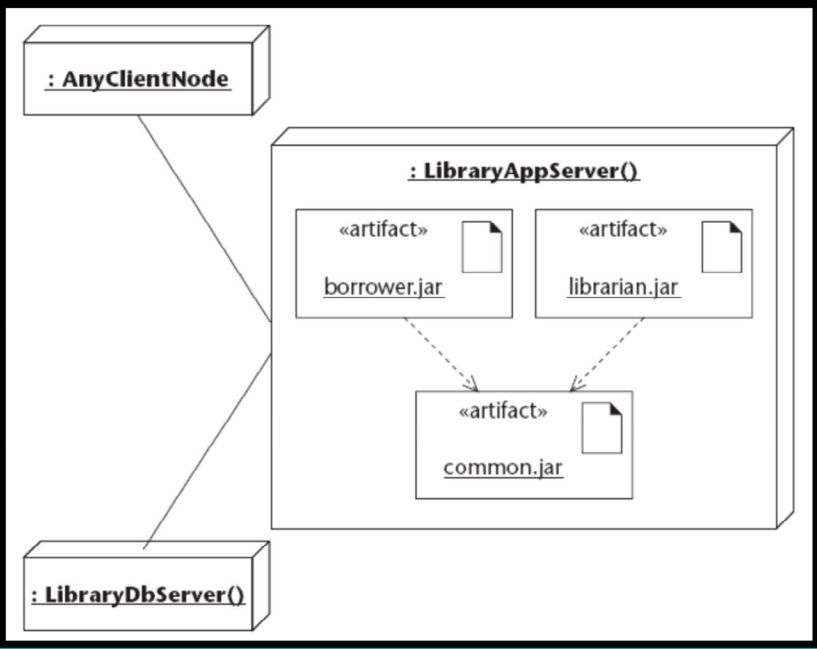
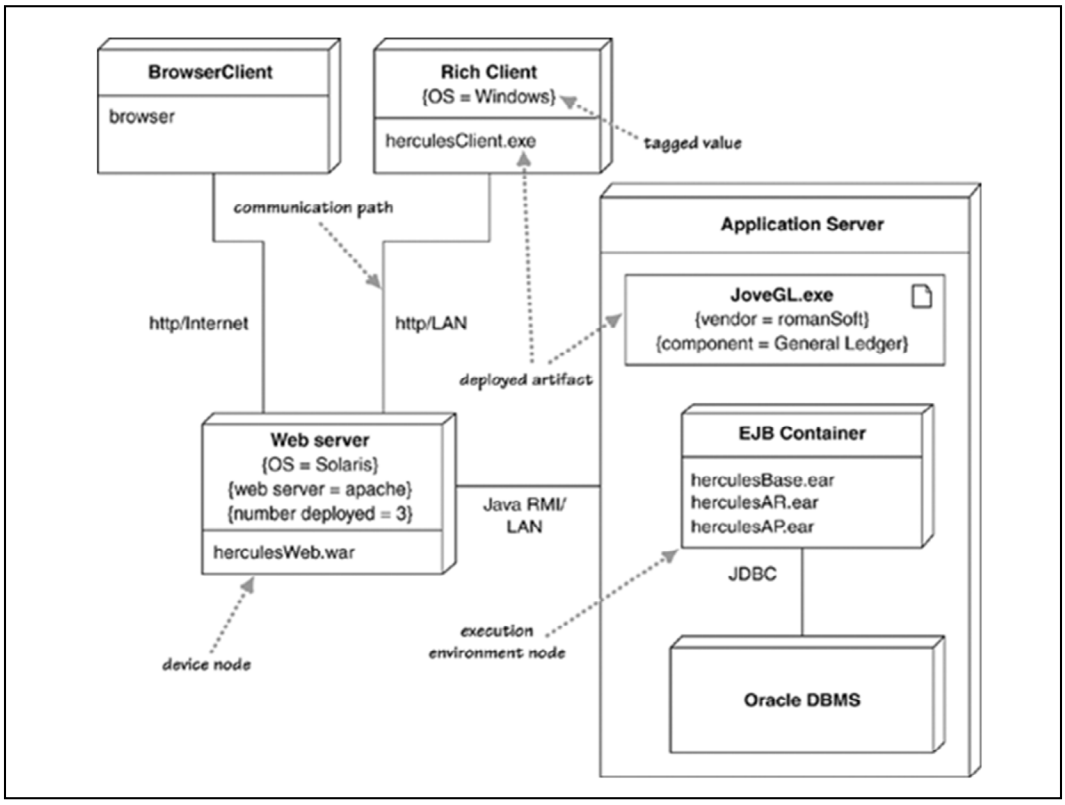
UML

Deployment Diagram


Thin Client
Rich Client
Firewall
Web Server
Application Server
Database

Deployment Diagram

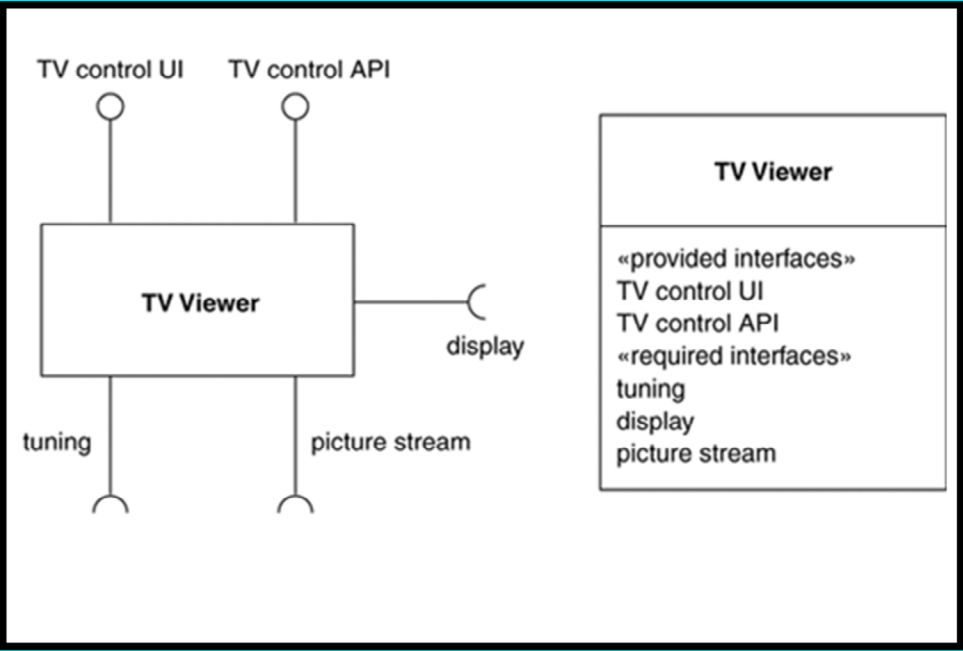



BrowserClient
browser
Rich Client
{OS = Windows}
herculesClient.exe
tagged value
communication path
Application Server
JoveGL.exe
{vendor = romanSoft}
{component = General Ledger}
http/Internet
http/LAN
deployed artifact
Web server
{OS = Solaris}
{web server = apache}
{number deployed = 3}
herculesWeb.war
Java RMI/
LAN
EJB Container
herculesBase.ear
herculesAR.ear
herculesAP.ear
JDBC
execution
environment node
device node
Oracle DBMS

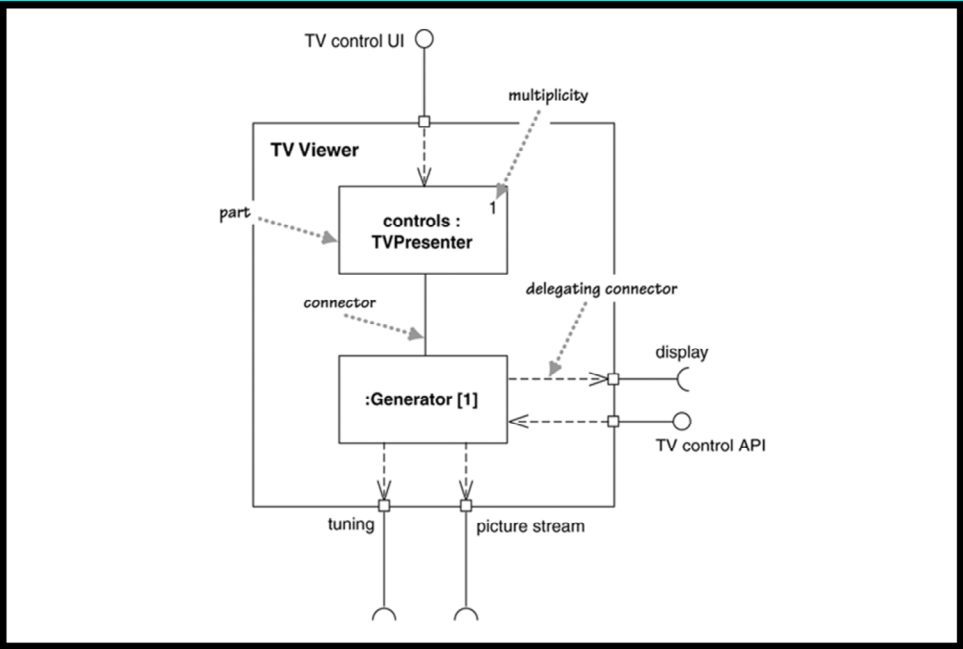
UML

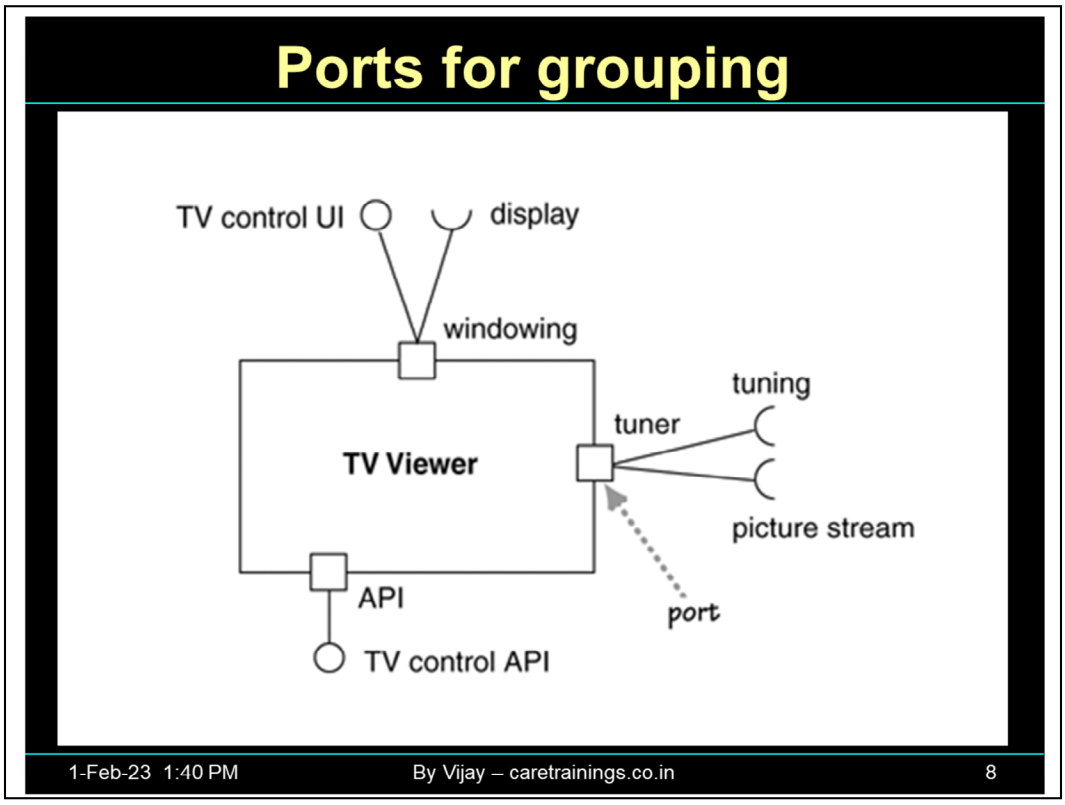
Composite Structures

Details of an Aggregate object



Internal view of Aggregate



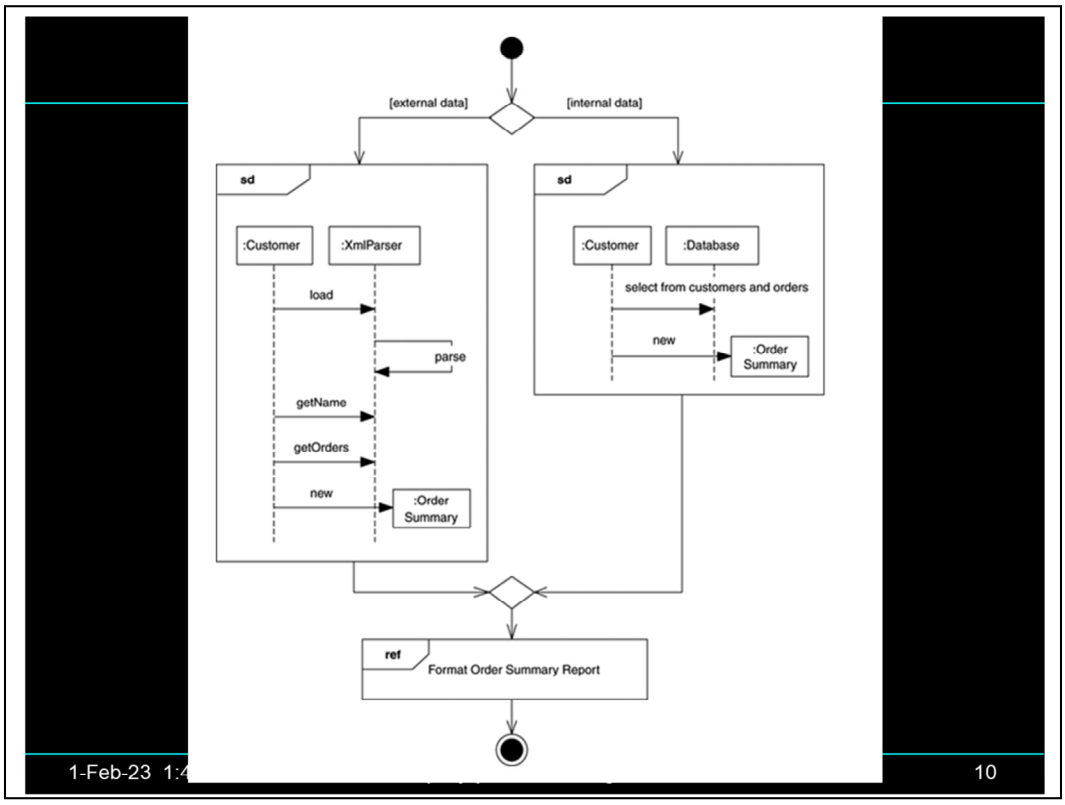


A good way of thinking about the difference between packages and composite structures is that packages are a compile-time grouping, while composite structures show runtime groupings. As such, they are a natural fit for showing components and how they are broken into parts; hence, much of this notation is used in component diagrams.

Port
A port is a typed element that represents an externally visible part of a containing classifier instance. Ports define the interaction between a classifier and its environment. A port can appear on the boundary of a contained part, a class or a composite structure. A port may specify the services a classifier provides as well as the services that it requires of its environment.

UML

Interaction Overview Diagram



Interaction overview diagrams are a grafting together of activity diagrams and sequence diagrams. You can think of interaction overview diagrams either as activity diagrams in which the activities are replaced by little sequence diagrams, or as a sequence diagram broken up with activity diagram notation used to show control flow. Either way, they make a bit of an odd mixture.

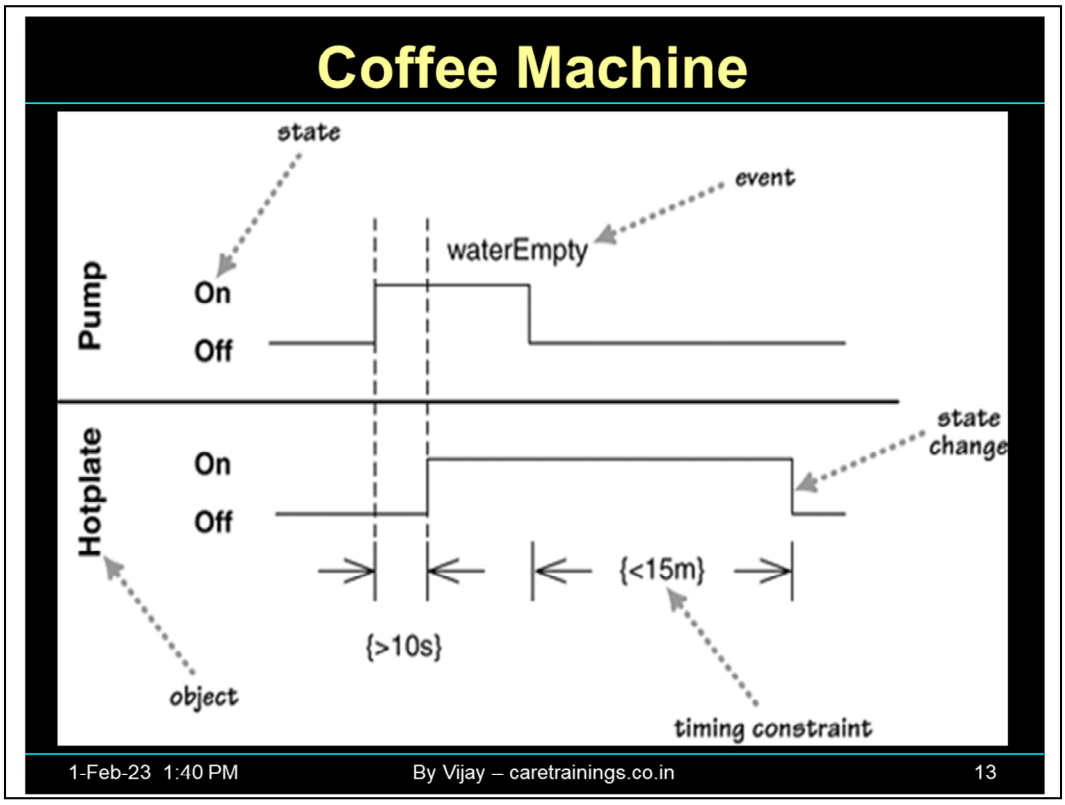
In this diagram, we want to produce and format an order summary report. If the customer is external, we get the information from XML; if internal, we get it from a database. Small sequence diagrams show the two alternatives. Once we get the data, we format the report; in this case, we don't show the sequence diagram but simply reference it with a reference interaction frame.

Interaction Overview Diagrams

- Interaction overview diagram is a type of activity diagram that shows control flow between various interactions.
- Instead of actions, these diagrams consist of:
  - Frames (instead of rounded rectangle nodes). A frame can contain an:
    - Interaction (any of the other interaction diagrams)
    - Interaction occurrence
  - Activity edges connecting these to show the flow.
- Interaction diagrams are not used commonly because these diagrams:
  - Do not provide much insight into the behavior of the system
  - May turn out to be large
  - May become difficult to manage

UML

Timing Diagram



Timing diagrams are another form of interaction diagram, where the focus is on timing constraints: either for a single object or, more usefully, for a bunch of objects. Let's take a simple scenario based on the pump and hotplate for a coffee pot. Let's imagine a rule that says that at least 10 seconds must pass between the pump coming on and the hotplate coming on. When the water reservoir becomes empty, the pump switches off, and the hotplate cannot stay on for more than 15 minutes more.

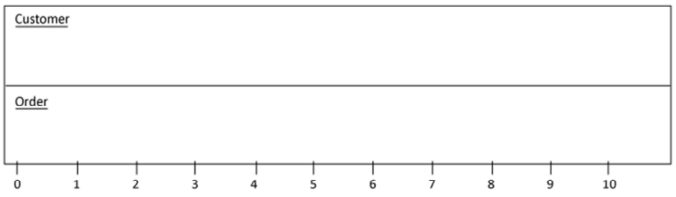
Timing diagrams are useful for showing timing constraints between state changes on different objects. The diagrams are particularly familiar to hardware engineers

Timing Diagrams

- Timing diagrams are interaction diagrams that describe the behavior of each object over a period of time. These diagrams show:
  - How the state and value of an object changes over time
  - Events that cause the change
  - Constrains related to timing of the change and the duration

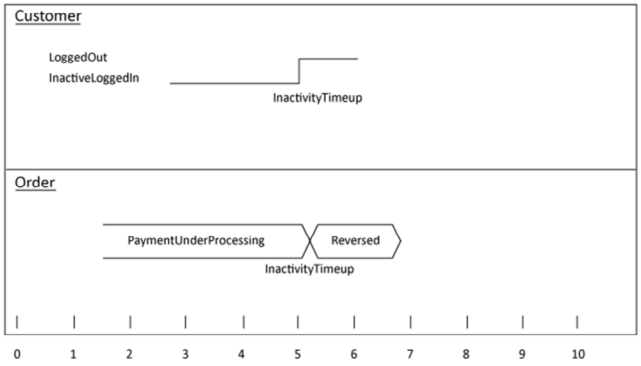
Timing Diagrams (Cont'd)

- A timing diagram consists of the following elements:
  - Time axis – This is a bounding box for the diagram having:
    - X axis, on which the time is specified in the desired units
    - Y axis, on which the states can be specified

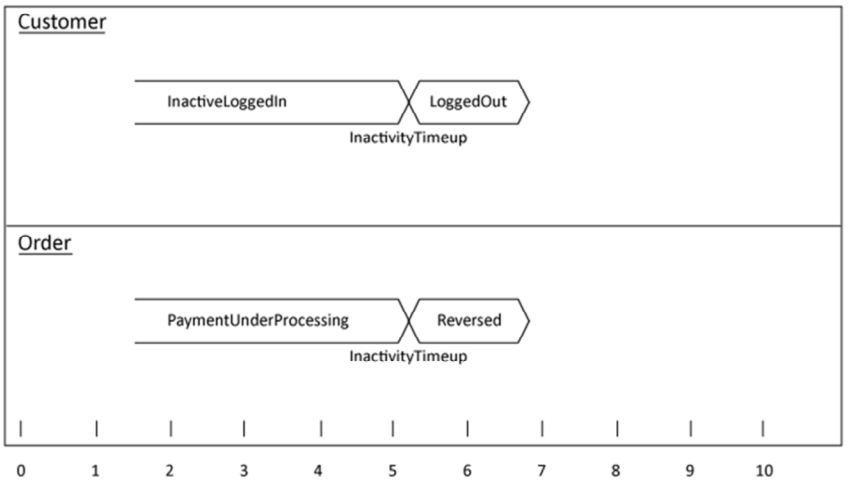


Timing Diagrams (Cont'd)

- Lifelines – These are the lifelines of the object and are plotted on the X and Y axis according to state changes in the object over time or as events occur. Two types of lifelines can be shown in timing diagrams:
  - Robust
  - Concise

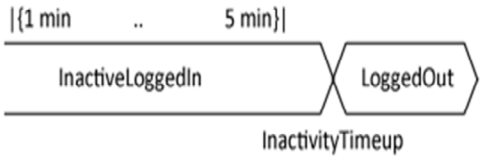


Alternative notation



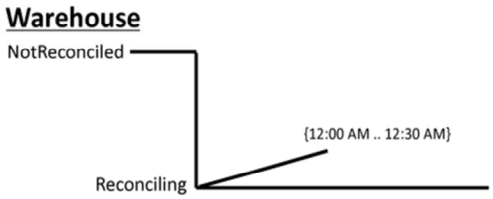
Timing Diagrams (Cont'd)

- Events – Events that cause the state of the object to be changed are marked on the lifeline in the timing diagram.
- Duration constraint – It is a constraint on the state change in a timing diagram and is specified as duration in curly brackets.



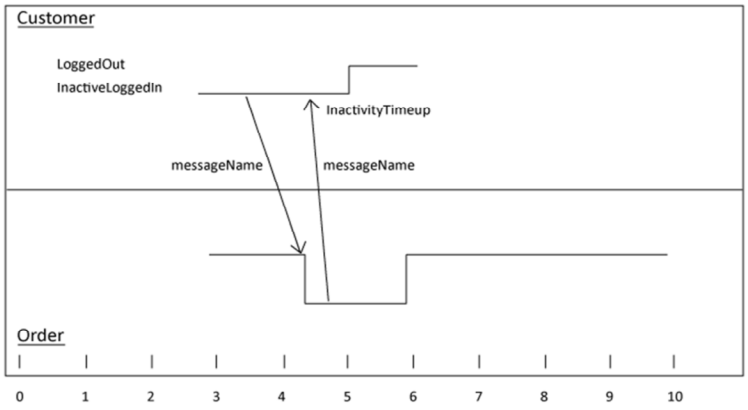
Timing Diagrams (Cont'd)

- Time constraint - It is a constraint that indicates a time interval within which the associated state change must happen.



Timing Diagrams (Cont'd)

- Messages – To depict the message passing between the various objects over a period of time, along with the state changes, you can mark the messages on the robust lifelines as shown below.



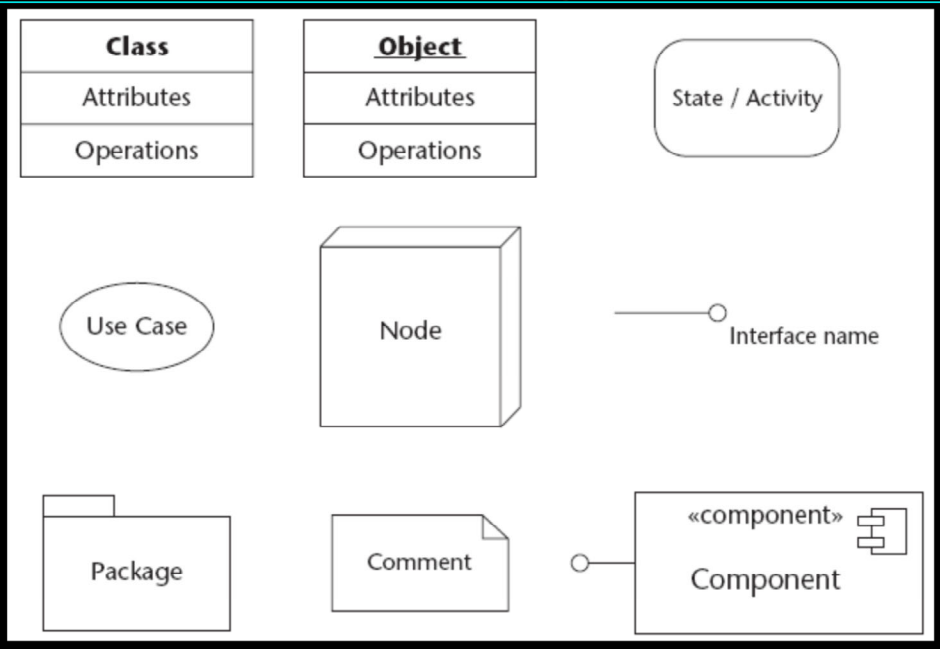
Discussion Time

Discuss the differences between timing diagrams and state diagrams.

UML

Summary

Summary


Class
Attributes
Operations
Object
Attributes
Operations
State / Activity
Use Case
Node
Interface name
Package
Comment
«component»
Component

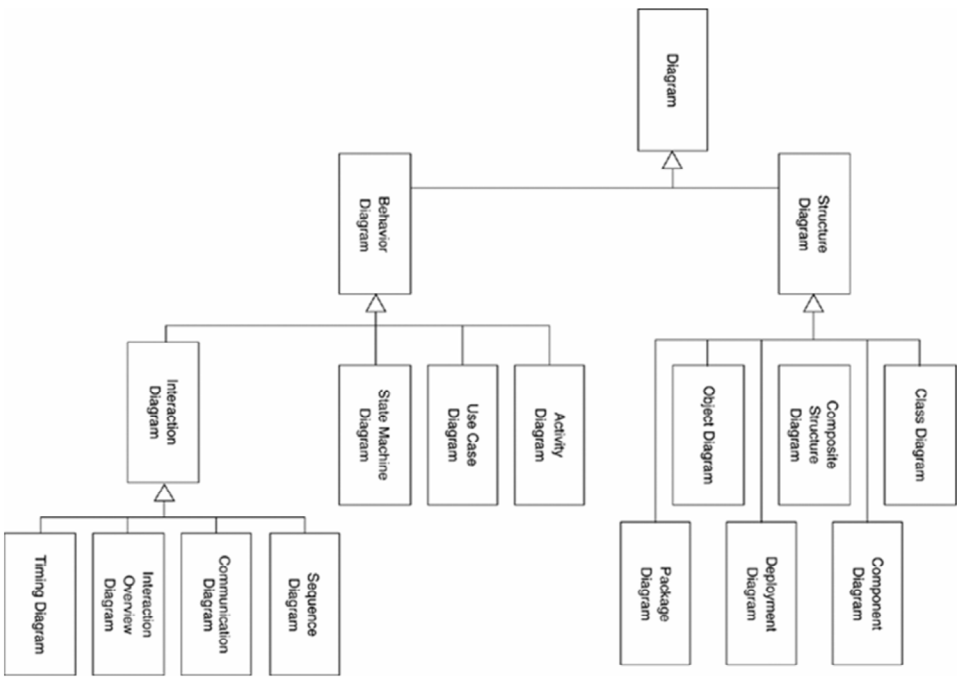
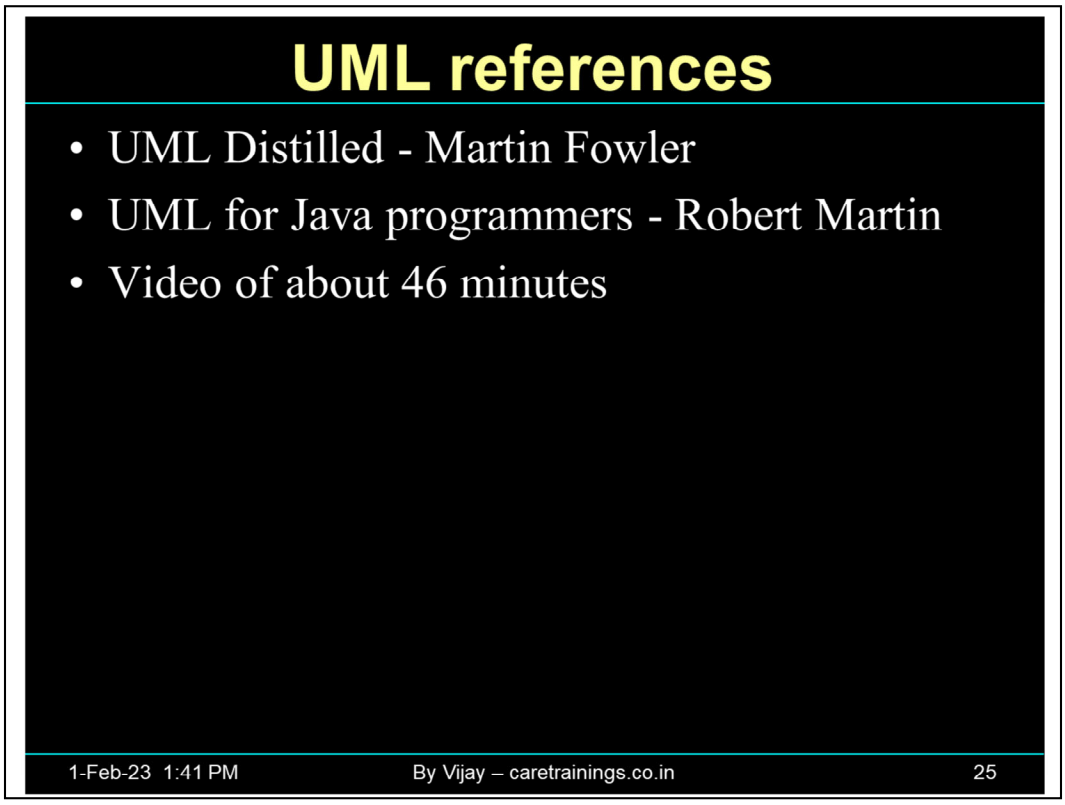


Diagram
Behavior Diagram
Structure Diagram
Interaction Diagram
State Machine Diagram
Use Case Diagram
Activity Diagram
Object Diagram
Composite Structure Diagram
Class Diagram
Timing Diagram
Interaction Overview Diagram
Communication Diagram
Sequence Diagram
Package Diagram
Deployment Diagram
Component Diagram



Same video is at two locations:

1. In Github
2. https://youtu.be/nPQyO02dNHg

Important points in this video are:

- Only a good programmer can become a good modeler.
- Model on giant whiteboards.
- Model a lot at the start of the sprint/analysis phase.
- You don't need tools to model.
- Only truth is code. UML diagrams assist in writing/understanding code.
- Bad idea: One person models and tells others to code exactly as per the model.
- Notation is not very important; Notation is good if it conveys the message.
- Sub teams can see each other's UML diagrams.
- Diagram with programmers and analysts together for shared understanding.
- Use Wikis.
- Never model alone.
- Multiple models go together, e.g., Software class diagram and interaction diagram.

- Any UML diagram before coding finishes will become wrong.
- Document important aspects of the project at the end of the project.
- We model to have a conversation.
- Keep walls free for a whiteboard.
- UML tool is very useful for reverse engineering i.e. generating diagrams from code.
- In design, communication diagrams are important.

The End

"Out of intense complexities, intense simplicities emerge."

—Winston Churchill