UNIT - IV

ADVANCED BEHAVIOURAL MODELING

Architectural Modeling:

Components:

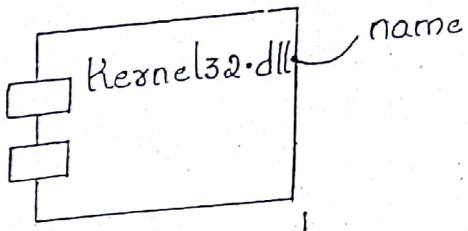


Figure : Components

Terms and Concepts:

- A component is a physical and replacable part of a system that conforms to and provides the realization of a set of interfaces.
- Graphically, a component is rendered as a rectangle with tabs.

Names:

- Every component must have a name that distinguishes it from oth components.

That name alone is known as a simple name.
A pathname is the component name prefixed by the name of the package in which that component lives.

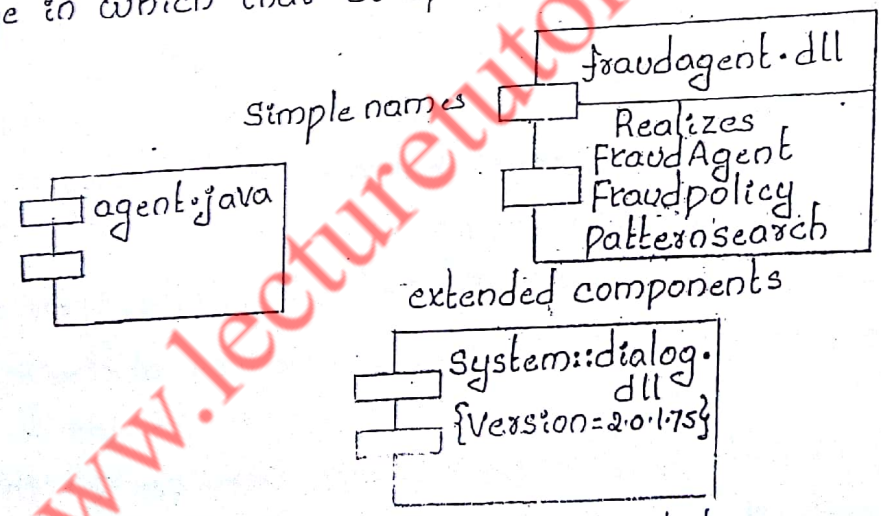


Figure : simple and Extended components

Components and classes:

In many ways, components are like classes : Both have names; both may realize a set of interfaces; both may participate in dependency generalization and association relationships; both may be nested; bo may have instances; both may be participants in interactions.

wever, there are some significant differences between components and classes.

Classes	Components
) classes represent logical bstractions.	(1) In short, components may live on nodes, classes may not.
) classes may have attributes nd operations directly.	(2) In general, components only have operations that are reachable only through their interfaces.

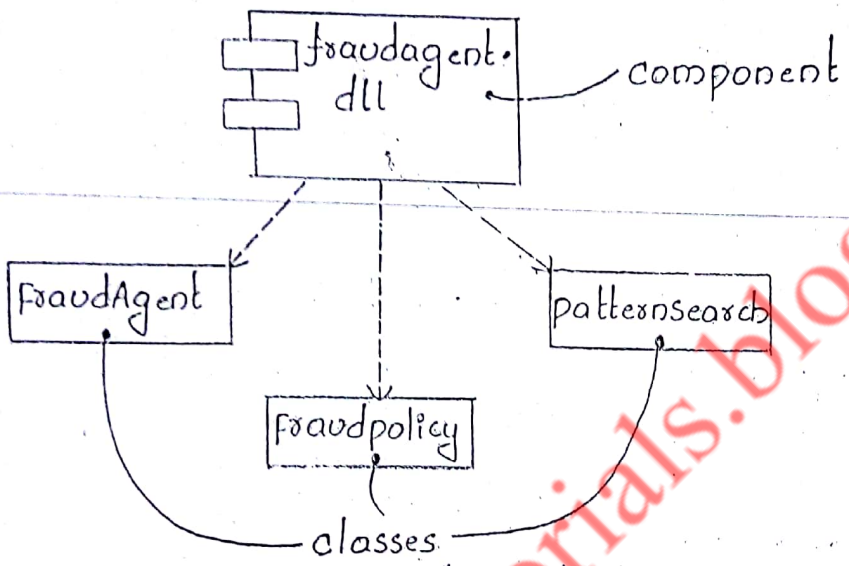


Figure : components and classes

Components and Interfaces:

n interface is a collection of operations that are used to specify a ervice of a class or a component.

he relationship between component and interface is important. he component that realizes the interface is connected to the interfa sing a full realization relationship.

n both cases, the component that accesses the services of the ther component through the interface is connected to the interfa sing a dependency relationship.

n interface that a component realizes is called an export interfac t means ~~an interface~~ an interface that the component provides a service to other components. A component may provide many xport interfaces.

he interface that a component uses is called an import interface.

it means an interface that the component conforms to and so builds on.

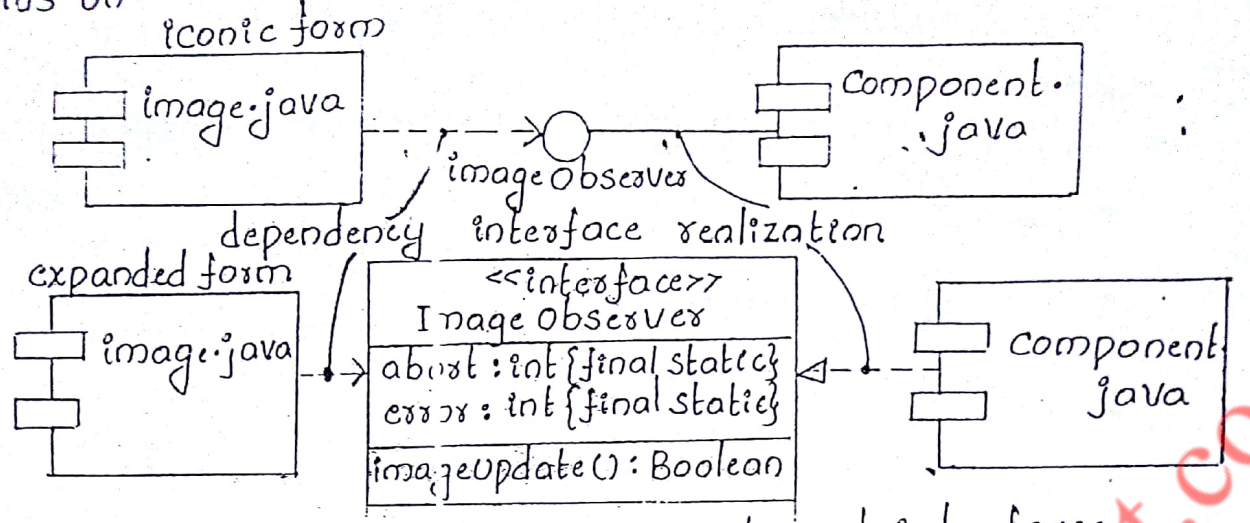


Figure: components and interfaces

Binary Replaceability:

- First, a component is physical. It lives in the world of bits, not concepts.
- Second, a component is replaceable.

Third, a component is part of a system.

Fourth, a component conforms to and provides the realization of a set of interfaces.

Kinds of Components:

Three kinds of components may be distinguised.

First, there are deployment components.

These are the components necessary and sufficient to form an executable system, such as dynamic libraries (DLLs) and executables (EXEs).

Second, there are work product components.

These components are essentially the residue of the development process, consisting of things such as source code files and data files from which deployment components are created.

Third are execution components.

These components are created as a consequence of an executing system, such as a COM object, which is instantiated from a DLL.

ndard Elements:

e UML defines five standard stereotypes that apply to component

executable → specifies a component that may be executed on a node.

library → specifies a static or dynamic object library.

table → specifies a component that represents a database table.

file → specifies a component that represents a document containing source code or data.

document → specifies a component that represents a document

eployment:

node is a physical element that exists at runtime and represent computational resource, generally having atleast some memory nd often processing capability.

raphically, a node is rendered as a cube.

lames:

very node must have a name that distinguishes it from other nodes. name is a textual string.

at name alone is known as a simple name; a path name is the ode name prefixed by the name of the package in which that node ves.

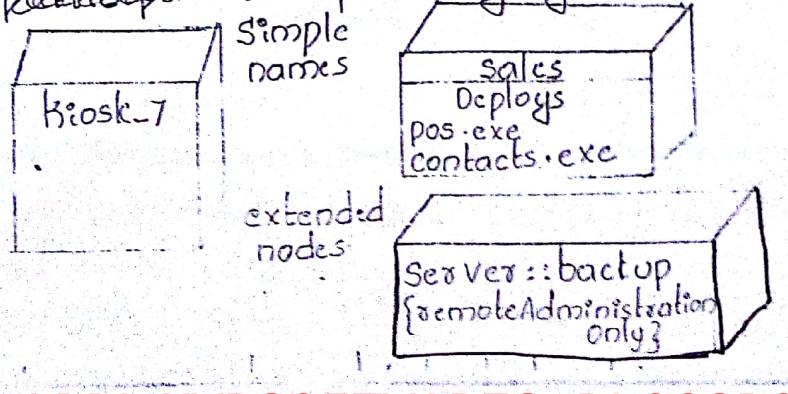
node is typically drawn showing only its name.

odes and Components:

n many ways, nodes are a lot like components:

oth have names.

oth may participate in dependency, generalization and association



odes and components:

In many ways, nodes are a lot like components.
Both have names.
Both may participate in dependency, generalization and associatio relationships; both may be nested; both may have instances; both may be participants in interactions.
However, there are some significant differences between nodes and components.

components	nodes
(1) components are things that participate in the execution of a system.	(1) nodes are things that execute components
(2) components represent the physical packaging of otherwise logical elements.	(2) nodes represent the physical deployment of components.

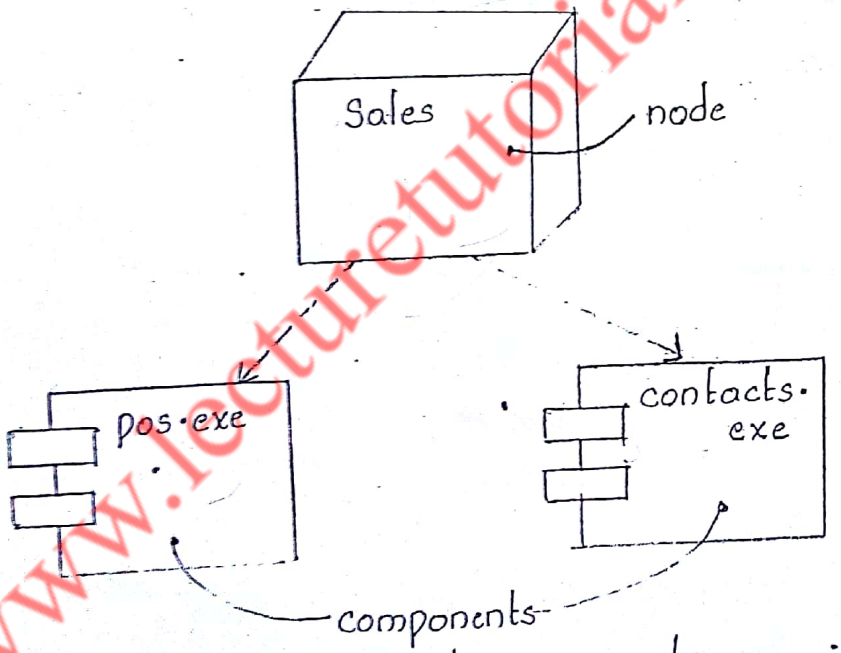


Figure : Nodes and components

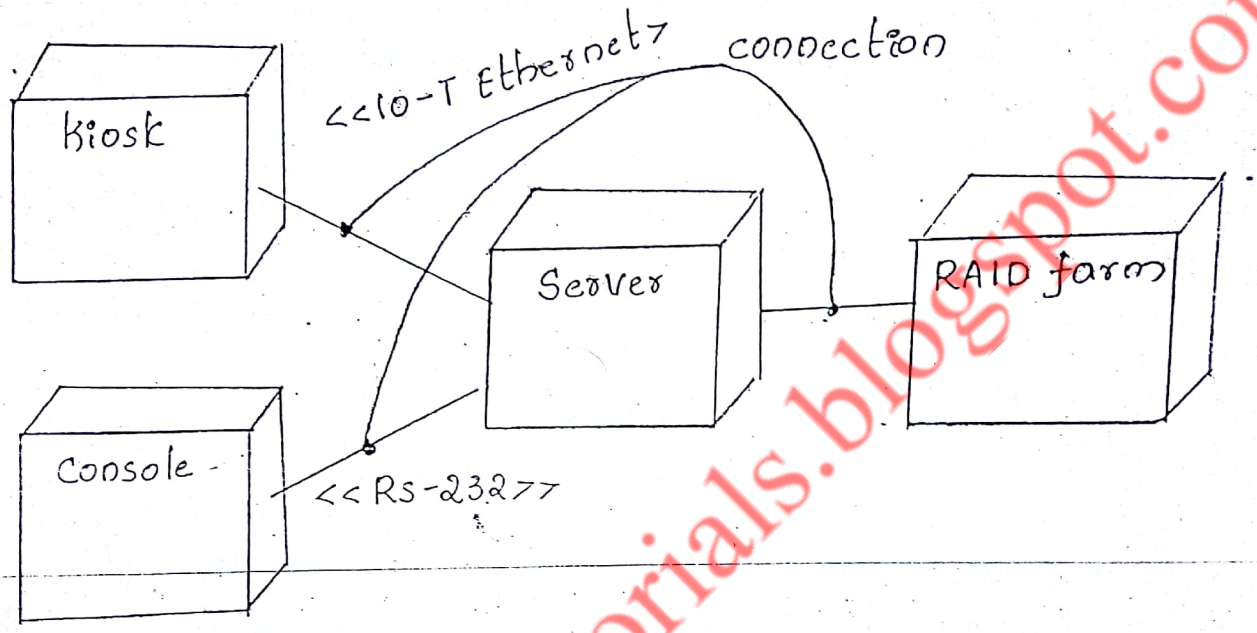
Organizing Nodes:

You can organize nodes by grouping them in packages in the same manner in which you can organize classes and components.

connections:

The most command kind of relationship you will use among

des is an association.
ou can even use associations to model indirect connections,
ch as a satellite link between distant processors.



Uses:
To model processors and devices.
To model the distributions of components.
To model Embedded systems.

ploym Component Diagrams:

rms and Concepts:

A component diagram shows a set of components and their relationsl Graphically, a component diagram is a collection of vertices and arc: A component diagram is just a special kind of diagram and shares th same common properties as do all other diagrams - a name and graphical contents that are a projection into a model.

Contents:

Component diagrams commonly contain

- ✓ components
- ✓ Interfaces
- ✓ Dependency, generalization, association and realization relationships.

Like all other diagrams, component diagrams may contain notes an constraints.

Component diagrams may also contain packages or subsystems, both of which are used to group elements of your model into larger chunks

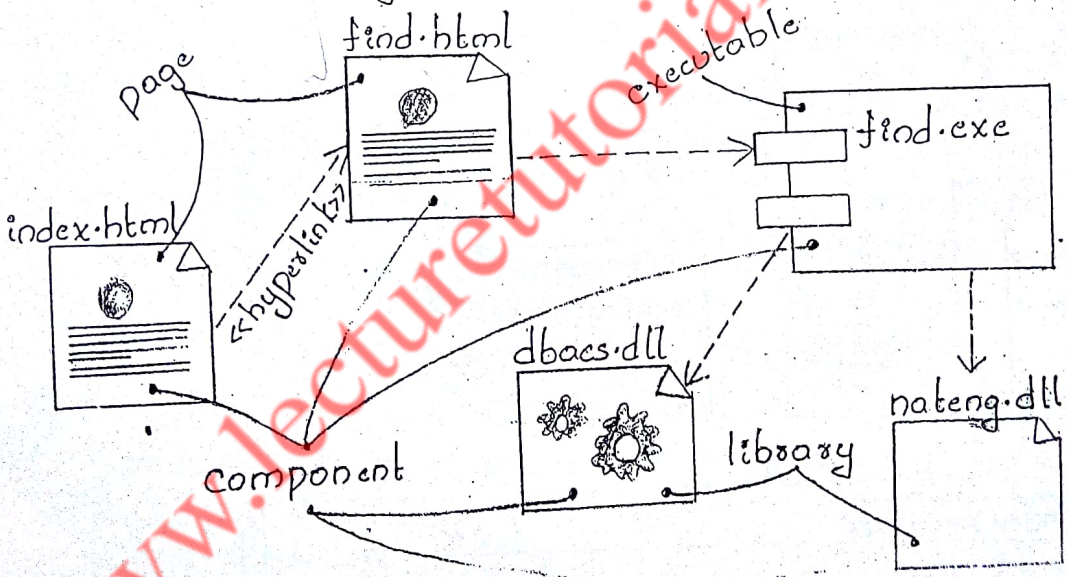


Figure: A component Diagram

Common uses:

- To model source code.
- To model executable releases.
- To model physical databases.
- To model adaptable systems.

loyment Diagrams:

ms and concepts:

leployment diagram is a diagram that shows the configuration of
time processing nodes and the components that live on them.
aphically, a deployment diagram is a collection of vertices and arcs.
deployment diagram is just a special kind of diagram and shares
same common properties as all other diagrams – a name and
phical contents that are a projection into a model.

ntents:

ployment diagrams commonly contain

- ✓ Nodes
- ✓ Dependency and association relationships

ke all other diagrams, deployment diagrams may contain notes
d constraints.

ployment diagrams may also contain components, each of which
ist live on some node.

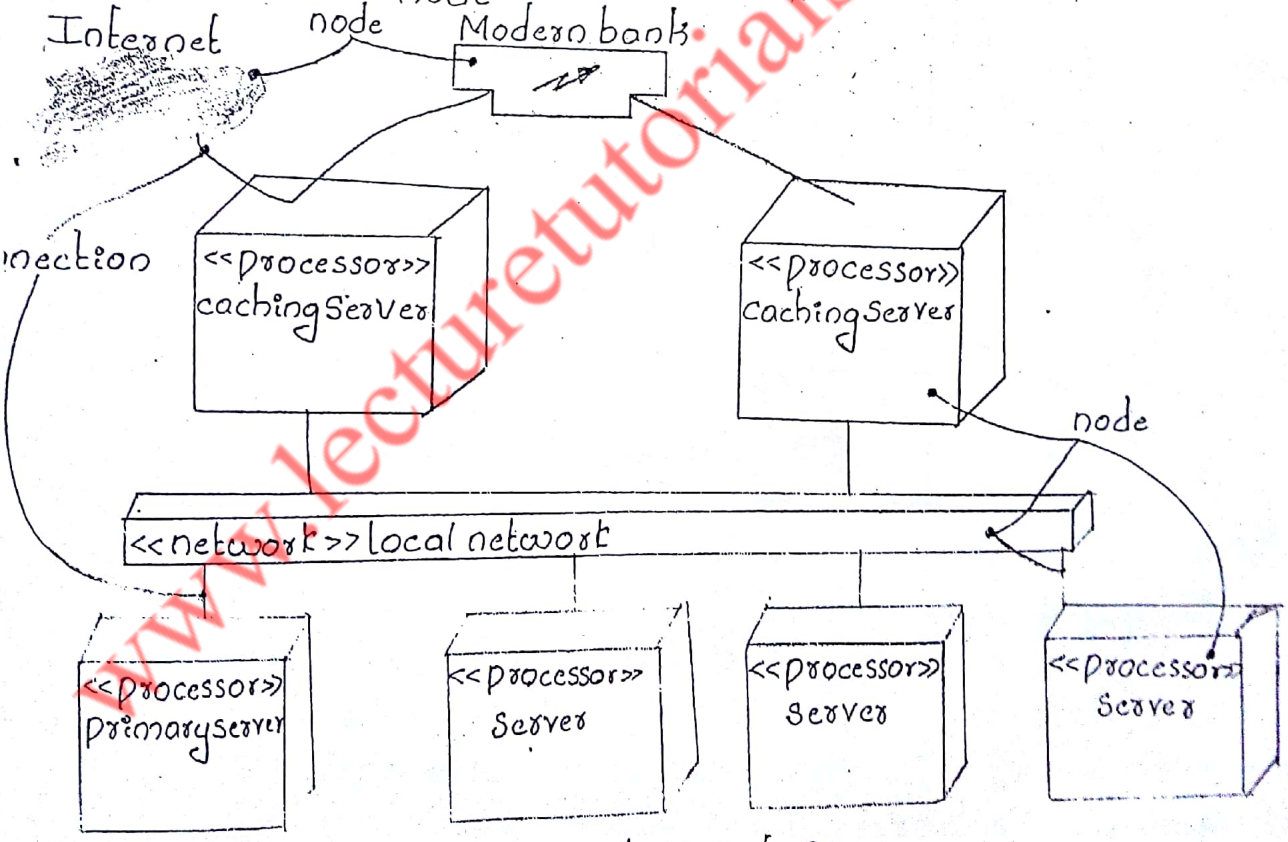


Figure: A Deployment Diagram

Common uses:

To model embedded systems.

To model client/server systems.

Common Modeling Techniques for component Diagrams:

Modeling Source Code:

To model a systems 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 depend encies. Again, use tools to help generate and manage these dependenci

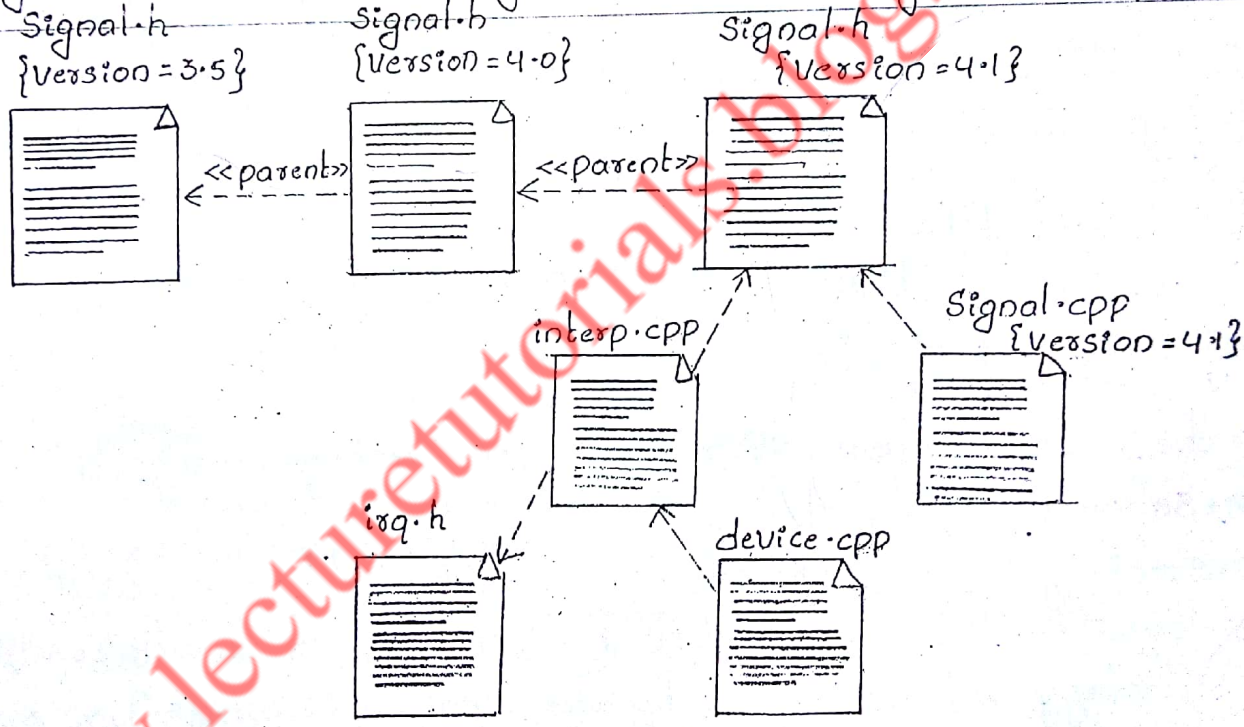


Figure: Modeling source code

Modeling an Executable Release:

To model an executable release,

Identify the set of components you did like to model. Typically, this will involve some or all the components that live on one node, or the distribution of these sets of components across all the nodes in the system.

Consider the stereotype of each component in this set. For most systems, you will find a small number of different kinds of comp

can use the UML's extensibility mechanisms to provide visual
s for these stereotypes.

each component in this set, consider its relationship to its neigh-
s. Most often, this will involve interfaces that are exported (realized
certain components and then imported (used) by others. If you won't
expose the seams in your system, model these interfaces explicitly.
you want your model at a higher level of abstraction, elide these
tionships by showing only dependencies among the components.

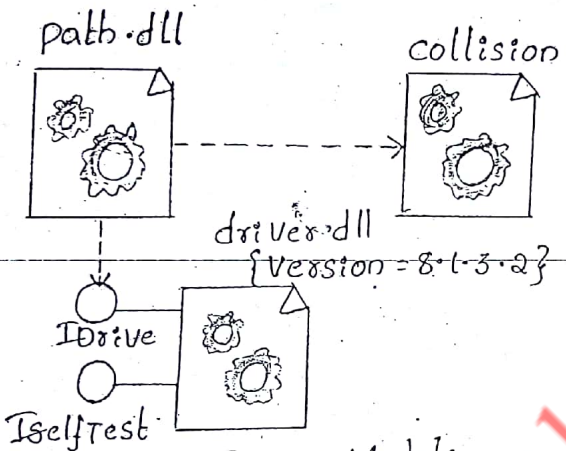


Figure: Modeling an Executable Release

Modeling a physical Database:

dentify the classes in your model that represent your logical
tabase schema.

lect a strategy for mapping these classes to tables. you will also
ant to consider the physical distribution of your databases. your
apping strategy will be affected by the location in which you want
ur data to live on your deployed system.

o visualize, specify, construct and document your mapping, create
component diagram that contains components stereotyped as table
here possible, use tools to help you transform your logical design
to a physical design.

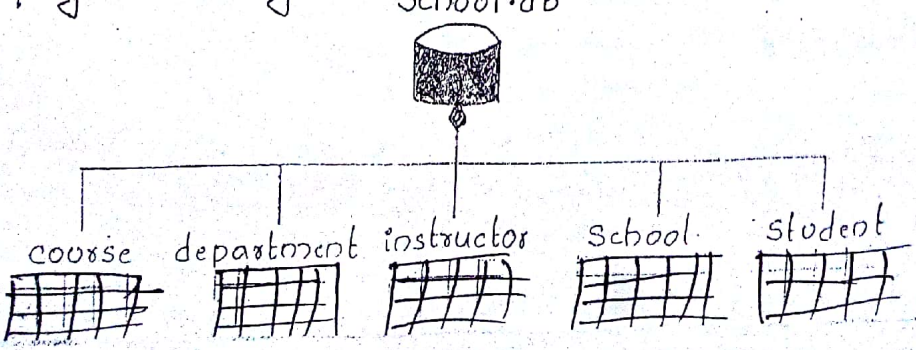


Figure: Modeling a physical Database

Modeling Adaptable Systems:

To model an Adaptable system,

consider the physical distribution of the components that may migrate from node to node. You can specify the location of a compone instance by marking it with a location tagged value, which you can then render in a component diagram (although, technically speaking, a diagram that contains only instances is an object diagra

If you want to model the actions that cause a component to migrate create a corresponding interaction diagram that contains component instances. You can illustrate a change of location by drawing the same instance more than once, but with different values for its location tagged value.

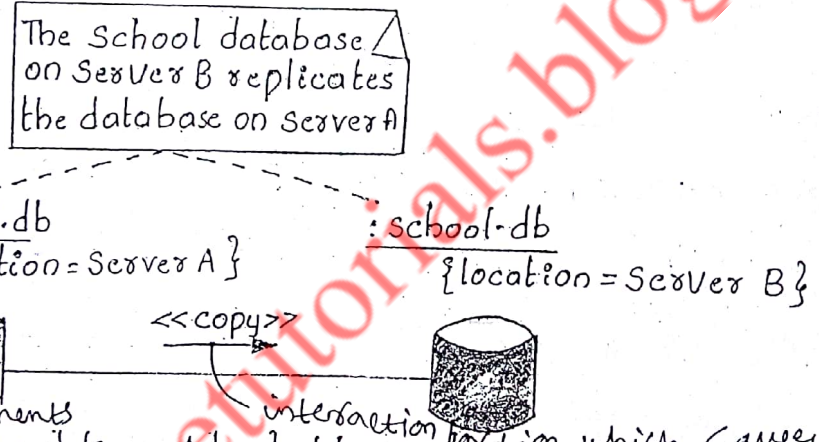


Figure: Modeling Adaptable Systems

tion which causes this migrati

Common modeling Techniques for Deployment Diagrams:

Modeling an Embedded system:

To model an embedded system,

Identify the devices and nodes that are unique to your system.

Provide visual cues, especially for unusual devices, by using the UML extensibility mechanisms to define system-specific stereotypes with appropriate icons. At the very last you will want to distinguish proces (which contain software components) and devices (which, at that leve of abstraction, don't directly contain software).

Model the relationships among these processors and devices in a Deployment diagram. Similarly, specify the relationship between the components in your system imple-

necessary, expand on any intelligent devices by modeling their
ucture with a more detailed deployment diagram.

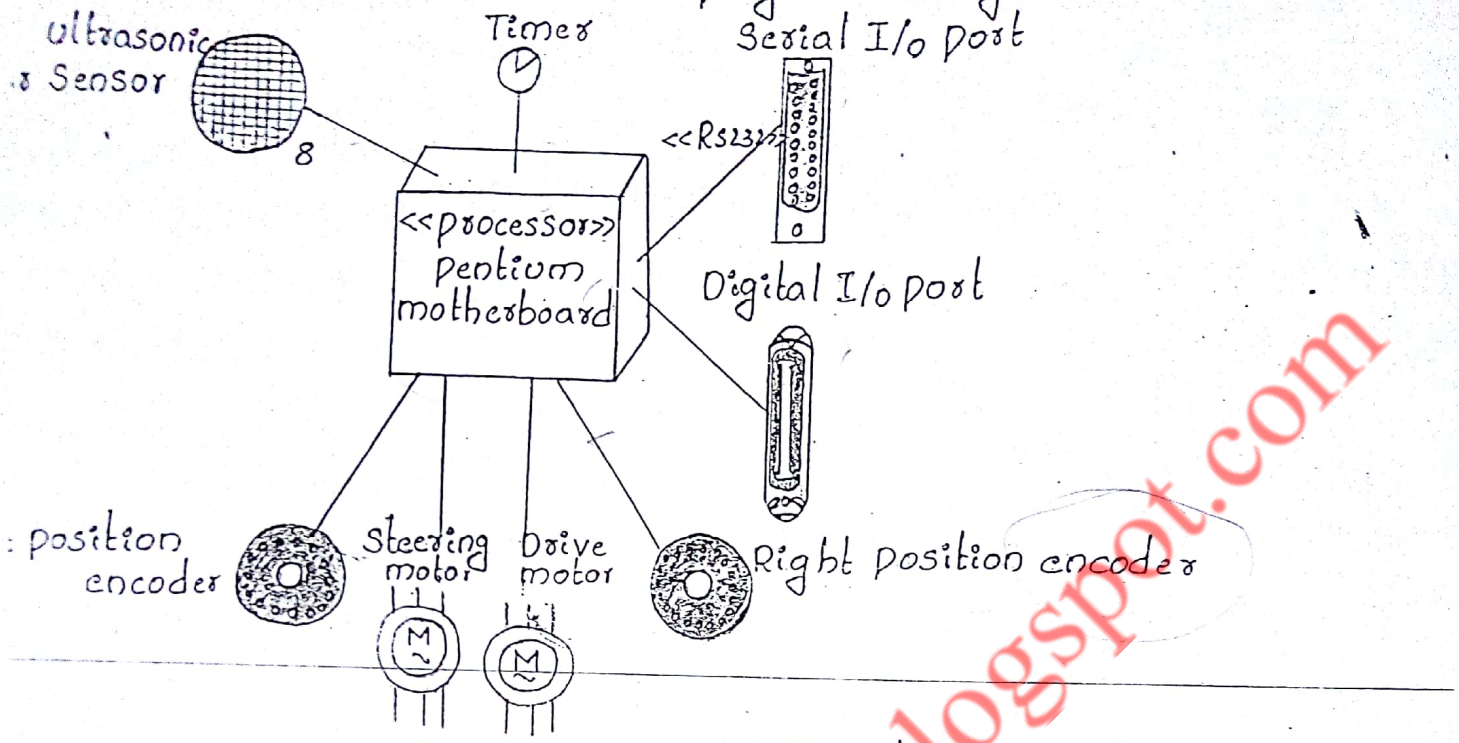


Figure: Modeling an Embedded system

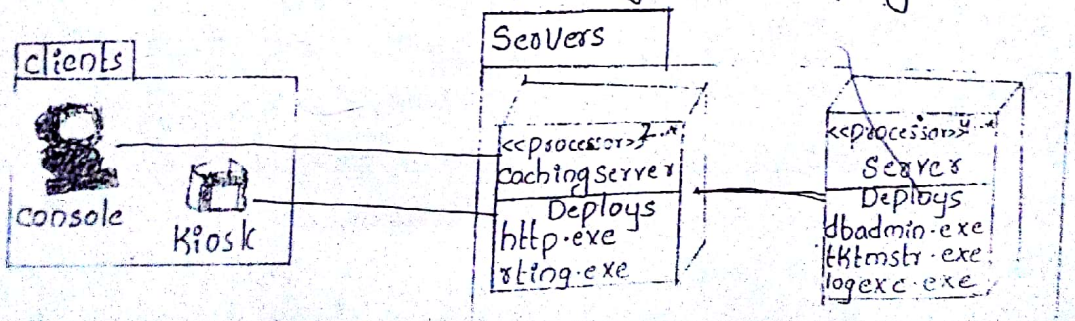
lodeling a client/server system:

o model a client/server system,

dentify the nodes that represent your system's client and server
rocessors.

lighlight those devices that are germane to the behavior of your
ystem. for example, you will want to model special devices, such a
edit card readers, badge readers and display devices other than
onitors, because their placement in the systems hardware topolog
ie likely to be architecturally significant.

rovide visual cues for these processors and devices via stereotypi
lodel the topology of these nodes in a deployment diagram. similarly,
pecify the relationship between the components in your systems imple
tation view and the nodes in your systems deployment view.



Modeling a Fully Distributed System:

To model a fully distributed system,

Identify the systems devices and processors as for simpler client/server systems.

If you need to reason about the performance of the systems network or the impact of changes to the network, be sure to model these communication devices to the level of detail sufficient to make these assessments.

Pay close attention to logical groupings of nodes, which you can specify by using packages.

Model these devices and processors using deployment diagrams.

where possible, use tools that discover the topology of your system by walking your system's network.

If you need to focus on the dynamics of your system, introduce usecase diagrams to specify the kinds of behavior you are interested in, and expand on these usecases with interaction diagrams.

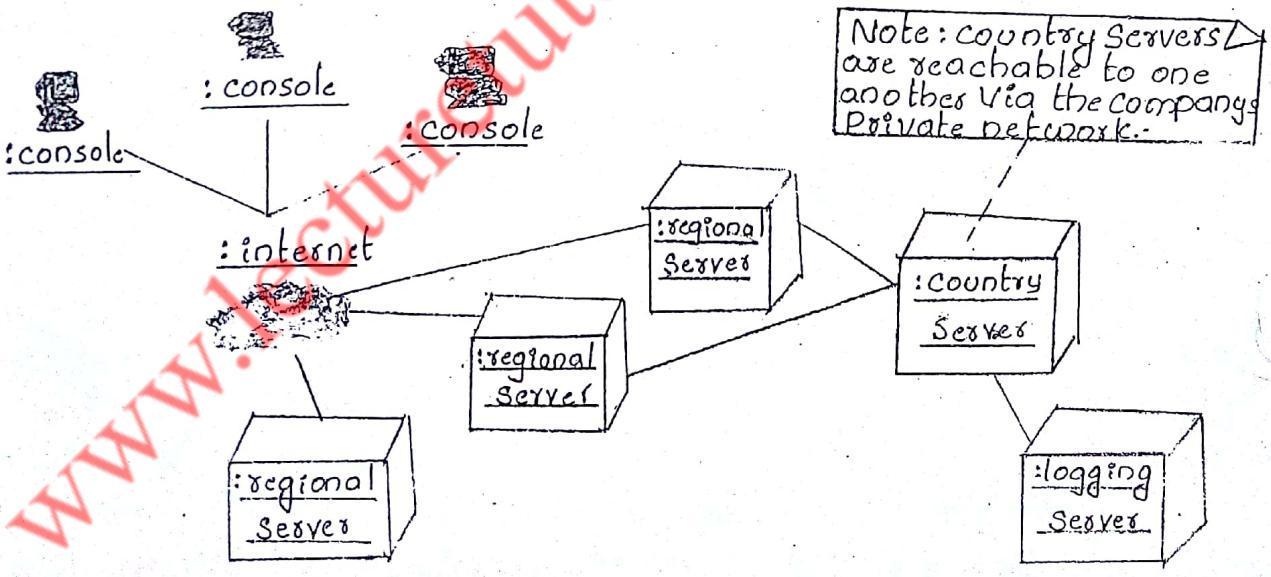


Figure: Modeling a Fully Distributed System