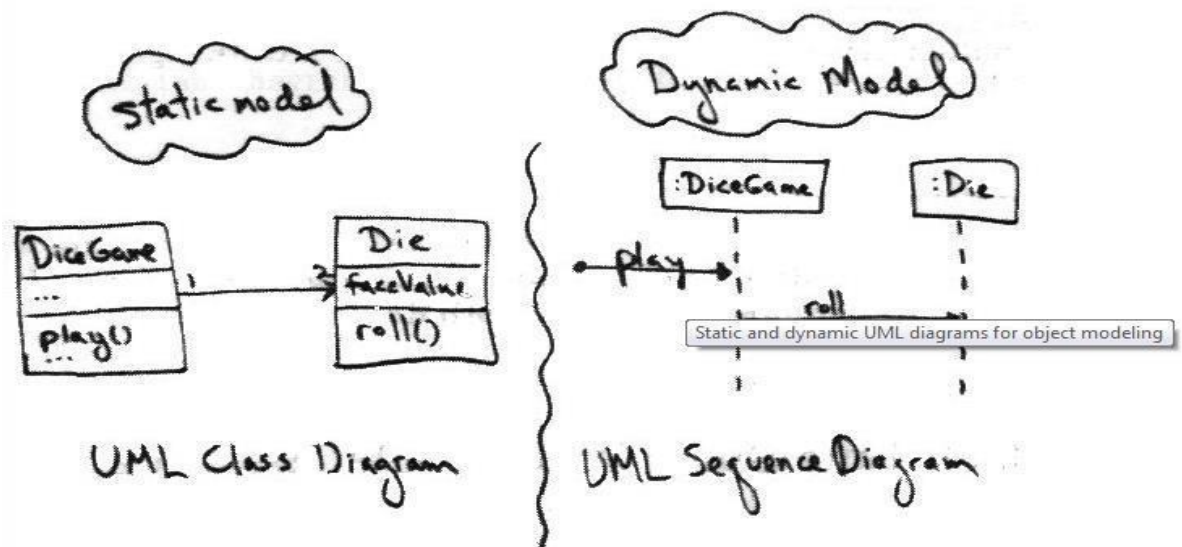


DYNAMIC DIAGRAMS

There are two kinds of object models: dynamic and static. **Dynamic models**, such as UML interaction diagrams (sequence diagrams or communication diagrams), State chart diagram, Activity diagram, help design the logic, the behavior of the code or the method bodies. They tend to be the more interesting, difficult, important diagrams to create. **Static models**, such as UML class diagrams, help design the definition of packages, class names, attributes, and method signatures (but not method bodies).

Static and dynamic UML diagrams for object modeling



The main behavior or dynamic diagrams in UML are

- Interaction diagrams are:
 - Sequence diagrams
 - Collaboration diagrams
- State chart diagrams
- Activity diagrams

UML INTERACTION DIAGRAMS

The UML includes interaction diagrams to illustrate how objects interact via messages. They are used for dynamic object modeling. There are two common types:

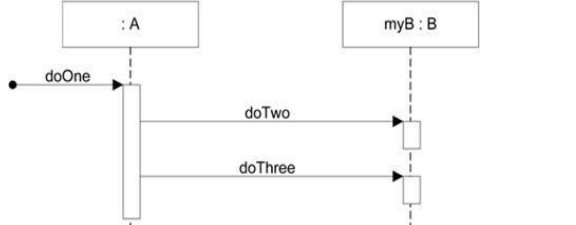
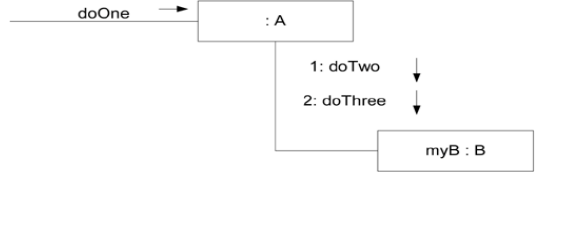
- Sequence Diagram
- Communication Diagram

Ex: Sequence and Communication Diagrams

```
public class A
{private B myB = new B();
```

```
public void doOne()
{ myB.doTwo();
  myB.doThree();
}}
```

Class A has a method named doOne and an attribute of type B. Also, that class B has methods named doTwo and doThree.

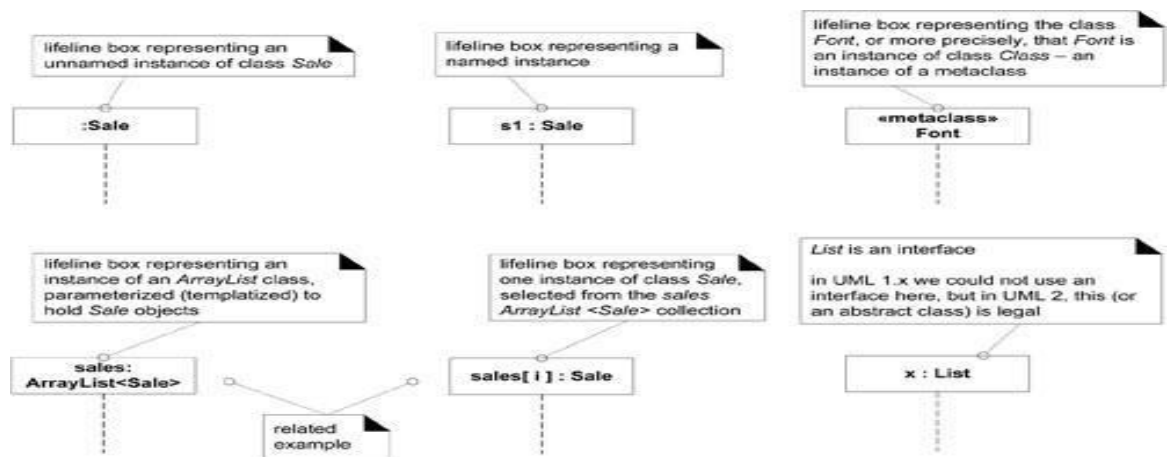
Sequence Diagram	Communication Diagram
Sequence diagrams illustrate interactions in a kind of fence format, in which each new object is added to the right	illustrate object interactions in a graph or network format, in which objects can be placed anywhere on the diagram
	

Strengths and Weaknesses of Sequence vs. Communication Diagrams

Type	Strengths	Weaknesses
sequence	clearly shows sequence or time ordering of messages large set of detailed notation options	forced to extend to the right when adding new objects; consumes horizontal space fewer notation options
communication	space economical flexibility to add new objects in two dimensions	more difficult to see sequence of messages

Common UML Interaction Diagram Notation

Lifeline boxes to show participants in interactions.



Basic Message Expression Syntax

The UML has a standard syntax for these message expressions:

return = message(parameter : parameterType) : returnType

Parentheses are usually excluded if there are no parameters, though still legal.

For example:

initialize(code)

initialize

d = getProductDescription(id)

d = getProductDescription(id:ItemID)

d = getProductDescription(id:ItemID) : ProductDescription

Singleton Objects

There is only one instance of a class instantiated never two. it is a "singleton" instance. In a UML interaction diagram (sequence or communication), such an object is marked with a '1' in the upper right corner of the lifeline box.

Singletons in interaction diagrams.



Basic Sequence Diagram Notation

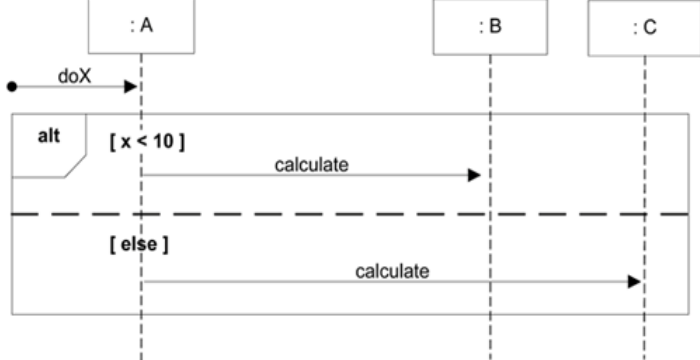
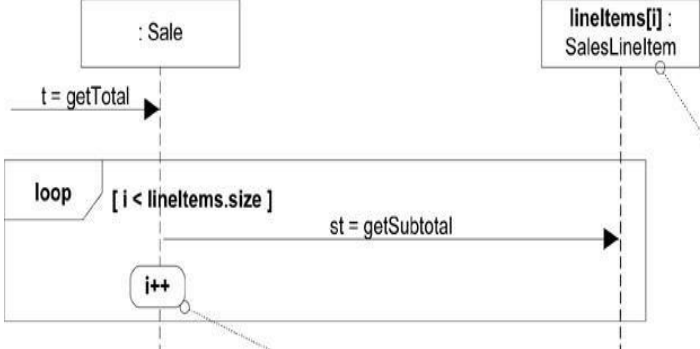
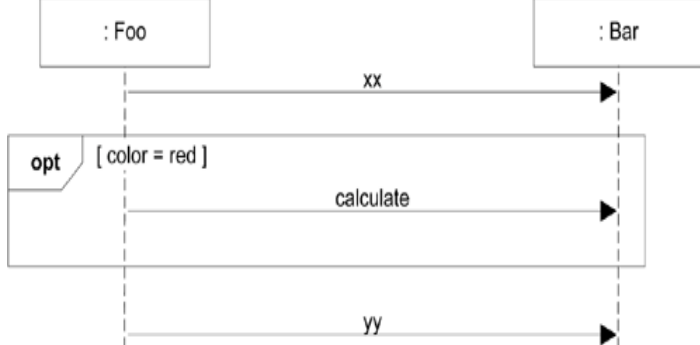
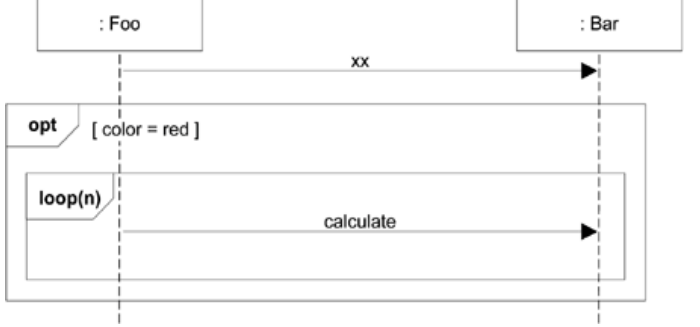
Name	Symbol	Description
Lifeline Boxes and Lifelines , Messages		<p>All UML examples show the lifeline as dashed (because of UML 1 influence), in fact the UML 2 specification says it may be solid or dashed.</p> <p>Found message the sender will not be specified, is not known, or that the message is coming from a random source.</p>

Reply or Return Msgs	<pre> sequenceDiagram participant Register as :Register participant Sale as :Sale Register->>Register: doX Register->>Sale: d1 = getDate Sale-->>Register: aDate </pre>	<p>There are two ways to show the return result from a message:</p> <ol style="list-style-type: none"> 1. Using the message syntax returnVar = message(parameter). 2. Using a reply (or return) message line at the end of an activation bar.
Messages to "self" or "this"	<pre> sequenceDiagram participant Register as :Register Register->>Register: doX activate Register Register->>Register: clear deactivate Register </pre>	<p>message being sent from an object to itself by using a nested activation bar</p>
Creation of Instances and Object Destruction	<pre> sequenceDiagram participant Sale as :Sale participant Payment as :Payment Sale->>Payment: create(cashTendered) activate Payment Sale->>Payment: «destroy» destroy Payment </pre>	<p>It is desirable to show explicit destruction of an object. The UML lifeline notation provides a way to express this destruction</p>

Frames

To support conditional and looping constructs (among many other things), the UML uses frames. Frames are regions or fragments of the diagrams; they have an operator or label (such as loop) and a guard (conditional clause).

Frame Operator	Meaning
alt	Alternative fragment for mutual exclusion conditional logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write loop(n) to indicate looping n times. There is discussion that the specification will be enhanced to define a FOR loop, such as loop(i, 1, 10)
opt	Optional fragment that executes if guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

Alt Frame	 <pre> sequenceDiagram participant A as : A participant B as : B participant C as : C A->>A: doX alt [x < 10] A->>B: calculate else [else] A->>C: calculate end </pre>	<p>Alternative fragment for mutual exclusion conditional logic expressed in the guards.</p>
Loop Frame	 <pre> sequenceDiagram participant Sale as : Sale participant lineItems as lineItems[i] : SalesLineItem Sale->>Sale: t = getTotal loop [i < lineItems.size] Sale->>lineItems: st = getSubtotal Note over Sale: i++ end </pre>	<p>Loop fragment while guard is true. Can also write loop(n) to indicate looping n times. the specification will be enhanced to define a FOR loop, such as loop(i, 1, 10)</p>
Opt Frame	 <pre> sequenceDiagram participant Foo as : Foo participant Bar as : Bar Foo->>Bar: xx opt [color = red] Foo->>Bar: calculate end Foo->>Bar: yy </pre>	<p>Optional fragment that executes if guard is true.</p>
Nesting of frames	 <pre> sequenceDiagram participant Foo as : Foo participant Bar as : Bar Foo->>Bar: xx opt [color = red] loop(n) Foo->>Bar: calculate end end </pre>	<p>Frames can be nested</p>

Relating Interaction Diagrams		<p>An interaction occurrence (also called an interaction use) is a reference to an interaction within another interaction. It is useful, for example, when you want to simplify a diagram and factor out a portion into another diagram, or there is a reusable interaction occurrence.</p>
to Invoke Static (or Class) Methods		<p>The classes Class and Type are metaclasses, which means their instances are themselves classes. A specific class, such as class Calendar, is itself an instance of class Class. Thus, class Calendar is an instance of a metaclass.</p>
Asynchronous and Synchronous Calls		<p>An asynchronous message call does not wait for a response. They are used in multi-threaded environments such as .NET and Java.</p> <p>The UML notation for asynchronous calls is a stick arrow message; regular synchronous (blocking) calls are shown with a filled arrow.</p>

Basic Communication Diagram Notation

Name	Symbol	Description
Links , Messages		<p>A link is a connection path between two objects; it indicates some form of navigation and visibility between the objects</p> <p>Each message between objects is represented with a message expression , direction of the message , message Number</p>
Messages to "self" or "this"		<p>A message can be sent from an object to itself.</p> <p>This is illustrated by a link to itself, with messages flowing along the link.</p>
Creation of Instances		<p>The message may be annotated with a UML stereotype, like so: «create». The create message may include parameters, indicating the passing of initial values.</p>
Message Number Sequencing		<p>The first message is not numbered. Thus, msg1 is unnumbered.</p> <p>The order and nesting of subsequent messages is shown with a legal numbering scheme in which nested messages have a number appended to them.</p>
Conditional Messages		<p>A conditional message by following a sequence number with a conditional clause in square brackets, similar to an iteration clause. The message is only sent if the clause evaluates to true.</p>
Mutually Exclusive Conditional Paths		<p>modify the sequence expressions with a conditional path letter. The first letter used is a by convention.</p> <p>either 1a or 1b could execute after msg1. Both are sequence number 1 since either could be the first internal message.</p>

Iteration or Looping		a simple * can be used
Messages to a Classes to Invoke Static (Class) Methods		Meta class stereotype is used to represent static method call
Asynchronous and Synchronous Calls		asynchronous calls are shown with a stick arrow; synchronous calls with a filled arrow.

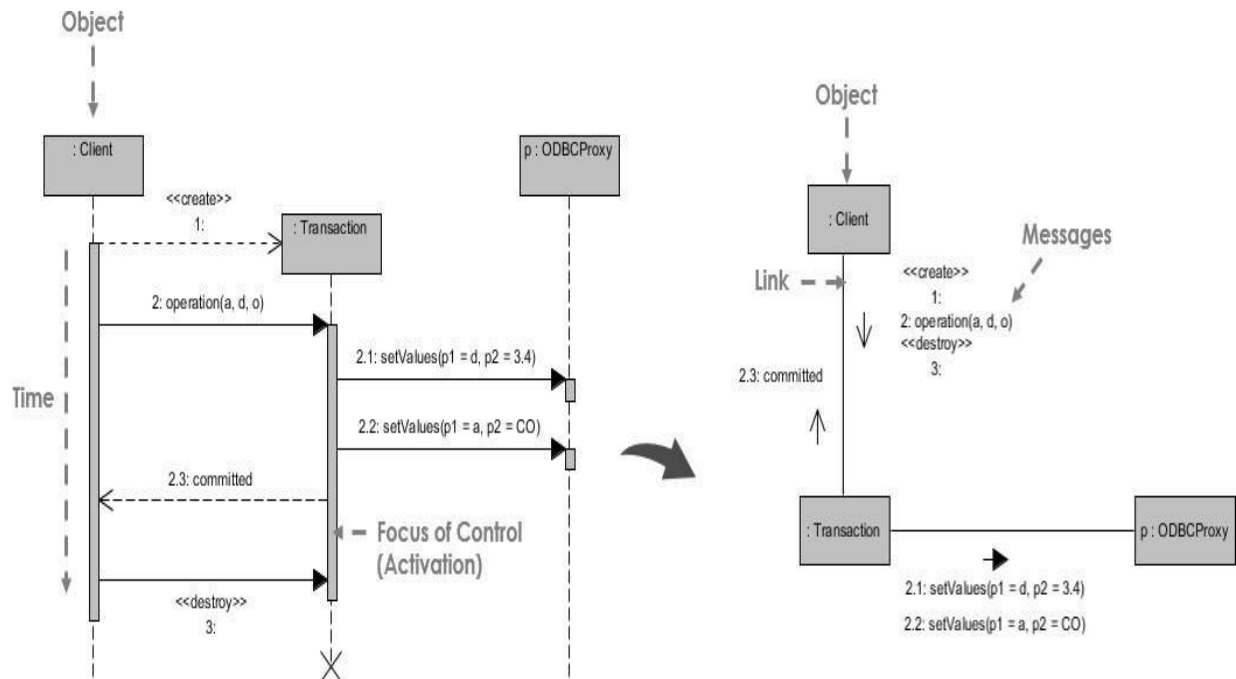
WHEN TO USE COMMUNICATION DIAGRAMS

The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle. The purpose of interaction diagram is –

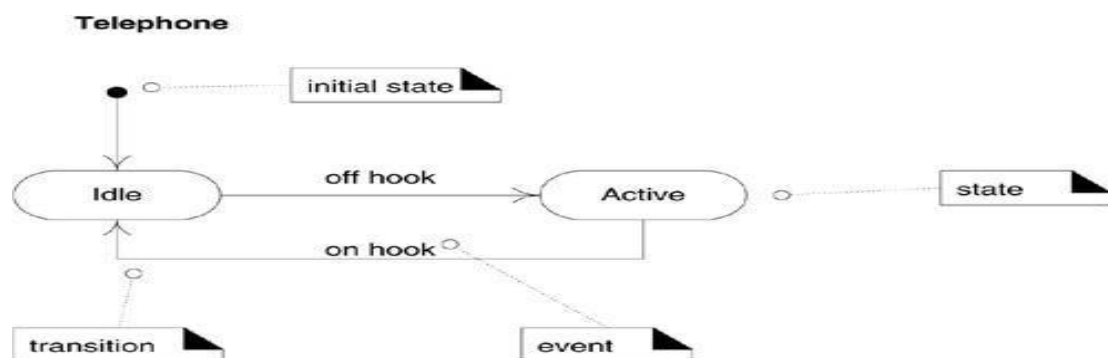
- To capture the dynamic behavior of a system.
- Model message passing between objects or roles that deliver the functionalities of use cases and operations
- To describe the structural organization of the objects.
- To describe the interaction among objects. Support the identification of objects (hence classes), and their attributes (parameters of message) and operations (messages) that participate in use cases

Sequence diagram vs Communication diagram Example



UML STATE MACHINE DIAGRAMS AND MODELLING

A UML state machine diagram illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event. Transitions are shown as arrows, labeled with their event. States are shown in rounded rectangles. It is common to include an initial pseudo-state, which automatically transitions to another state when the instance is created.



State machine diagram for a telephone

A state machine diagram shows the lifecycle of an object: what events it experiences, its transitions, and the states it is in between these events. Therefore, we can create a state machine diagram that describes the lifecycle of an object at arbitrarily simple or complex levels of detail, depending on our needs.

Definitions: Events, States, and Transitions

An **event** is a significant or noteworthy occurrence. For example:

- A telephone receiver is taken off the hook.

A **state** is the condition of an object at a moment in time the time between events. For example:

- A telephone is in the state of being "idle" after the receiver is placed on the hook and until it is taken off the hook.

A **transition** is a relationship between two states that indicates that when an event occurs, the object moves from the prior state to the subsequent state. For example:

- When the event "off hook" occurs, transition the telephone from the "idle" to "active" state.

Guidelines : To Apply State Machine Diagrams:

Object can be classified into

- 1) **State-Independent Object** - If an object always responds the same way to an event, then it is considered state-independent (or modeless) with respect to that event. The object is state-independent with respect to that message.
- 2) **State-Dependent Objects** - State-dependent objects react differently to events depending on their state or mode.

Guideline : Consider state machines for state-dependent objects with complex behavior, not for state-independent objects . For example, a telephone is very state-dependent. The phone's reaction to pushing a particular button (generating an event) depends on the current mode of the phone off hook, engaged, in a configuration subsystem, and so forth.

Modeling State-Dependent Objects : state machines are applied in two ways:

1. To model the behavior of a **complex reactive object** in response to events.
2. To model **legal sequences of operations** protocol or language specifications.
 - This approach may be considered a specialization of #1, if the "object" is a language, protocol, or process. A formal grammar for a context-free language is a kind of state machine.

1. Complex Reactive Objects

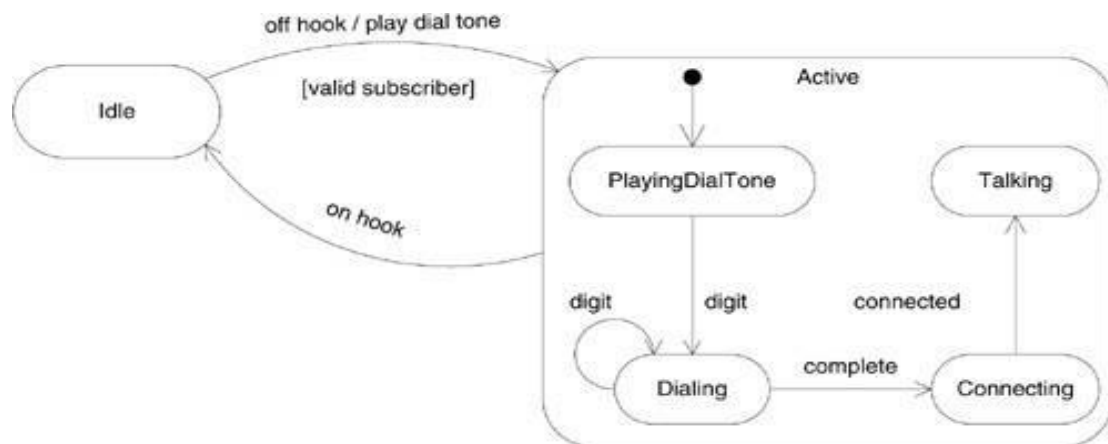
- a) **Physical Devices controlled by software**
 - Phone, car, microwave oven: They have complex and rich reactions to events, and the reaction depends upon their current mode.
- b) **Transactions and related Business Objects**
 - How does a business object (a sale, order, payment) react to an event? For example, what should happen to an Order if a cancel event occurs? And understanding all the events and states that a Package can go

through in the shipping business can help with design, validation, and process improvement.

- c) **Role Mutators** : These are objects that change their role.
 - o A Person changing roles from being a civilian to a veteran. Each role is represented by a state.

Example 1: Physical Devices / Nested States – Telephone Object

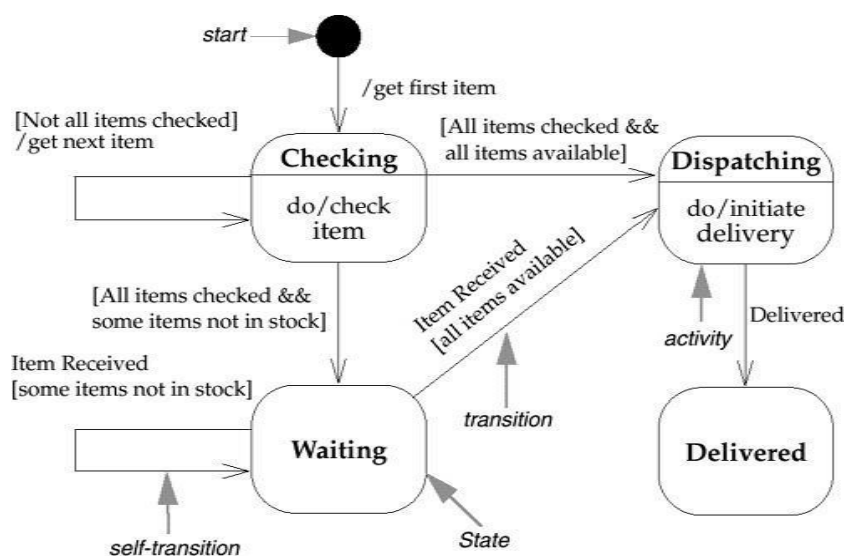
A state allows nesting to contain substates; a substate inherits the transitions of its superstate (the enclosing state). It may be graphically shown by nesting them in a superstate box..



For example, when a transition to the Active state occurs, creation and transition into the PlayingDialTone substate occurs. No matter what substate the object is in, if the on hook event related to the Active superstate occurs, a transition to the Idle state occurs.

Example 2: Transactions and related Business Objects

Order Processing System – Order Object



2) Protocols and Legal Sequences

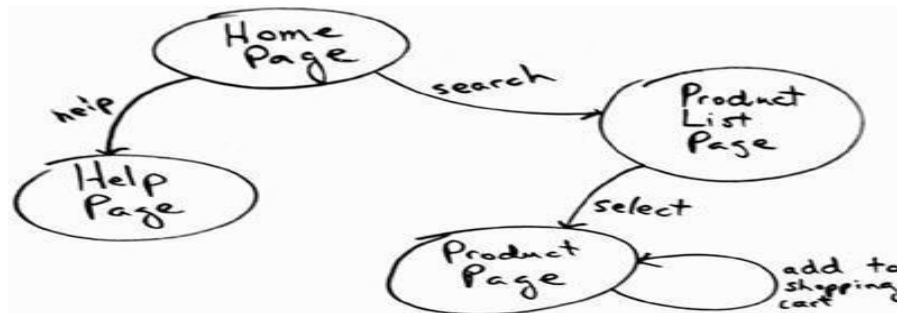
a) Communication Protocols

- TCP, and new protocols, can be easily and clearly understood with a state machine diagram. For example, a TCP "close" request should be ignored if the protocol handler is already in the "closed" state.

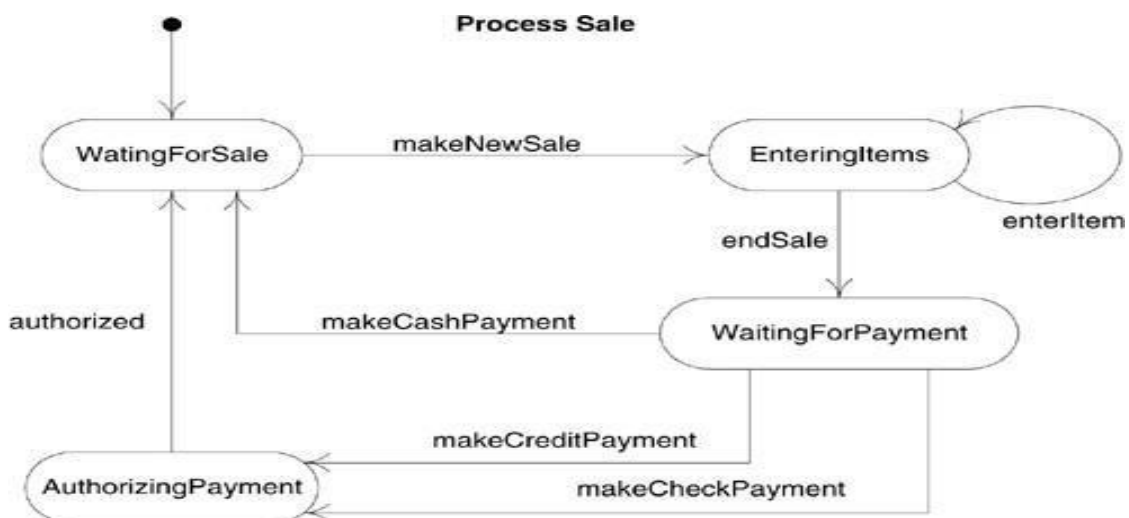
b) UI Page/Window Flow or Navigation

When doing UI modeling, it can be useful to understand the legal sequence between Web pages or windows;

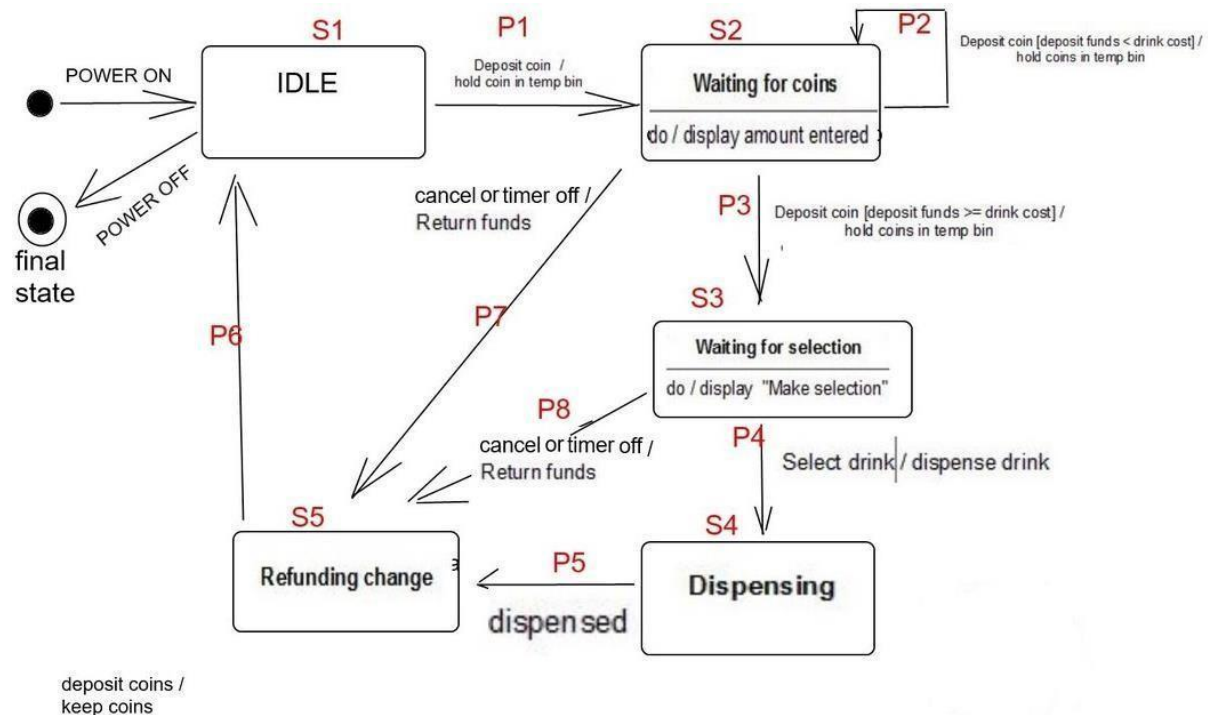
Applying a state machine to Web page navigation modeling.



- ### c) UI Flow Controllers or Sessions
- These are usually server-side objects representing an ongoing session or conversations with a client. For example, a Web application that remembers the state of the session with a Web client and controls the transitions to new Web pages, or the modified display of the current Web page, based upon the state of the session and the next operation that is received.
- ### d) Use Case System Operations
- Do you recall the system operations for Process Sale: makeNewSale, enterItem etc. These should arrive in a legal order; for example, endSale should only come after one or more enterItem operations.



STATE DIAGRAM : COFFEE VENDING MACHINE



WHEN TO USE STATE DIAGRAM

State chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

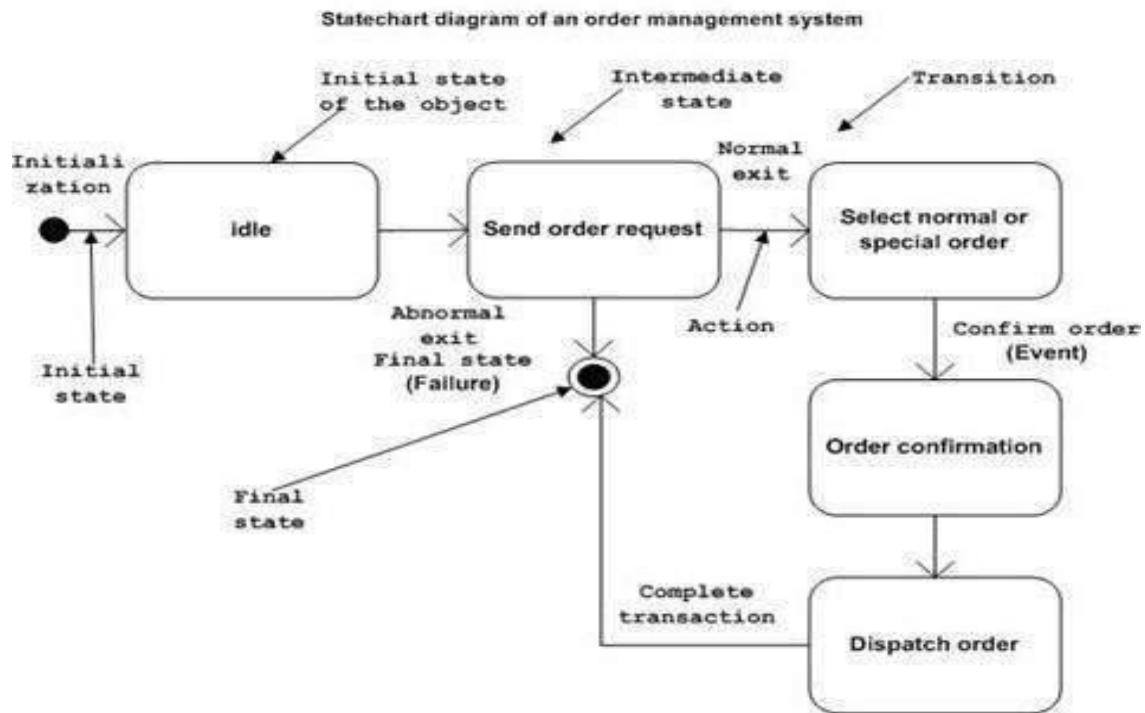
State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of State chart diagram is to model lifetime of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

The main usage can be described as –

- To model the object states of a system.
- To model the reactive system. Reactive system consists of reactive objects.
- To identify the events responsible for state changes.
- Forward and reverse engineering.

Example:



UML ACTIVITY DIAGRAMS





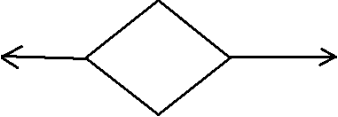
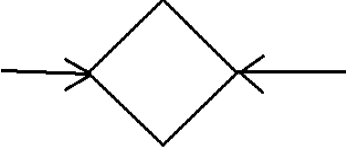

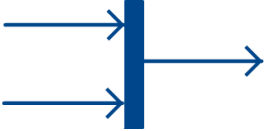

A UML activity diagram shows sequential and parallel activities in a process. They are useful for modeling business processes, workflows, data flows, and complex algorithms.

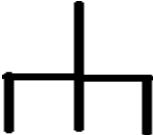


Purpose: To understand & Communicate the structure & dynamics of the organization in which a system is to be deployed.

Elements:

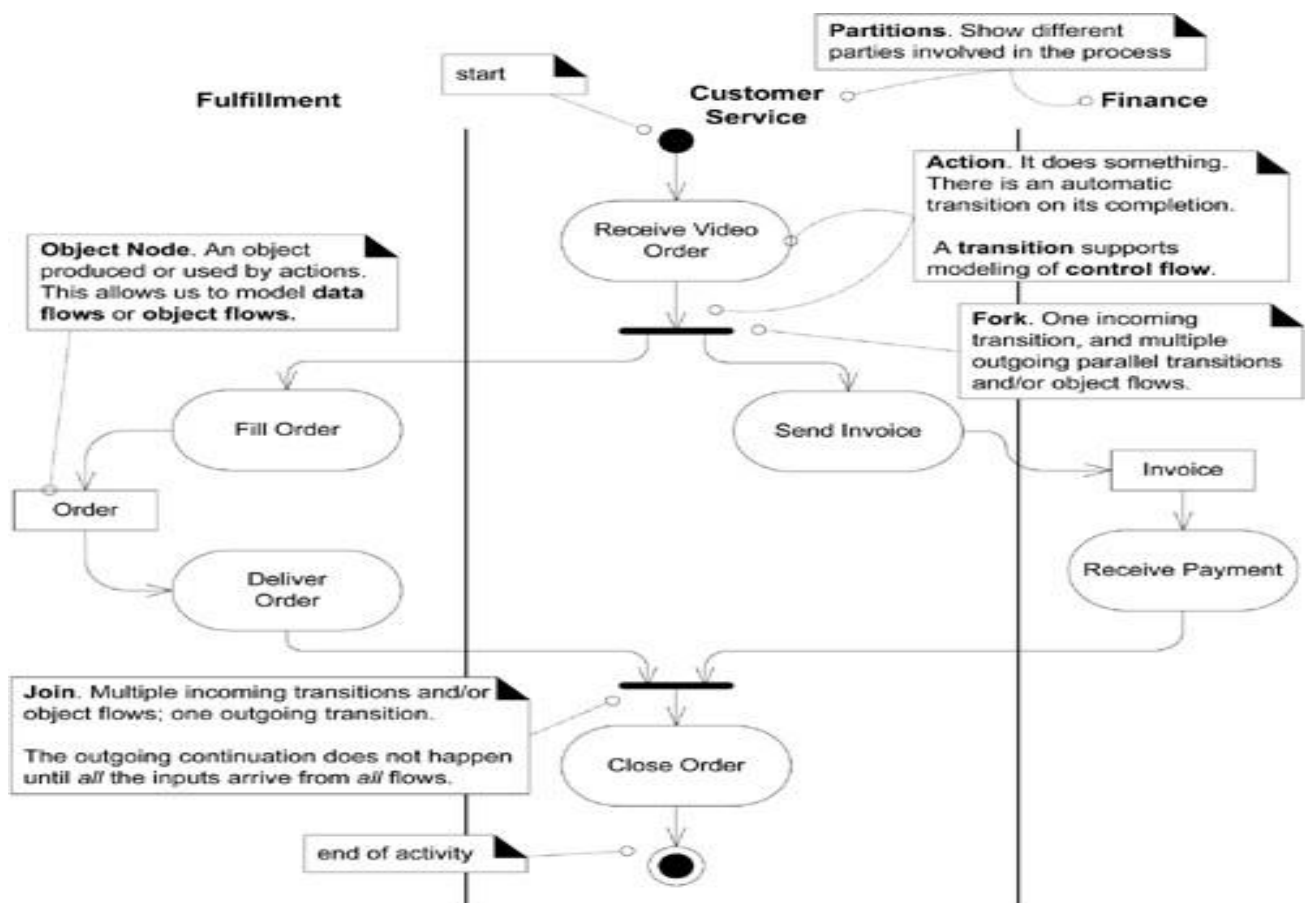
Start , end, activity ,action, object , fork ,join, decision , merge, time signal , rake , accept signal , swim lane

Symbol	Name	Use
	Start/ Initial Node	Used to represent the starting point or the initial state of an activity
	Activity / Action State	Used to represent the activities of the process

	Action	Used to represent the executable sub-areas of an activity
	Object	Used to represent the object
	Control Flow / Edge	Used to represent the flow of control from one action to the other
	Object Flow / Control Edge	Used to represent the path of objects moving through the activity
	Decision Node	Used to represent a conditional branch point with one input and multiple outputs
	Merge Node	Used to represent the merging of flows. It has several inputs, but one output.
	Fork	Used to represent a flow that may branch into two or more parallel flows
	Merge	Used to represent a flow that may branch into two or more parallel flows
	Signal Receipt	Used to represent that the signal is received

	rake	Further can be expanded into sub activity diagram
	swim lane	Divides the diagram into vertical zones
	Time Signal	Timing condition can be specified

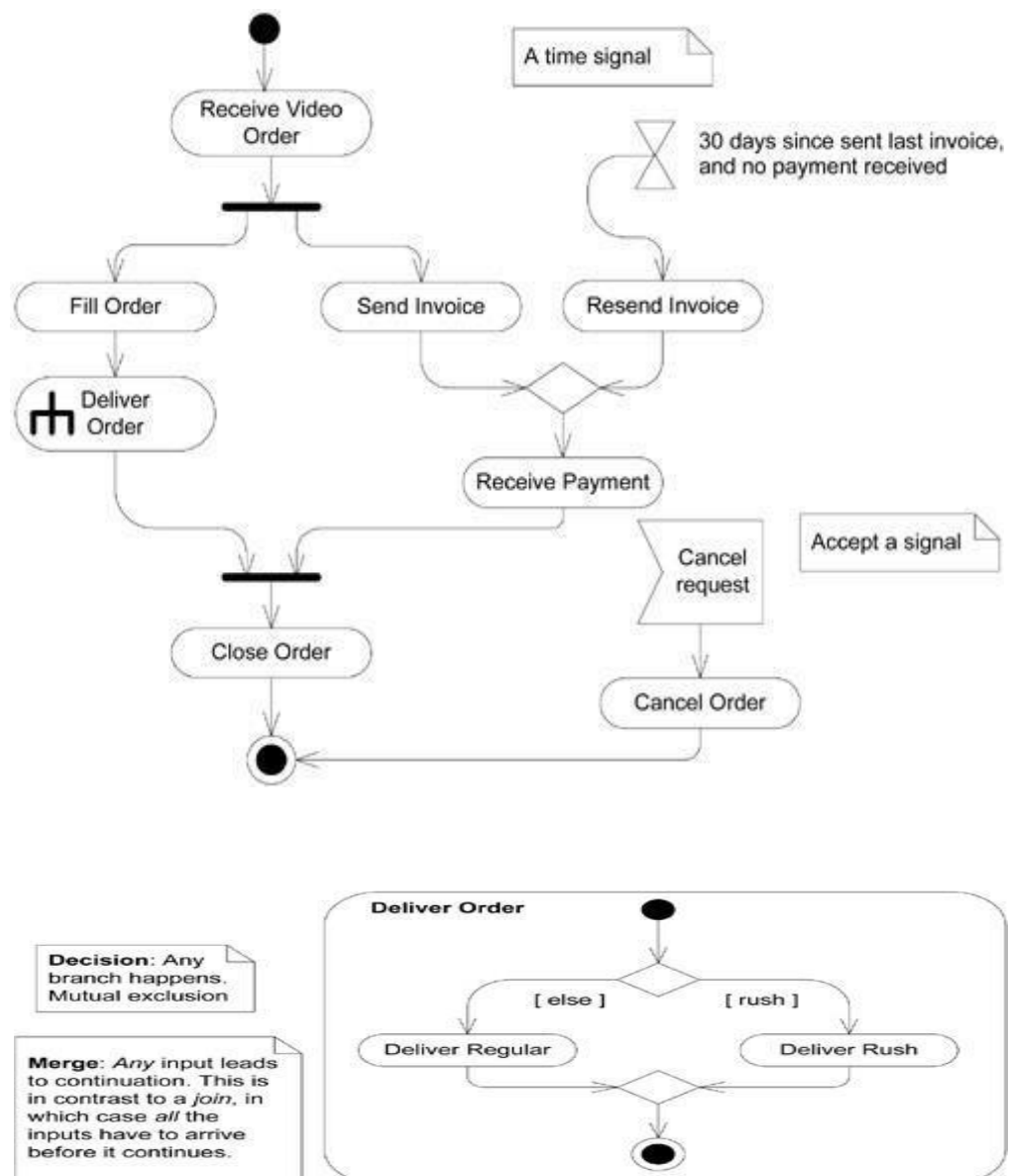
Example: Showing usage of symbols



Example

This diagram shows a sequence of actions, some of which may be parallel. Two points to remember :

- once an action is finished, there is an automatic outgoing transition
- the diagram can show both control flow and data flow



Guideline to Apply Activity Diagrams

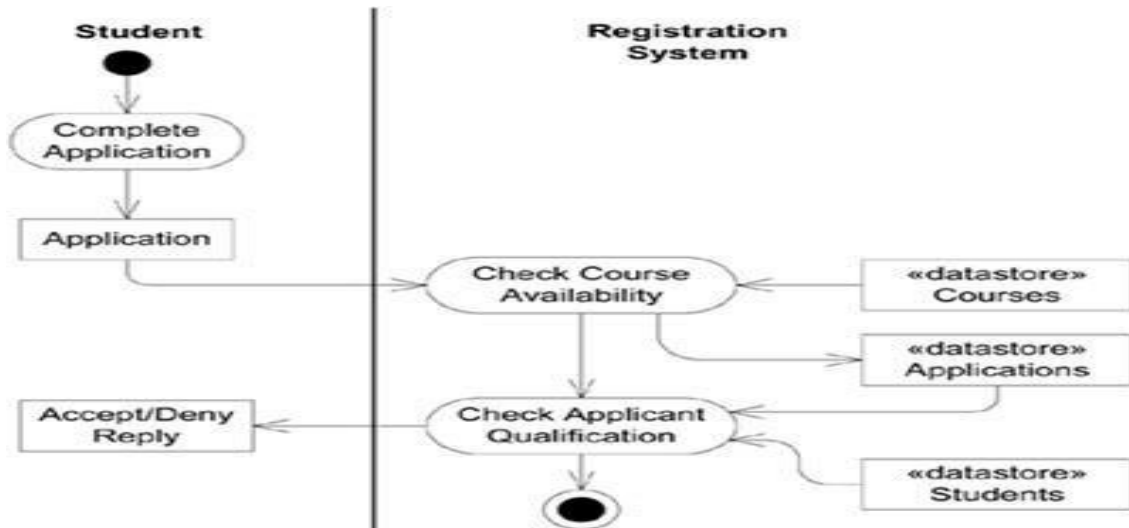
1. Business Process Modeling

Client uses activity diagrams to understand their current complex business processes by visualizing them. The partitions are useful to see the multiple parties and parallel actions involved in the shipping process, and the object nodes illustrate what's moving around.

Ex: Borrow books return books usecase of Library Information system

2. Data Flow Modeling

data flow diagrams (DFD) became a popular way to visualize the major steps and data involved in software system processes. This is not the same as business process modeling; rather, DFDs were usually used to show data flows in a computer system, although they could in theory be applied to business process modeling.



3. Concurrent Programming and Parallel Algorithm Modeling

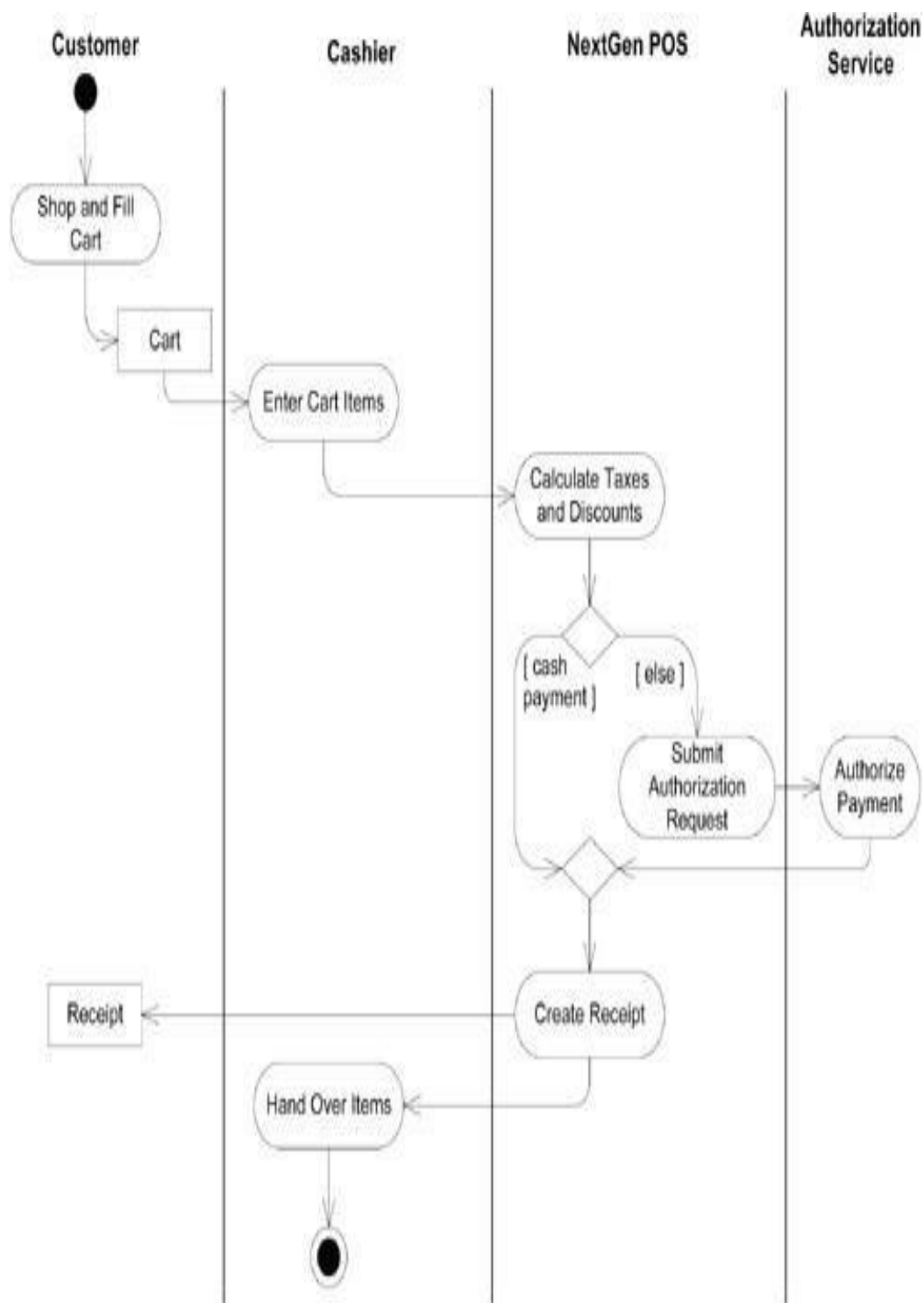
Parallel algorithms in concurrent programming problems involve multiple partitions, and fork and join behavior. The UML activity diagram partitions can be used to represent different operating system threads or processes. The object nodes can be used to model the shared objects and data. Forking can be used to model the creation and parallel execution of multiple threads or processes, one per partition.

4. Guidelines

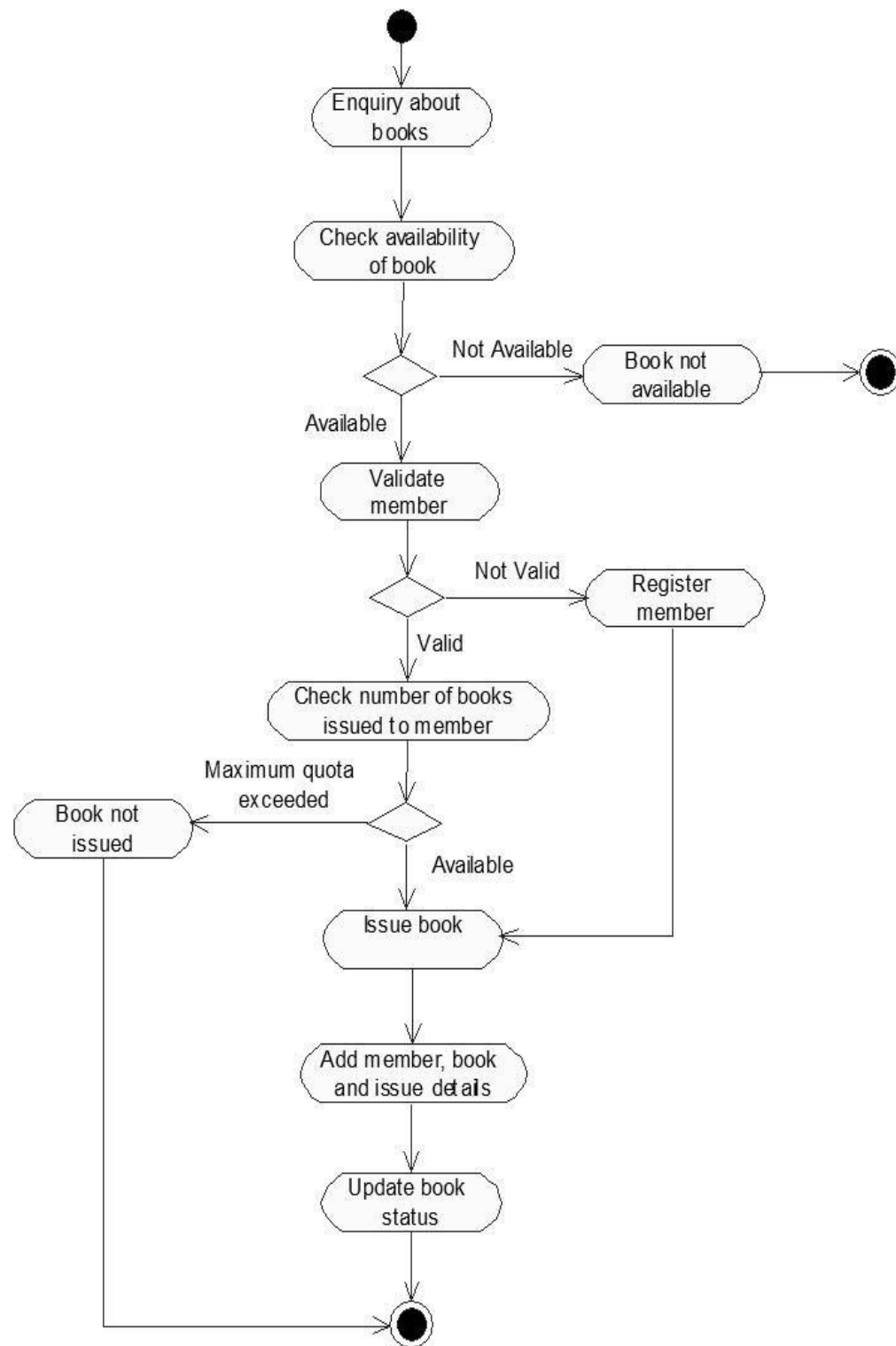
- If modeling a business process, take advantage of the "rake" notation and sub-activity diagrams. On the first overview "level 0" diagram, keep all the actions at a very high level of abstraction, so that the diagram is short and sweet. Expand the details in sub-diagrams at the "level 1" level, and perhaps even more at the "level 2" level, and so forth.

Example: NextGen Activity Diagram

The partial model in Figure illustrates applying the UML to the Process Sale use case process.



Activity Diagram Ex1: Borrow Books (Library Information System)



WHEN TO USE ACTIVITY DIAGRAMS

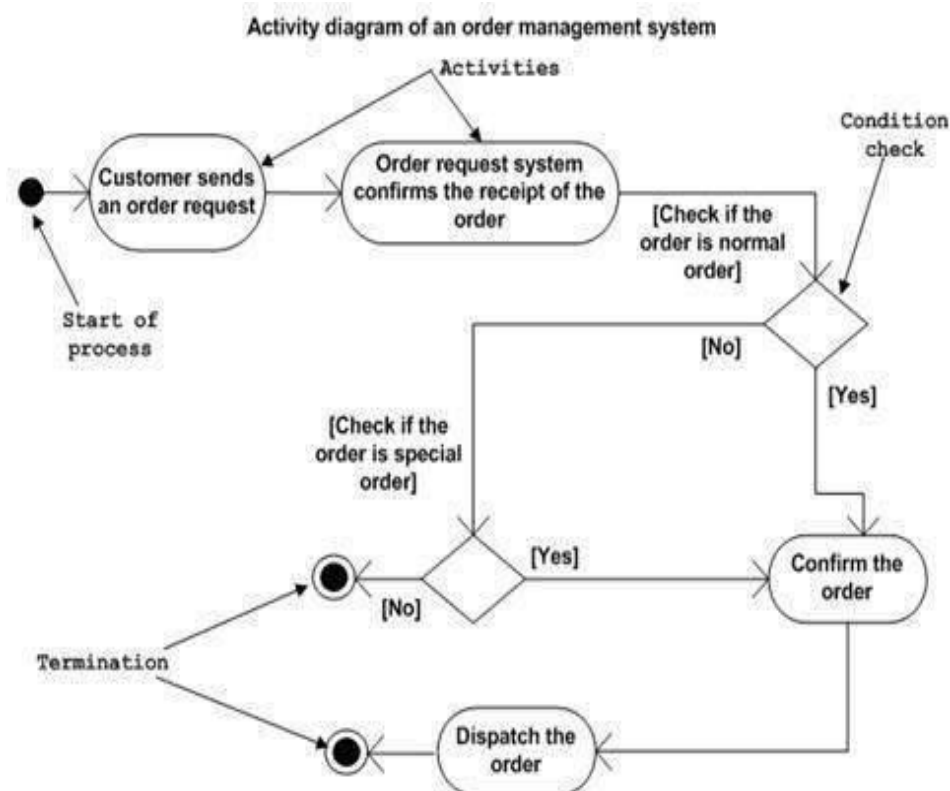
Activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes the flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues, or any other system.

Activity diagram gives high level view of a system. This high level view is mainly for business users or any other person who is not a technical person. This diagram is used to model the activities of business requirements. The diagram has more impact on business understanding rather than on implementation details.

Activity diagram can be used for –

- Modeling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

Example



UML PACKAGE DIAGRAMS

UML package diagrams are often used to illustrate the logical architecture of a system-the layers, subsystems, packages . A layer can be modeled as a UML package; It is part of the Design Model and also be summarized as a view in the Software Architecture Document.

Logical Architecture is the large scale organization of the software classes into packages subsystem and layers It is called logical architecture because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network.

Layer is a coarse grained grouping of classes , packages, or subsystems that has cohesive responsibility for major aspect of the system .

There are 2 types of Layers.


- 1) Higher Layer (Contain more application specific services ex: UI layer)
- 2) Lower layer (Contain more generalized services ex: Technical Services layer)


Higher Layer calls upon services of lower layer , but vice versa is not .

Typically layers in the Object Oriented System has 7 standard layers. The important layers are

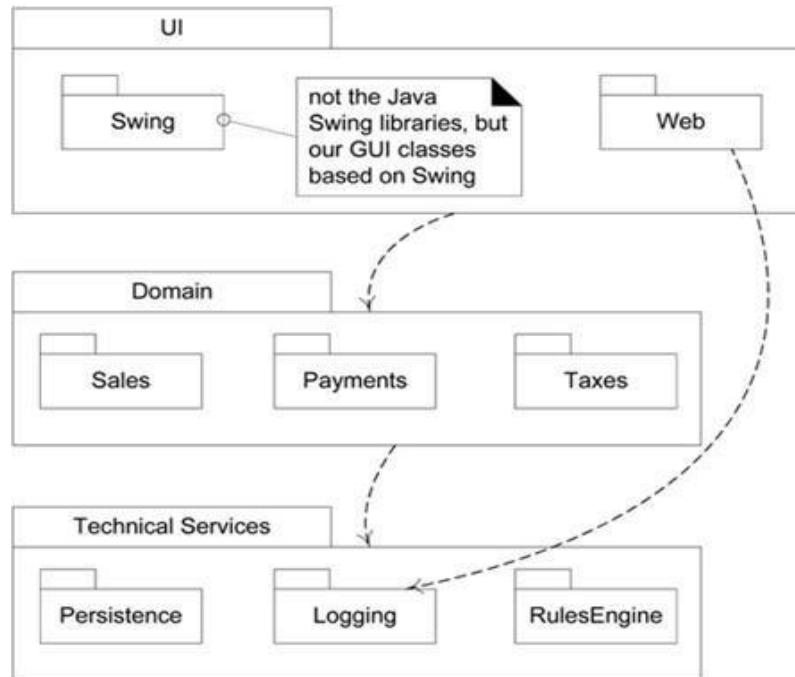
- **User Interface** – Has various I/O formats & forms.
- **Application Logic and Domain Objects** - software objects representing domain concepts (for example, a software class Sale) that fulfill application requirements, such as calculating a sale total.
- **Technical Services** general purpose objects and subsystems that provide supporting technical services, such as interfacing with a database or error logging.

Architecture Types

strict layered architecture  a layer only calls upon the services of the layer directly below it. This design is common in network protocol stacks, but not in information systems

Relaxed layered architecture  a higher layer calls upon several lower layers. Example, the UI layer may call upon its directly subordinate application logic layer, and also upon elements of a lower technical service layer, for logging and so forth.

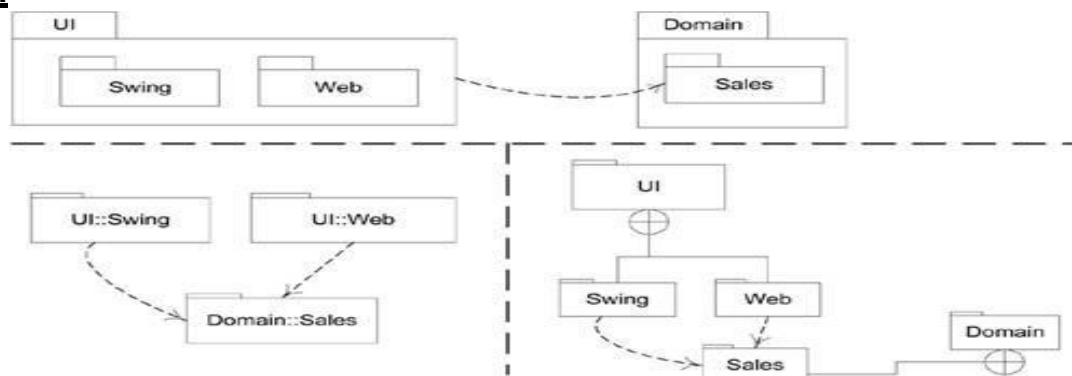
Layers shown with UML package diagram notation.



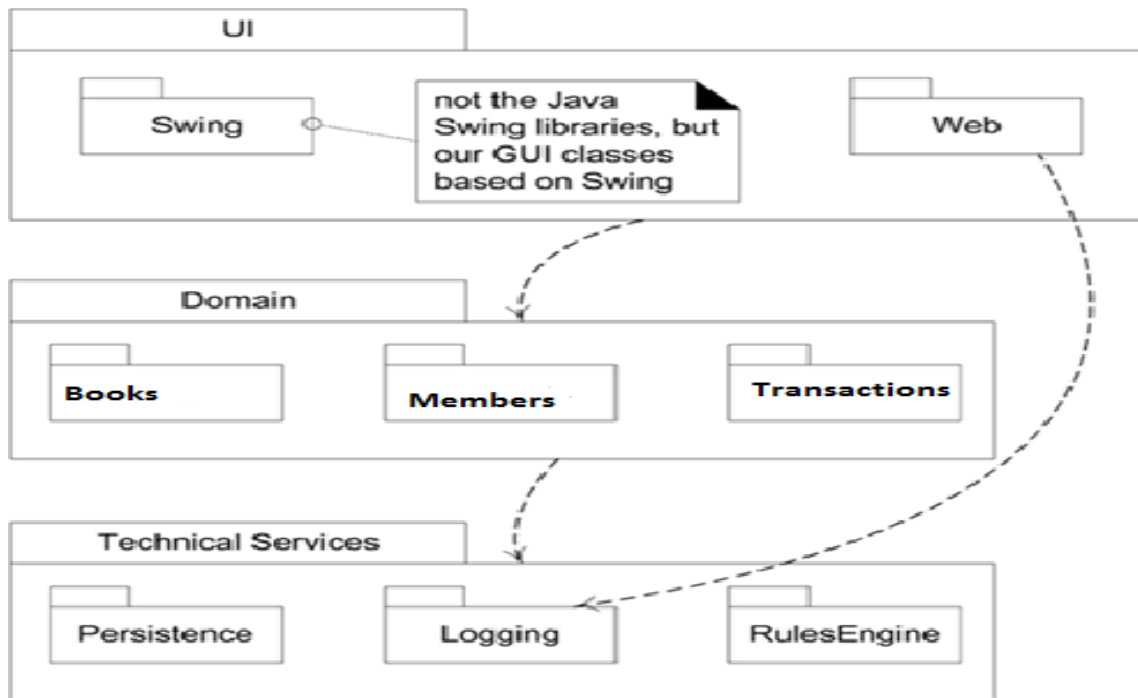
Elements

Name	Symbol	Description
Package		package can group anything: classes, other packages, use cases
Dependency		depended-on package
Fully qualified Name	java::util::Date	To represents a namespace (outer package named "java" with a nested package named "util" with a Date class)

Ex:



Package Diagram : Library Information System



WHEN TO USE PACKAGE DIAGRAMS

1. It is used in large scale systems to picture dependencies between major elements in the system.
2. Package diagrams represent a compile time grouping mechanism.

UML DEPLOYMENT AND COMPONENT DIAGRAMS

Deployment Diagrams : A deployment diagram shows the assignment of concrete software artifacts (such as executable files) to computational nodes (something with processing services). It shows the deployment of software elements to the physical architecture and the communication (usually on a network) between physical elements

Two Nodes of Deployment Diagram :

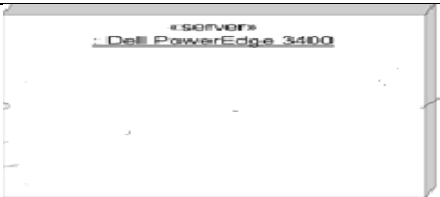
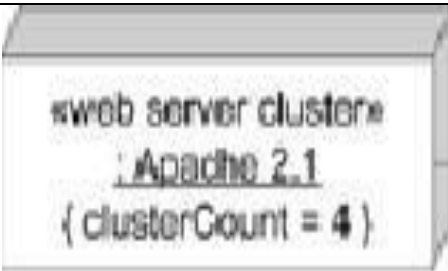

- 1) **Device Node :** This is a Physical Computing Resource representing a computer with memory or mobile.
- 2) **Execution Environment Node :**

This is a software computing resource that runs within an outer node (such as a computer) and which itself provides a service to host and execute other executable software elements. **For example:**

- an operating system (OS) is software that hosts and executes programs
- a virtual machine (VM, such as the Java or .NET VM) hosts and executes programs

- a database engine (such as PostgreSQL) receives SQL program requests and executes them, and hosts/executes internal stored procedures (written in Java or a proprietary language)
- a Web browser hosts and executes JavaScript, Java applets, Flash, and other executable technologies
- a workflow engine
- a servlet container or EJB container

Elements:

Name	Symbol	Description
Device Node :		Physical Computing Resource
Execution Environment Node	 { OS = Linux } { JVM = sun Hot Spot 2.0 }	a software computing resource
Communication path		Connection between nodes with protocol name
Artifact:		Name of the project file

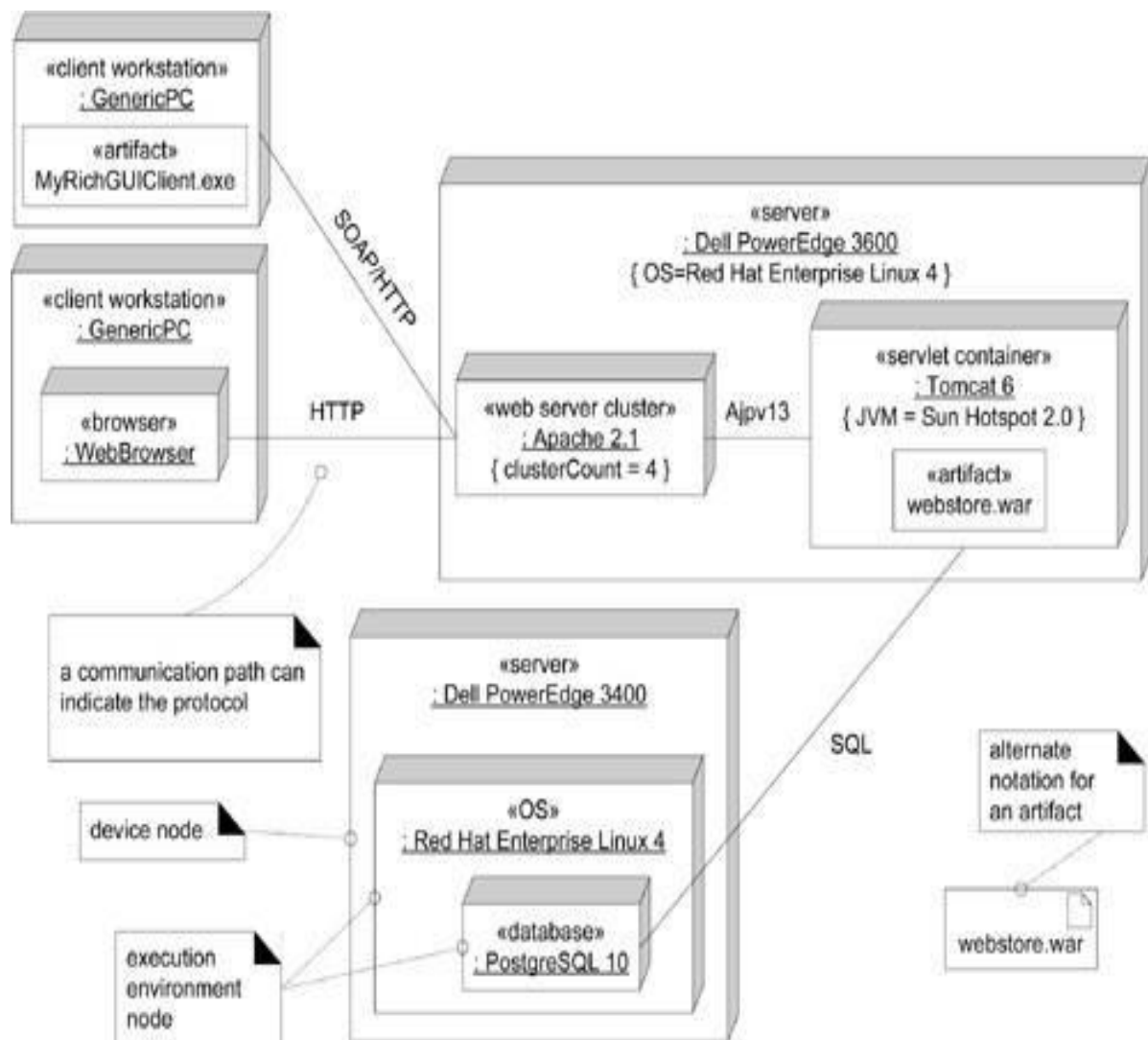
- As the UML specification suggests, many node types may show stereotypes, such as «server», «OS», «database», or «browser», but these are not official predefined UML stereotypes.
- Note that a device node or EEN may contain another EEN. For example, a virtual machine within an OS within a computer.
- A particular EEN can be implied, or not shown, or indicated informally with a UML property string; for example, { OS=Linux }.
- The normal connection between nodes is a communication path, which may be labeled with the protocol. These usually indicate the network connections.

- A node may contain and show an artifact a concrete physical element, usually a file. This includes executables such as JARs, assemblies, .exe files, and scripts. It also includes data files such as XML, HTML, and so forth.

Deployment Diagram Ex: Next Generation POS System

In the diagram ,there are 2 servers namely DellpowerEdge 3600 with RedHat Linux OS , Tomcat 6, Apache 2.1 are software computing resources , in tomcat server webstore.war file is loaded. In the other server DellpowerEdge 3400 with RedHat Linux OS, database PostgreSQL 10 are software computing resources.

There are 2 client nodes connected to server via HTTP & SOAP /HTTP protocol. In first client exe file is shown as artifact. In the other client , web base application is enabled by browser .



COMPONENT DIAGRAMS

A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. A component serves as a type, whose conformance is defined by these provided and required interfaces.

Features of UML component

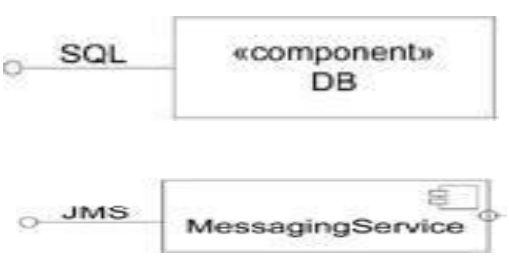
- 1) It has interfaces
- 2) it is modular, self-contained and replaceable.

The second point implies that a component tends to have little or no dependency on other external elements . it is a relatively stand-alone module.

Example : A good analogy for software component modeling is a home entertainment system; we expect to be able to easily replace the DVD player or speakers. They are modular, self-contained, replaceable, and work via standard interfaces.

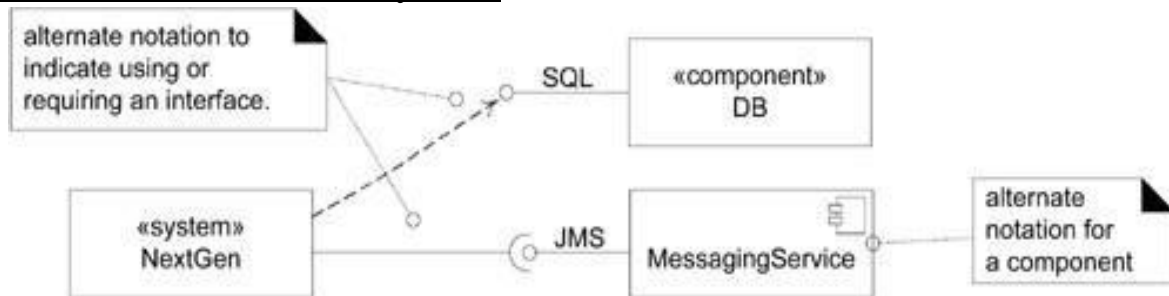
For example, at a large-grained level, a SQL database engine can be modeled as a component; any database that understands the same version of SQL and supports the same transaction semantics can be substituted. At a finer level, any solution that implements the standard Java Message Service API can be used or replaced in a system.

Elements:

Name	Symbol	Description
Component with Provided Interface		stand-alone module.
Dependency		System getting services from the component

Guideline : Component-based modeling is suitable for relatively large-scale elements, because it is difficult to think about or design for many small, fine-grained replaceable parts.

Ex: Next Generation POS system .



WHEN TO USE COMPONENT AND DEPLOYMENT DIAGRAMS

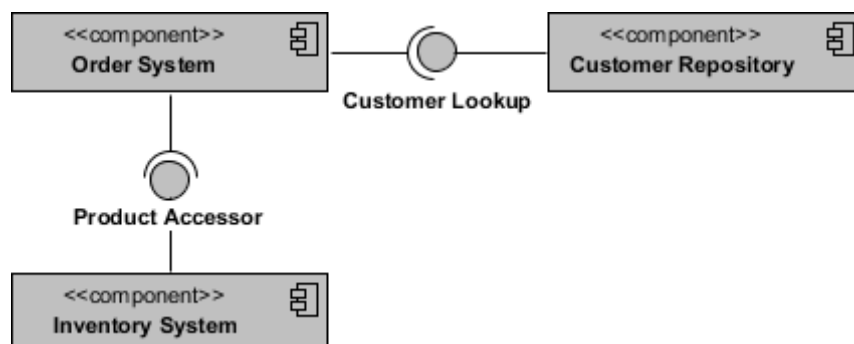
Component diagrams are used to visualize the static implementation view of a system. Component diagrams are special type of UML diagrams used for different purposes.

Component diagrams can be used to –

- Model the components of a system.
- Model the database schema.
- Model the executables of an application.
- Model the system's source code.

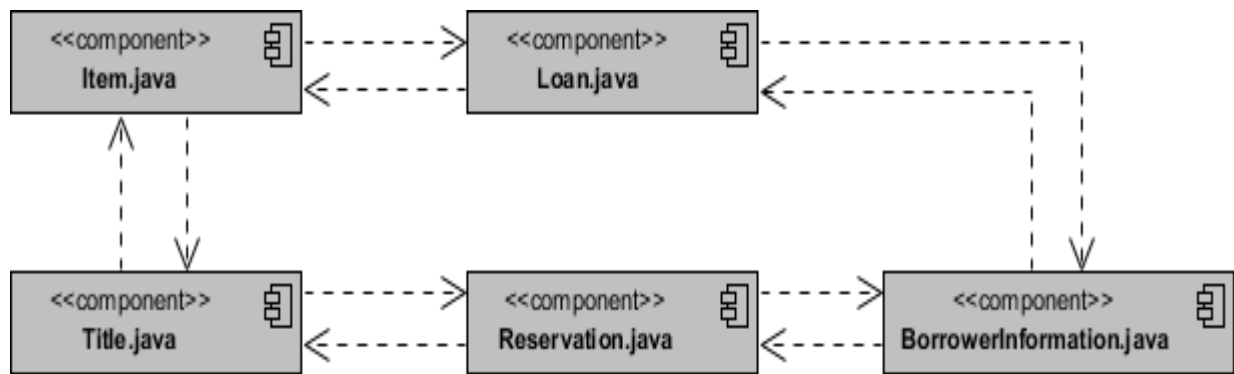
Model the components of a system

Ex: Order System



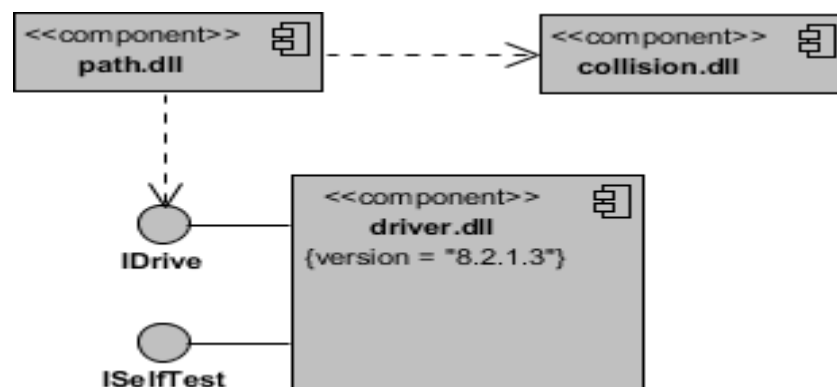
Modeling Source Code

- Either by forward or reverse engineering, identify the set of source code files of interest and model them as components stereotyped as files.
- For larger systems, use packages to show groups of source code files.
- Consider exposing a tagged value indicating such information as the version number of the source code file, its author, and the date it was last changed. Use tools to manage the value of this tag.
- Model the compilation dependencies among these files using dependencies. Again, use tools to help generate and manage these dependencies.



Modeling an Executable Release

- Identify the set of components to model.
- Consider the stereotype of each component in this set. find a small number of different kinds of components (such as executables, libraries, tables, files, and documents).
- For each component in this set, consider its relationship to its neighbors. It shows only dependencies among the comp



Modeling a Physical Database

- Identify the classes in the model that represent the logical database schema.
- Select a strategy for mapping these classes to tables.
- To visualize, specify, construct, and document your mapping, create a component diagram that contains components stereotyped as tables.
- Where possible, use tools to help you transform your logical design into a physical design.

