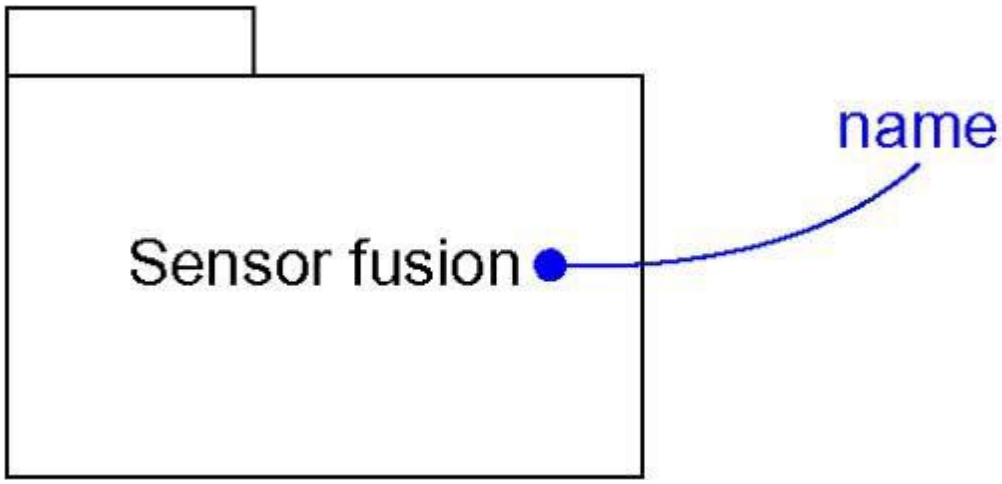
UNIT-IV

STRUCTURAL MODELING

Package Diagram

In the UML, the chunks that organize a model are called packages. A package is a general purpose mechanism for organizing elements into groups. Packages help you organize the elements in your models so that you can more easily understand them. Packages also let you control access to their contents so that you can control the seams in your system's architecture. The UML provides a graphical representation of package

This notation permits you to visualize groups of elements that can be manipulated as a whole and in a way that lets you control the visibility of and access to individual elements.



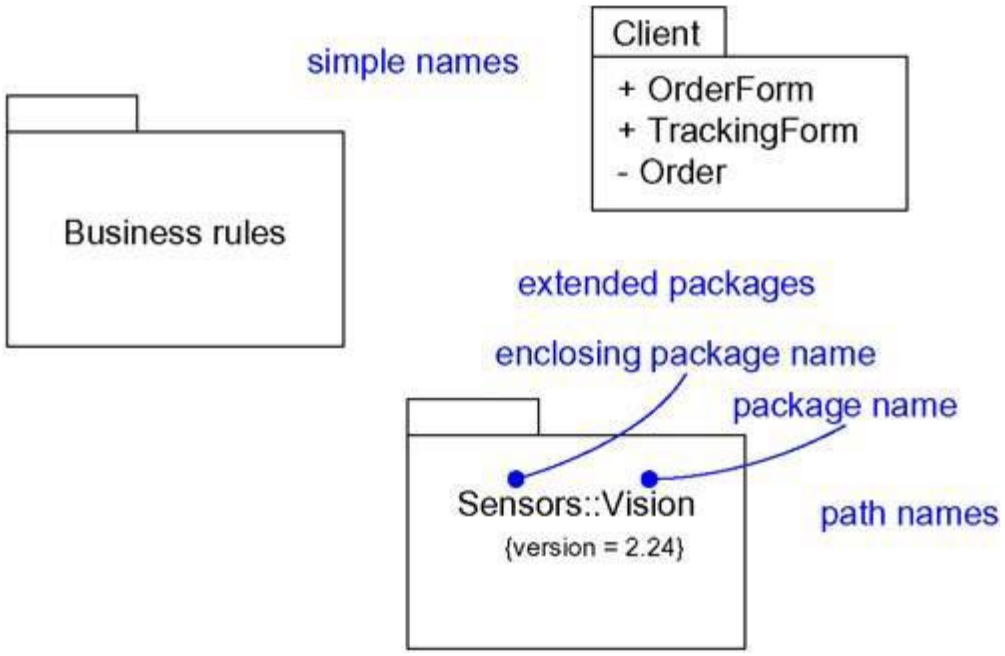
Terms and Concepts

A *package* is a general-purpose mechanism for organizing elements into groups. Graphically, a package is rendered as a tabbed folder.

Names

A package name must be unique within its enclosing package.

Every package must have a name that distinguishes it from other packages. A *name* is a textual string. That name alone is known as a *simple name;* a *path name* is the package name prefixed by the name of the package in which that package lives, if any

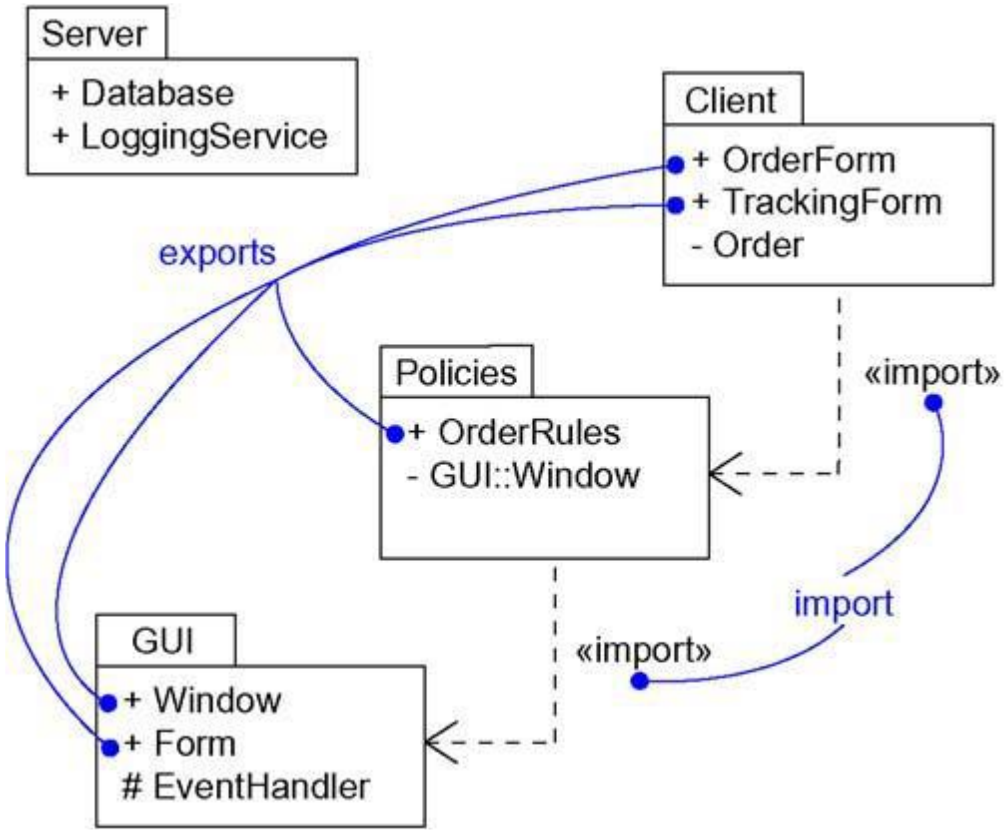


Importing and Exporting

Suppose you have two classes named A and B sitting side by side. Because they are peers, A can see B and B can see A, so both can depend on the other. Just two classes makes for a trivial system, so you really don't need any kind of packaging. Now, imagine having a few hundred such classes sitting side by side. There's no limit to the tangled web of relationships that you can weave. Furthermore, there's no way that you can understand such a large, unorganized group of classes. That's a very real problem for large systems• simple, unrestrained access does not scale up. For these situations, you need some kind of controlled packaging to organize your abstractions. So suppose that instead you put A in one package and B in another package, both packages sitting side by side. Suppose also that A and B are both declared as public parts of their respective packages. This is a very different situation. Although A and B are both public, neither can access the other because their enclosing packages form an opaque wall. However, if A's package imports B's package, A can now see B, although B cannot see A. Importing grants a one-way permission for the elements in one package to access the elements in another package.

In the UML, you model an import relationship as a dependency adorned with the stereotype import. By packaging your abstractions into meaningful chunks and then

controlling their access by importing, you can control the complexity of large numbers of abstractions



Component Diagrams

Terms and Concepts

A *component diagram* shows a set of components and their relationships. Graphically, a component diagram is a collection of vertices and arcs.

Common Properties

A component diagram is just a special kind of diagram and shares the same common properties as do all other diagrams• a name and graphical contents that are a projection into a model. What distinguishes a component diagram from all other kinds of diagrams is its particular content.

Contents

Component diagrams commonly contain

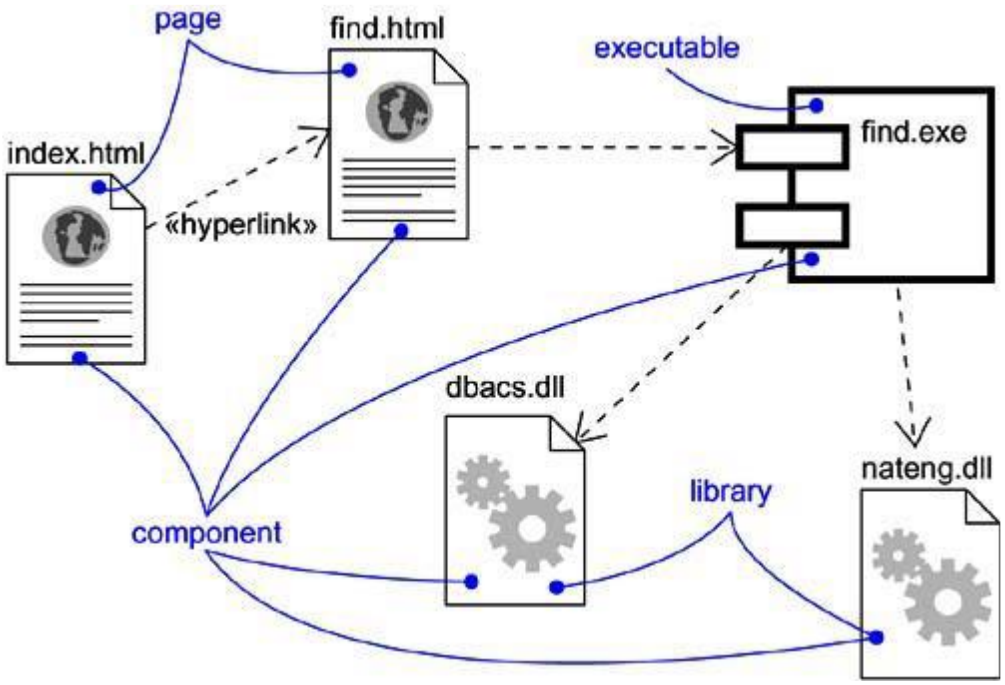
· Components

· Interfaces

· Dependency, generalization, association, and realization relationships

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

Component diagrams may also contain packages or subsystems, both of which are used to group elements of your model into larger chunks. Sometimes, you'll want to place instances in your component diagrams, as well, especially when you want to visualize one instance of a family of component-based systems.



Deployment Diagrams

Deployment diagrams are one of the two kinds of diagrams used in modeling the physical aspects of an object-oriented system. A deployment diagram shows the configuration of run time processing nodes and the components that live on them.

Terms and Concepts

A *deployment diagram* is a diagram that shows the configuration of run time processing nodes and the components that live on them. Graphically, a deployment diagram is a collection of vertices and arcs.

Common Properties

A deployment diagram is just a special kind of diagram and shares the same common properties as all other diagrams• a name and graphical contents that are a projection into a model. What distinguishes a deployment diagram from all other kinds of diagrams is its particular content.

Contents

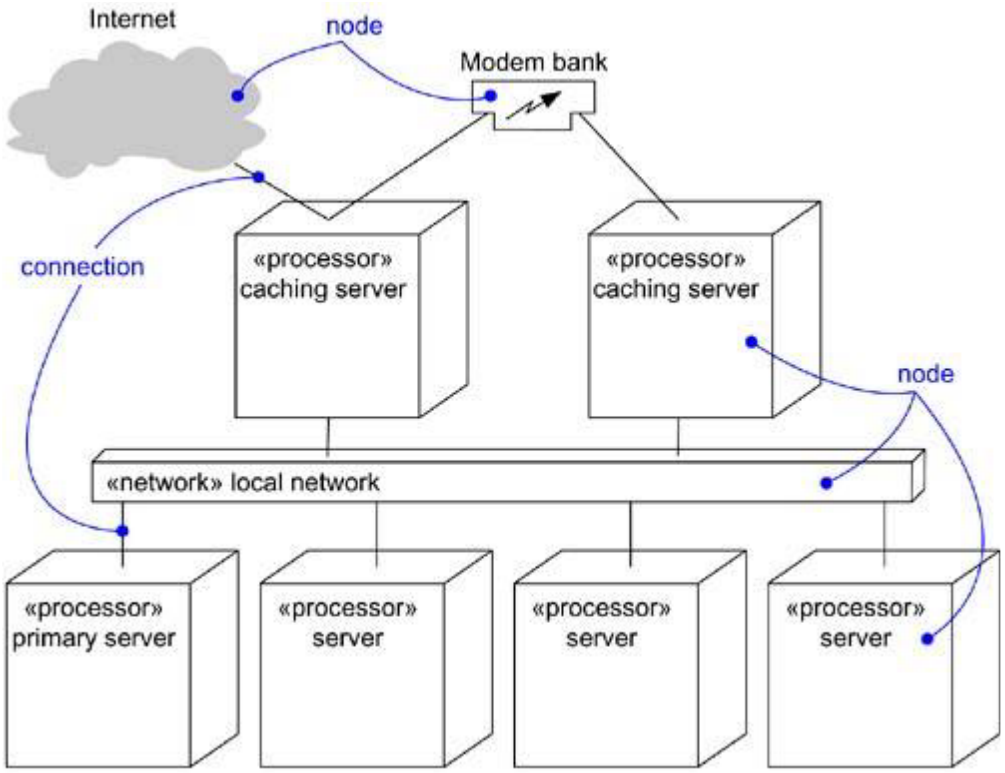
Deployment diagrams commonly contain

· Nodes

· Dependency and association relationships

Like all other diagrams, deployment diagrams may contain notes and constraints.

Deployment diagrams may also contain components, each of which must live on some node. Deployment diagrams may also contain packages or subsystems, both of which are used to group elements of your model into larger chunks. Sometimes, you'll want to place instances in your deployment diagrams, as well, especially when you want to visualize one instance of a family of hardware topologies.



Common Modeling Techniques

Modeling an Embedded System

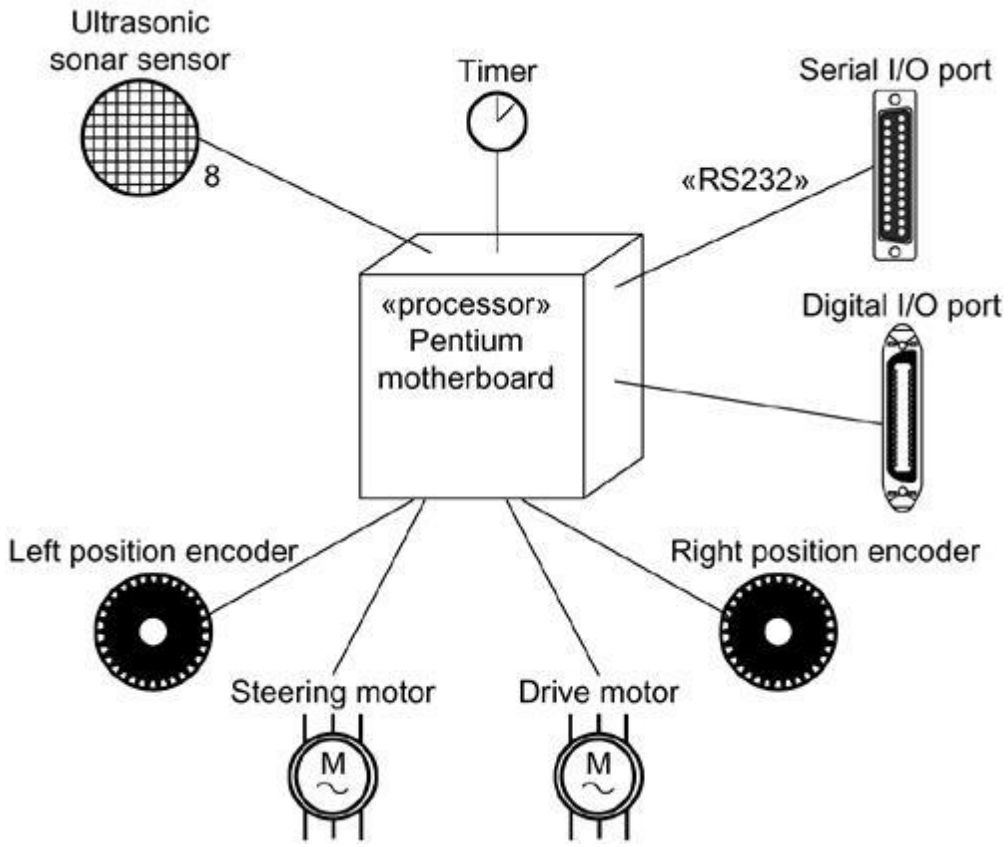
To model an embedded system,

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

· Provide visual cues, especially for unusual devices, by using the UML's extensibility mechanisms to define system-specific stereotypes with appropriate icons. At the very least, you'll want to distinguish processors (which contain software components) and devices (which, at that level 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's implementation view and the nodes in your system's deployment view.

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



Modeling a Client/Server System

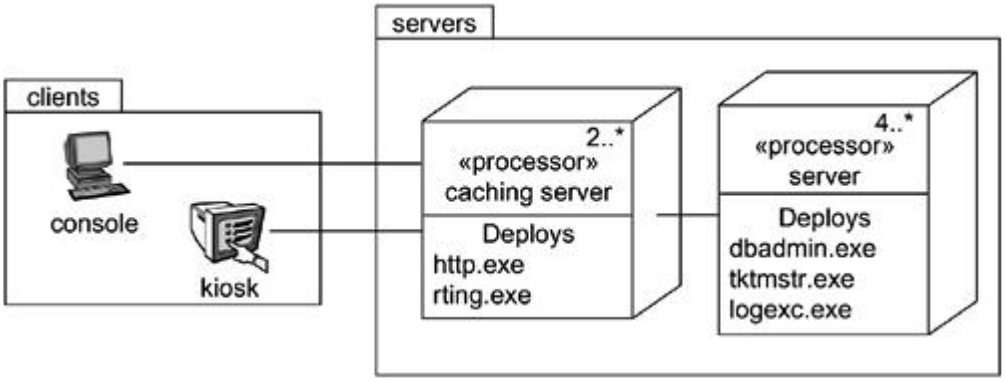
To model a client/server system,

· Identify the nodes that represent your system's client and server processors.

· Highlight those devices that are germane to the behavior of your system. For example, you'll want to model special devices, such as credit card readers, badge readers, and display devices other than monitors, because their placement in the system's hardware topology are likely to be architecturally significant.

· Provide visual cues for these processors and devices via stereotyping.

· Model the topology of these nodes in a deployment diagram. Similarly, specify the relationship between the components in your system's implementation view and the nodes in your system's deployment view.



Modeling a Fully Distributed System

To model a fully distributed system,

· Identify the system's devices and processors as for simpler client/server systems.

· If you need to reason about the performance of the system's 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 use case diagrams to specify the kinds of behavior you are interested in, and expand on these use cases with interaction diagrams.