Implementation Diagrams

Component Diagram

A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.

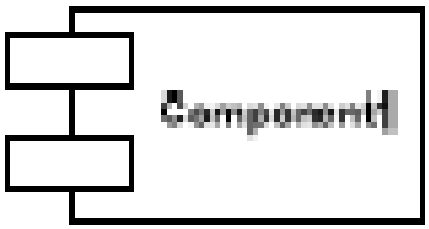
The purpose of the Component diagram is to define software modules and their relationships to one another. Each component is a chunk of code that resides in memory on a piece of
hardware.

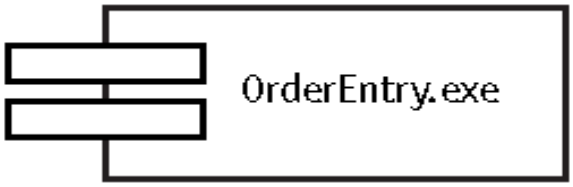
_ The UML groups components into three broad categories:

- Deployment components, which are required to run the system
- Work product components including models, source code, and data files used to create deployment components
- Execution components, which are components created while running the application

Notations

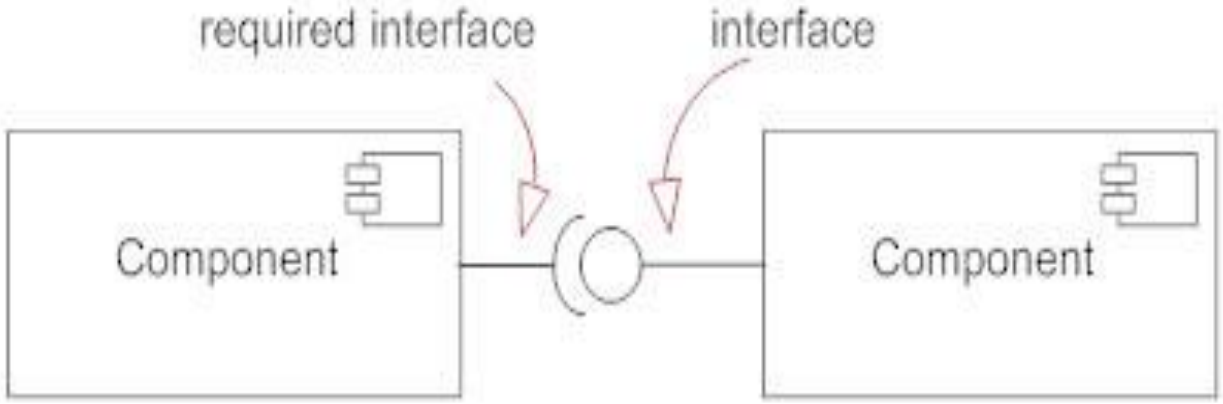
The component icon is a rectangle with two small rectangles on the left edge.

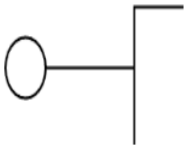




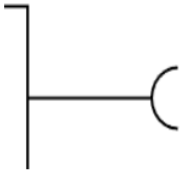
Interface

Interfaces in component diagrams show how components are wired together and interact with each other. The assembly connector allows linking the component's required interface (represented with a semi-circle and a solid line) with the provided interface (represented with a circle and solid line) of another component. This shows that one component is providing the service that the other is requiring.





Provided interfaces: A straight line from the component box with an attached circle. These symbols represent the interfaces where a component produces information used by the required interface of another component.

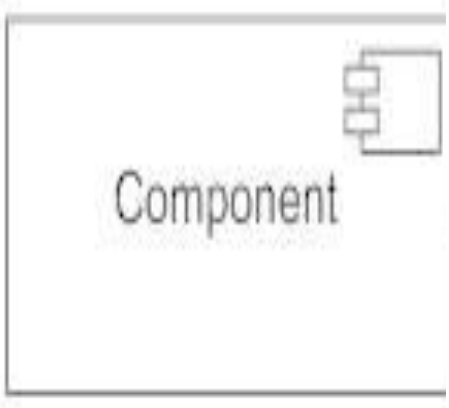


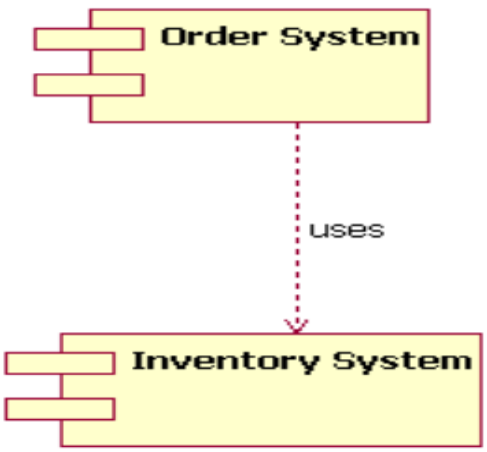
Required interfaces: A straight line from the component box with an attached half circle (also represented as a dashed arrow with an open arrow). These symbols represent the interfaces where a component requires information in order to perform its proper function.

Dependencies

Draw dependencies among components using dashed arrows.

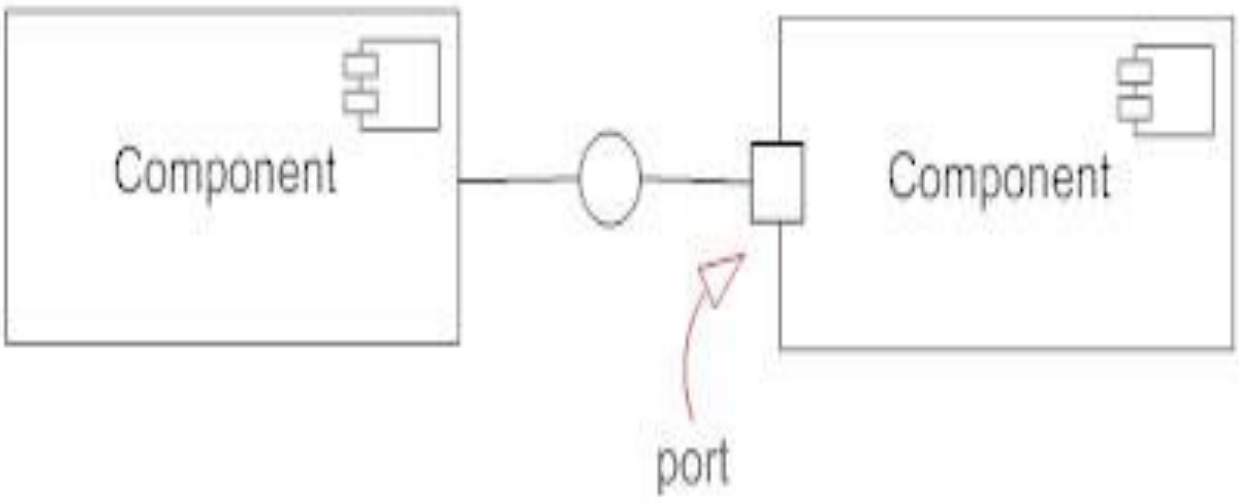
Although you can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface), you can just as well use a dependency arrow to show the relationship between two components.



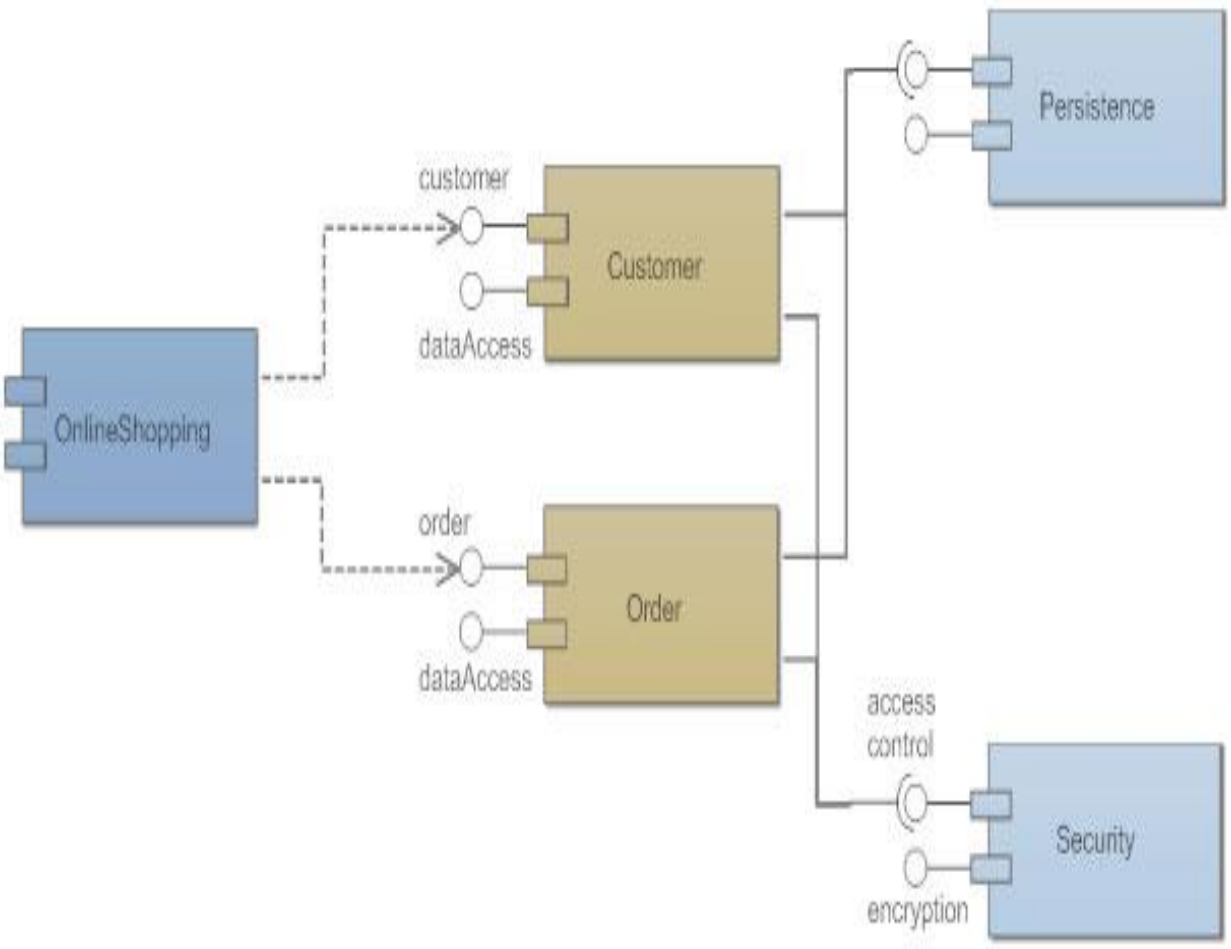


Port

Port (represented by the small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.



NB : Component Diagram does not describe the functionality of the system but it describes the components used to make those functionalities



Component stereotypes

Component stereotypes provide visual clues to the role that the component plays in the implementation. Some common component stereotypes include:

_ **<<executable>>**: A component that runs on a processor

_ **<<library>>**: A set of resources referenced by an executable during runtime

_ **<<table>>:** A database component accessed by an executable

_ **<<file>>**: Typically represents data or source code

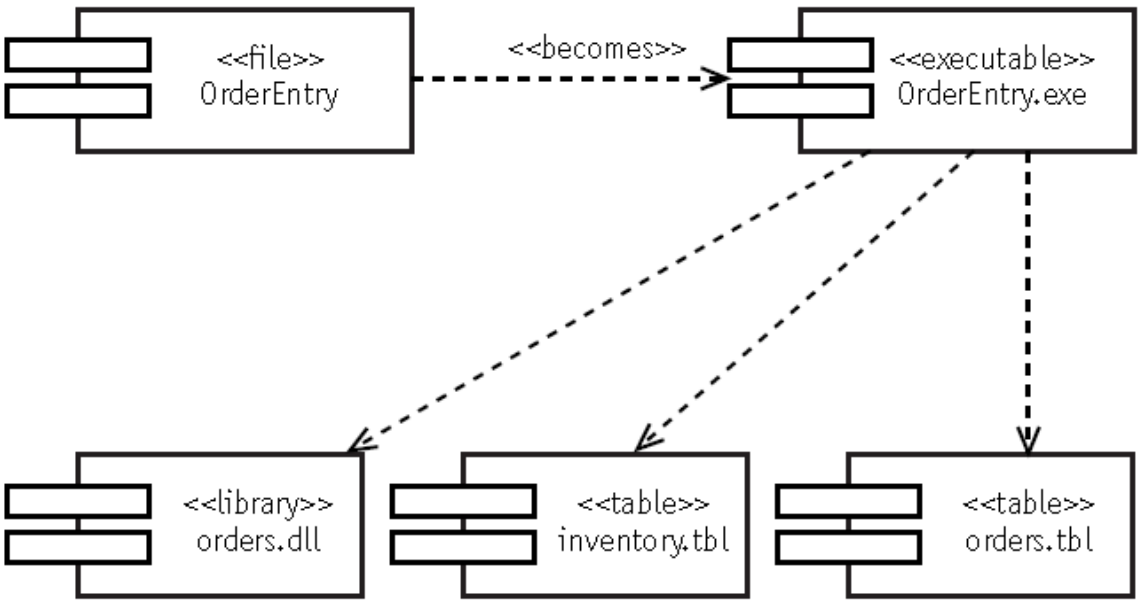
_ **<<document>>**: A document such as a page inserted into a Web page

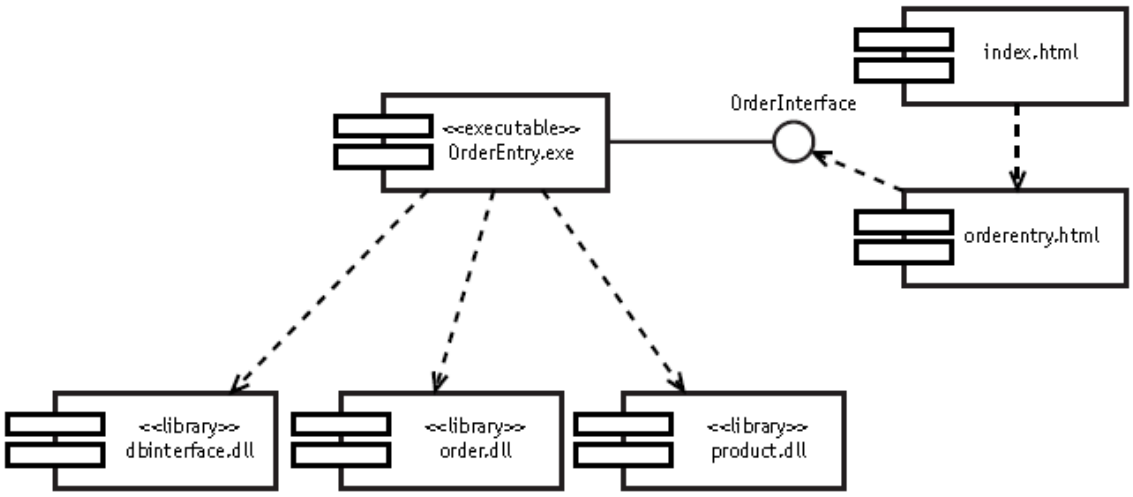
Component dependencies

Dependencies between components are drawn with the dashed arrow from the dependent component to the component it needs help from. In Figure below, the OrderEntry depends on the OrderEntry.exe component.

The UML stereotype <<becomes>> means that the OrderEntry file literally

becomes the OrderEntry executable at runtime. OrderEntry would be the code sitting on a storage device. At runtime it is loaded into memory and possibly even compiled. Then during execution the OrderEntry.exe component would depend on the three other components: orders.dll, inventory.dll, and orders.tbl.





How to Draw a Component Diagram

A component diagram in UML gives a bird's-eye view of your software system. Understanding the exact service behavior that each piece of your software provides will make you a better developer. Component diagrams can help software developer:

- Imagine the system's physical structure.
- Pay attention to the system's components and how they relate.
- Emphasize the service behavior as it relates to the interface.

You can use a component diagram when you want to represent your system as components and want to show their interrelationships through interfaces. It helps you get an idea of the implementation of the system. Following are the steps you can follow when drawing a component diagram.

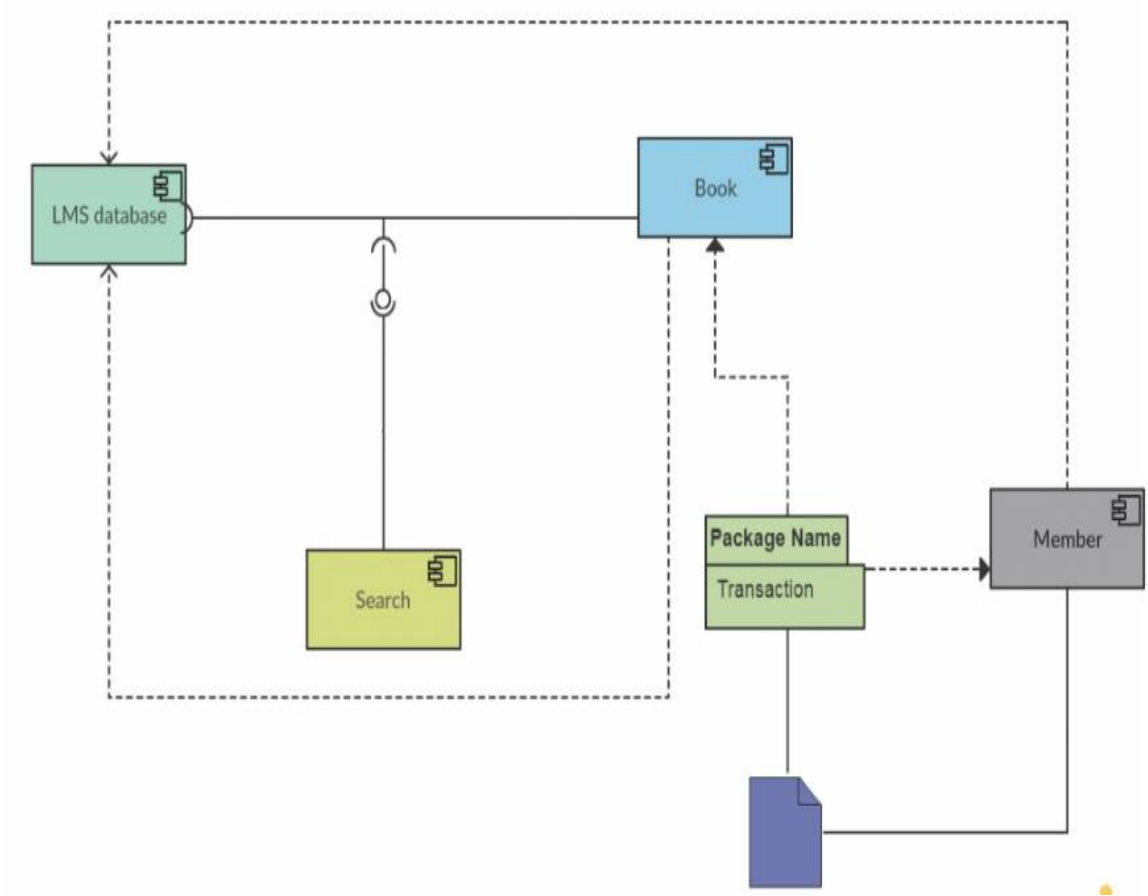
Step 1: figure out the purpose of the diagram and identify the artifacts such as the files, documents etc. in your system or application that you need to represent in your diagram.

Step 2: As you figure out the relationships between the elements you identified earlier, create a mental layout of your component diagram

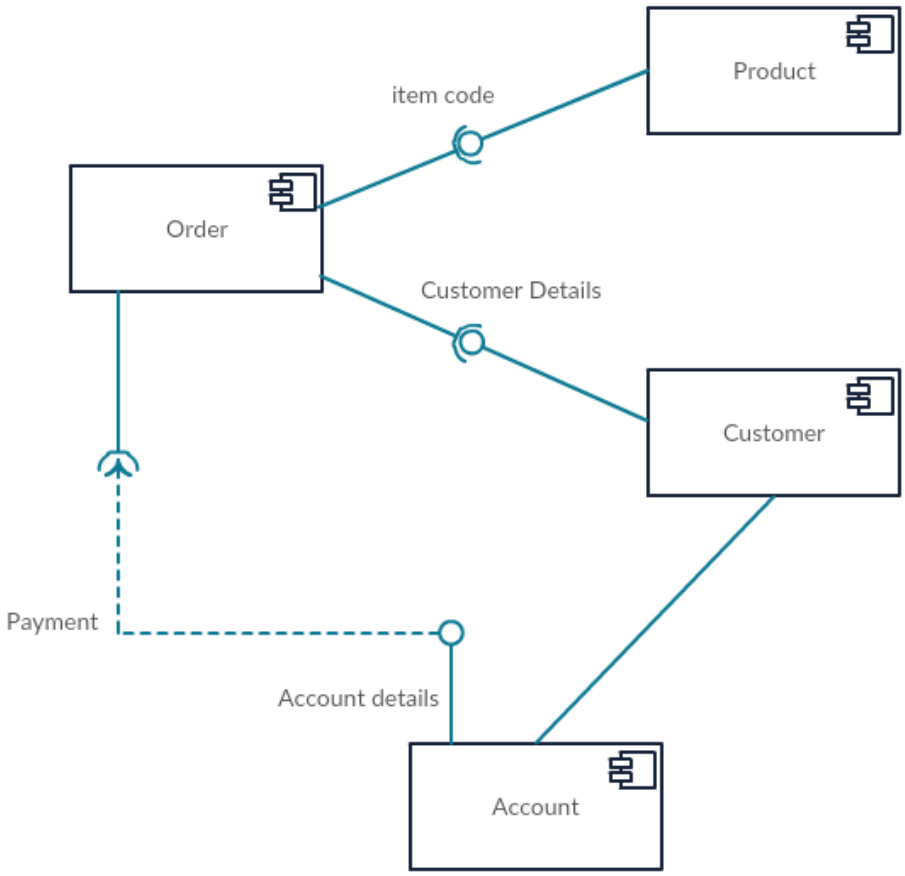
Step 3: As you draw the diagram, add components first, grouping them within other components as you see fit

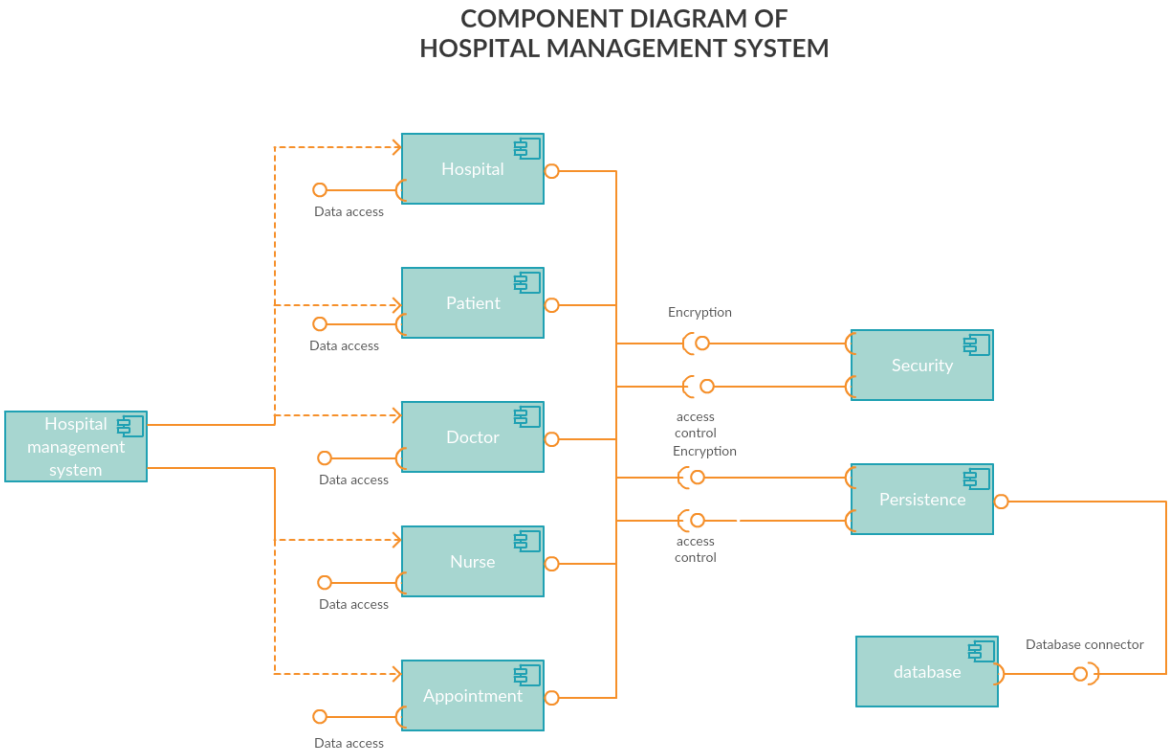
Step 4: Next step is to add other elements such as interfaces, classes, objects, dependencies etc. to your component diagram and complete it.

Step 5: You can attach notes on different parts of your component diagram to clarify certain details to others.



COMPONENT DIAGRAM FOR ONLINE SHOPPING SYSTEM





Deployment Diagrams

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

A UML deployment diagram depicts a static view of the run-time configuration of hardware nodes and the software components that run on those nodes. Deployment diagrams show the hardware for the system, the software that is installed on that hardware, and the middleware used to connect the disparate machines to one another.

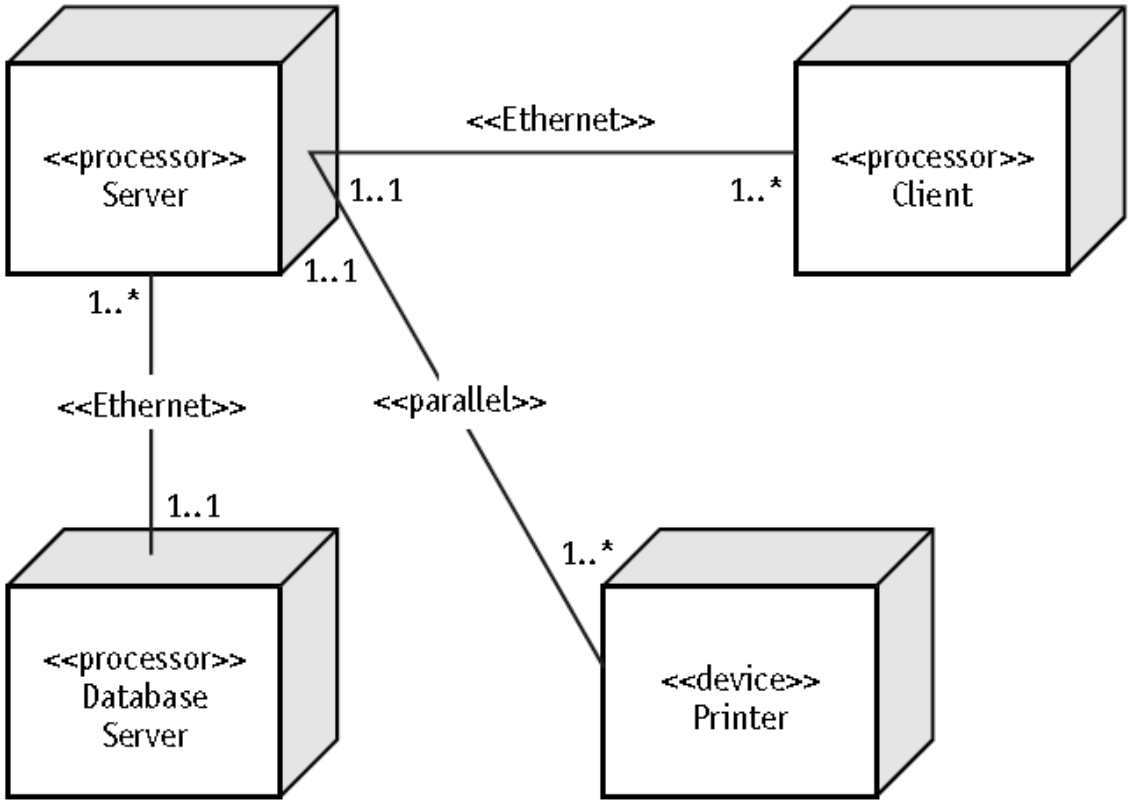
Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.

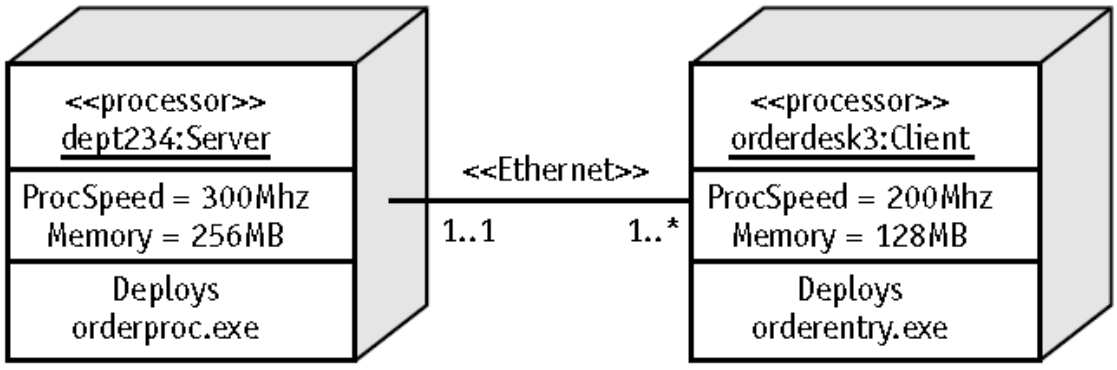
You create a deployment model to:

a) Explore the issues involved with installing your system into production.
b) Explore the dependencies that your system has with other systems that are currently in, or planned for, your production environment.
c) Depict a major deployment configuration of a business application.
d) Design the hardware and software configuration of an embedded system.
e) Depict the hardware/network infrastructure of an organization.

The Deployment diagram models the hardware architecture by identifying the processors.
Processors are typically computers but may also be people who perform manual processing.
Nodes, or processors, may contain and run software components. Node connections are modeled as associations, complete with stereotypes for names and multiplicity.

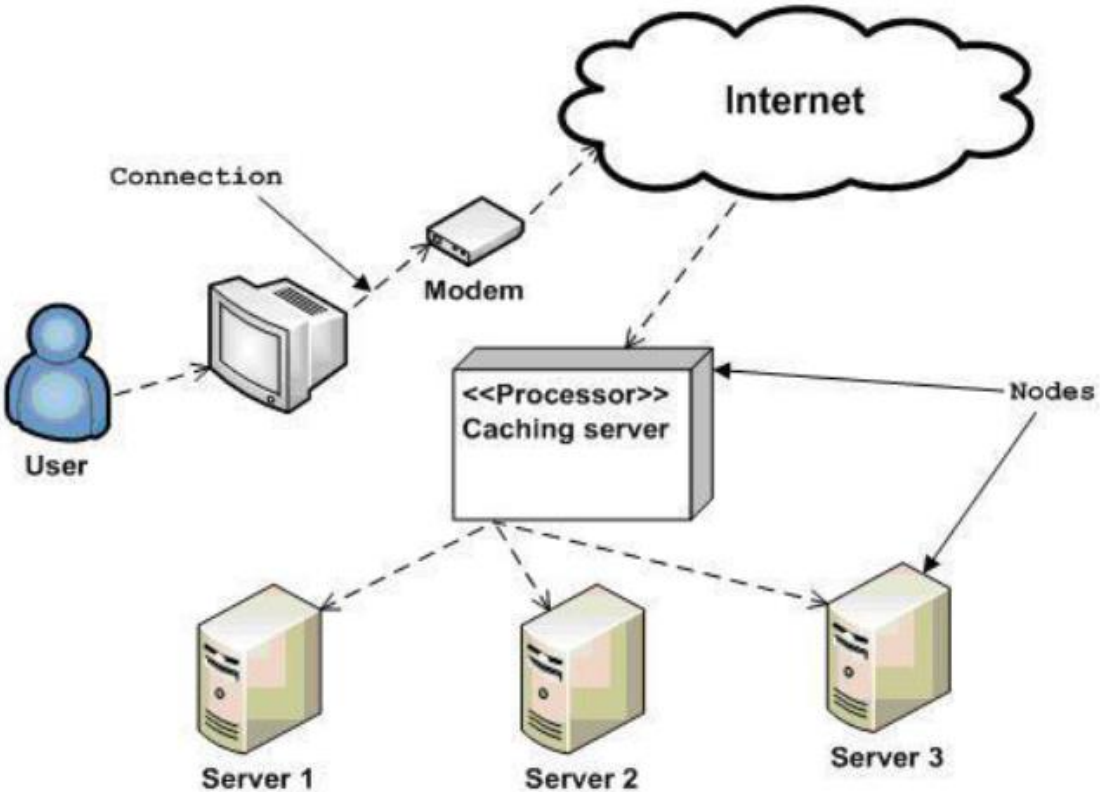
Component and Deployment diagrams may be combined. Components reside on a node. Component icons are placed inside the expanded node icon.
Communication between components on the nodes is modeled as dependencies. The dependencies model the logical communication requirements.


<<processor>>
Server
<<Ethernet>>
<<processor>>
Client
1..1
1..*
1..1
1..*
<<Ethernet>>
<<parallel>>
1..1
1..*
<<processor>>
Database
Server
<<device>>
Printer

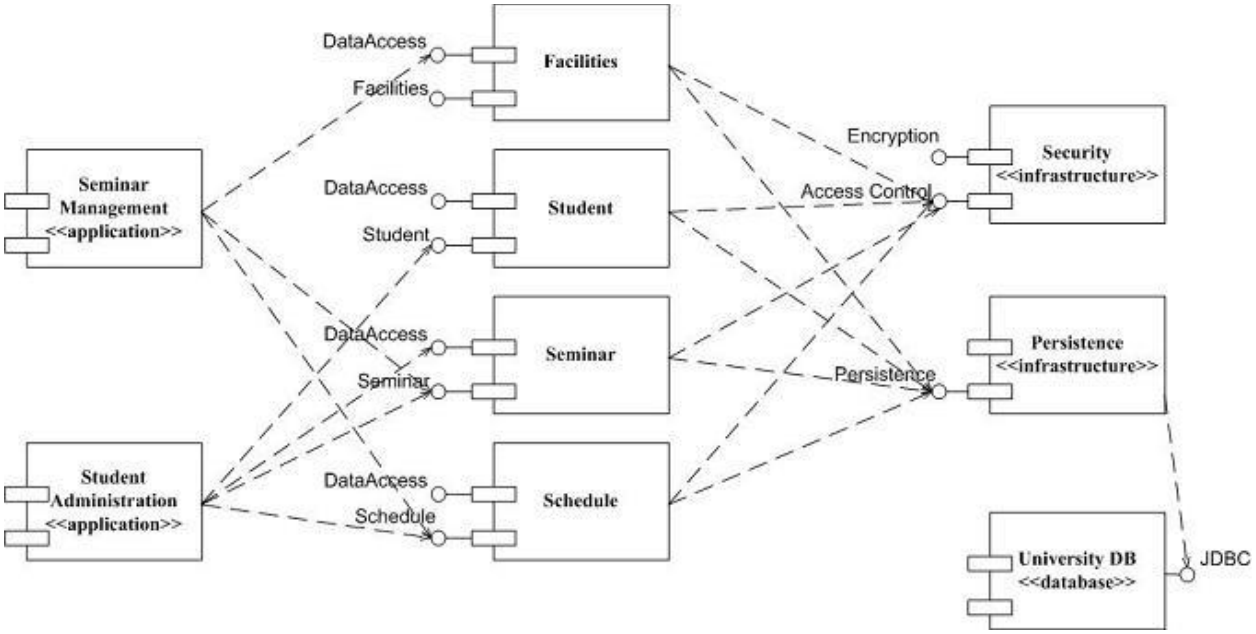

<<processor>>
dept234:Server
ProcSpeed = 300Mhz
Memory = 256MB
Deploys
orderproc.exe
<<Ethernet>>
1..1
1..*
<<processor>>
orderdesk3:Client
ProcSpeed = 200Mhz
Memory = 128MB
Deploys
orderentry.exe

\

Deployment diagram of an order management system

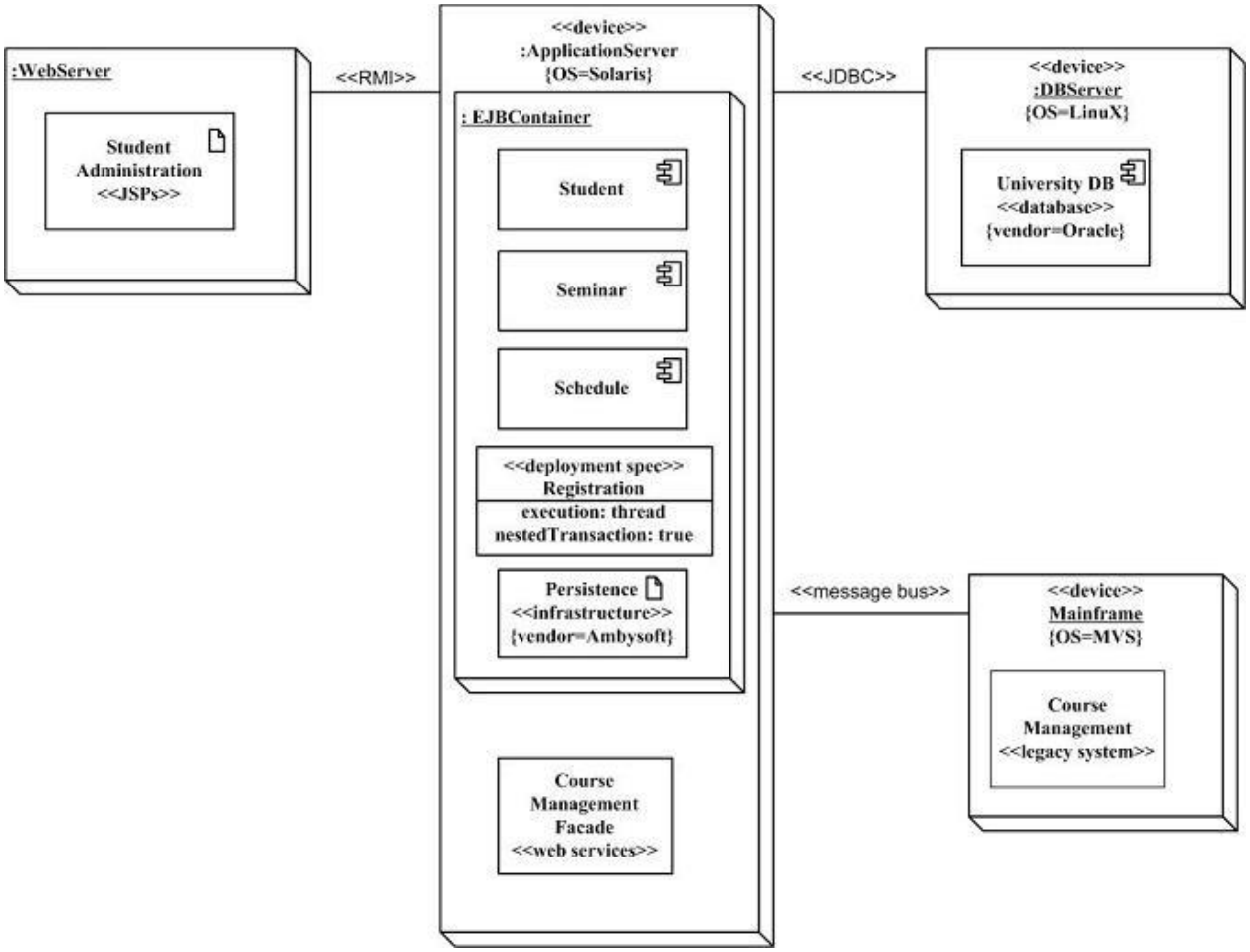


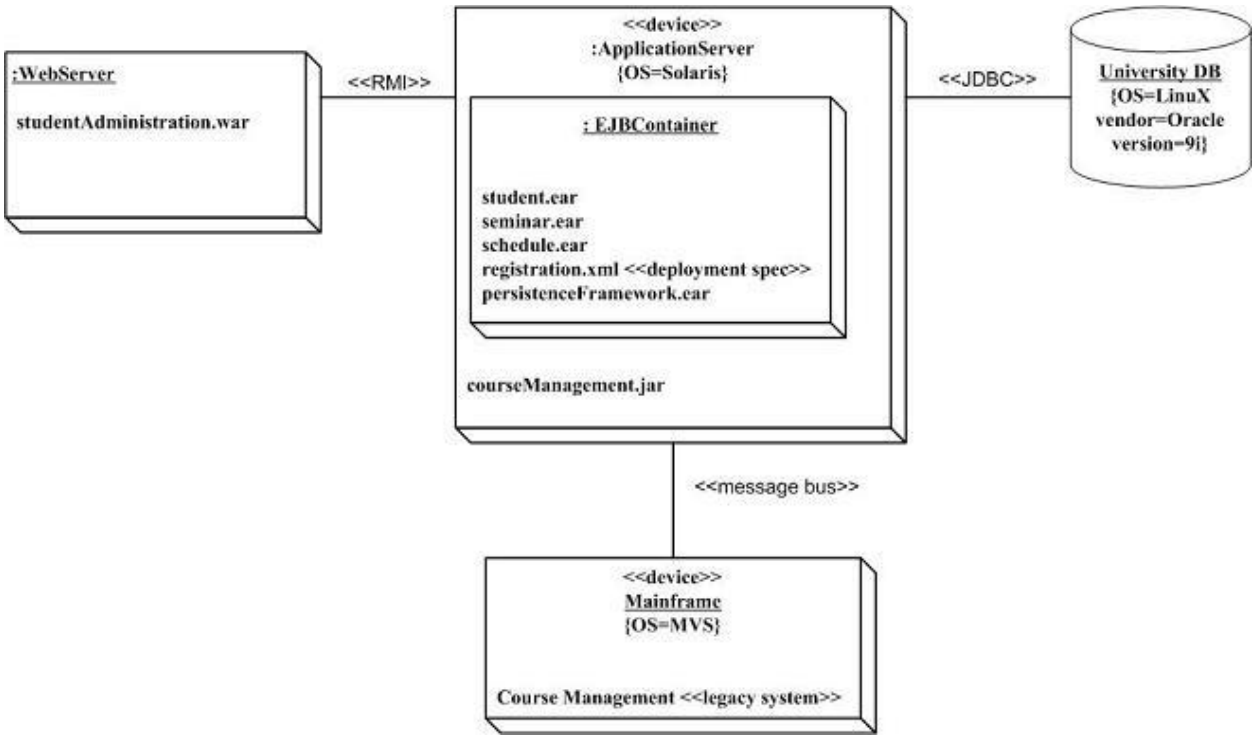
Sample Component Diagram



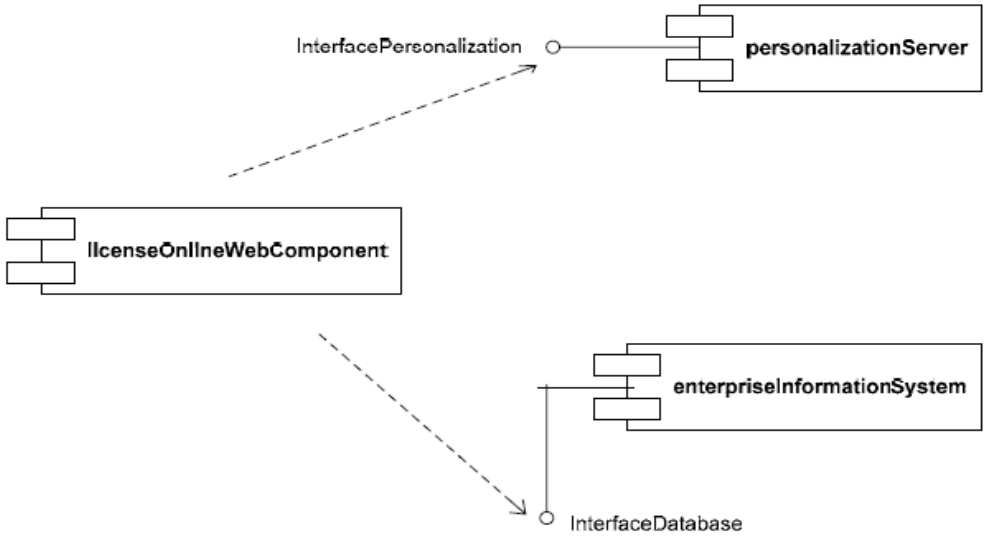
Diagrams such as one above are often referred to as "wiring diagrams" because they show how the various software components are "wired together" to build your overall application. The lines between components are often referred to as connectors, the implication being that some sort of messaging will occur across the connectors.

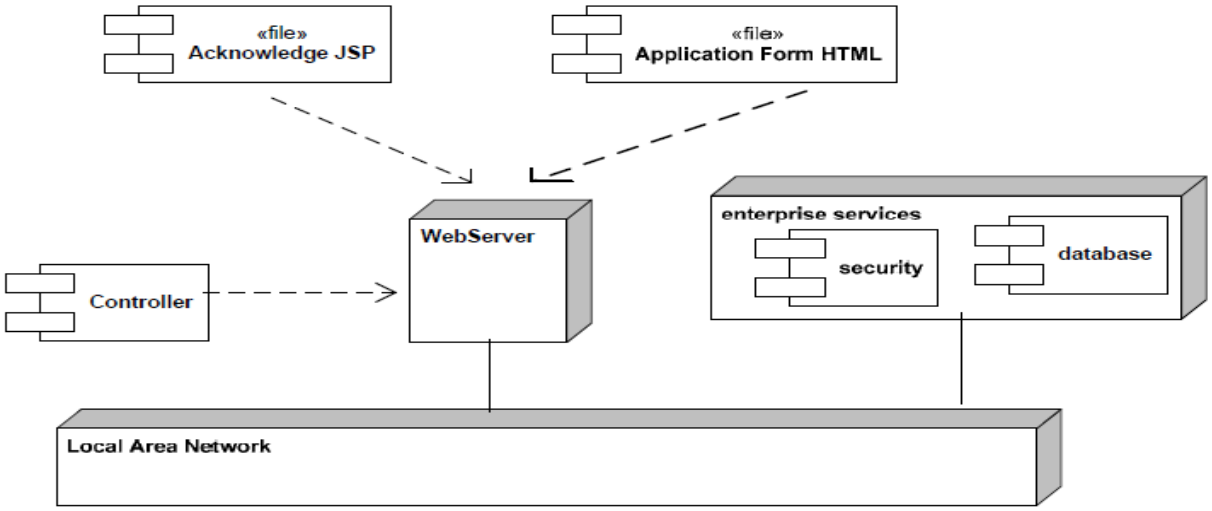
Sample Deployment Diagram





Components Diagram




«file»
Acknowledge JSP
«file»
Application Form HTML
WebServer
enterprise services
security
database
Controller
Local Area Network