SLIIT

Discover Your Future

Software Engineering (IT2020) 2022

Lecture 5 - Physical Diagram

SLIIT
FACULTY OF COMPUTING

UML Diagram Classification

Static (Structural Diagrams)

- Class diagrams
- Object Diagram

Dynamic (Behavioral Diagrams)

- State diagram
- Activity diagram
- Sequence diagram
- Collaboration diagram

Implementation (kind of a Structural diagrams)

- Physical Diagram
  - Component diagram
  - Deployment diagram

Implementation Diagram describes the elements required to implement a system

UML Physical Diagram

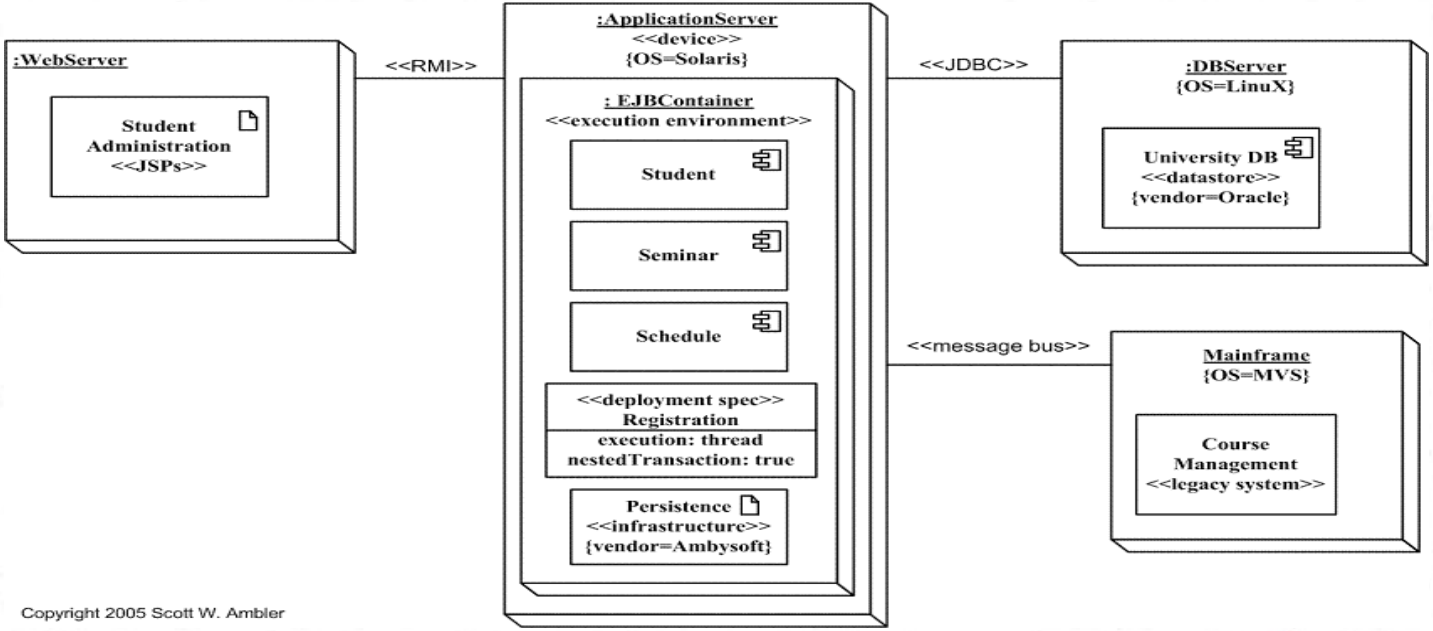
- Physical diagrams are used when the development of the system is complete.
- This use to give descriptions of the physical information about a system.
- Physical diagrams consists of two main diagrams. They are :
  - **Component diagram**
  - **Deployment diagram**

UML Physical Diagram

- **Component diagrams** show the software components of a system and how they are related to each other.
- **Deployment diagrams** show the physical relationship between hardware and software in a system.

Physical Diagrams	**=**	**Component Diagram**	**+**	**Deployment Diagram**

UML Physical Diagram Example



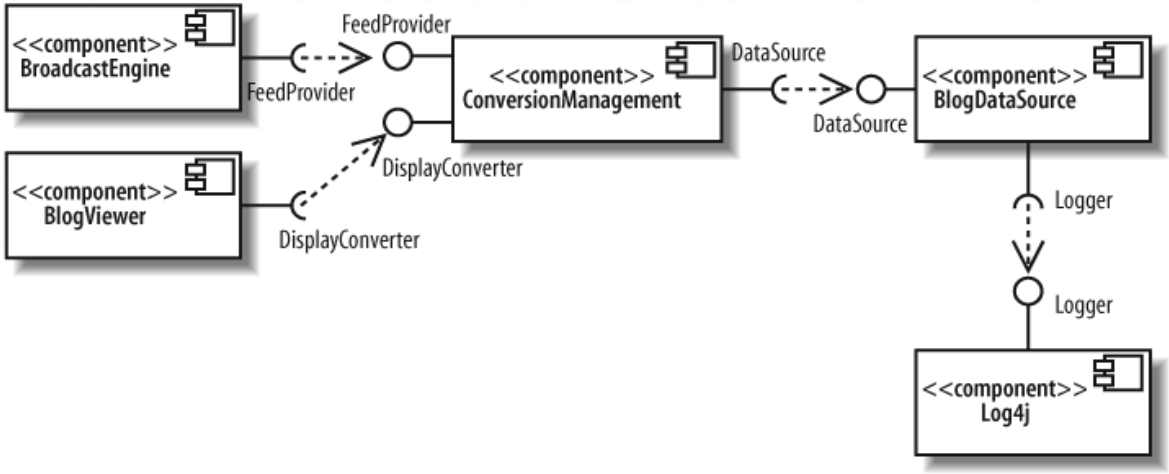
Component Diagram

Component Diagram

- Components are used to organize a system into manageable, reusable, and swappable pieces of software.
- Component Diagram Consists of:
    - Components
    - Interfaces
    - Relationships in between interfaces

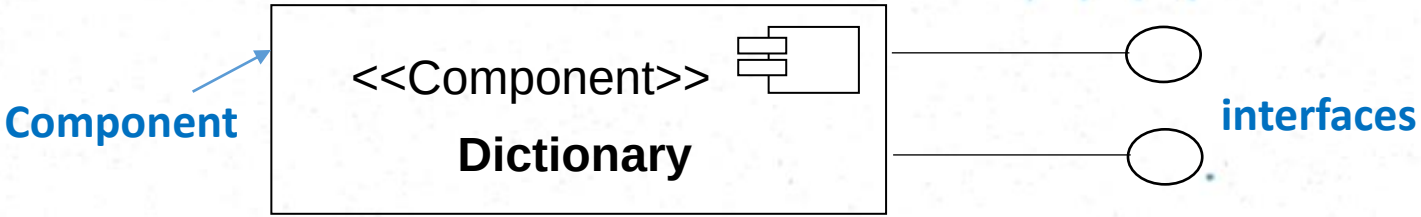
Component Diagram

- **Component diagram** shows components, provided and required interfaces and relationships between them.



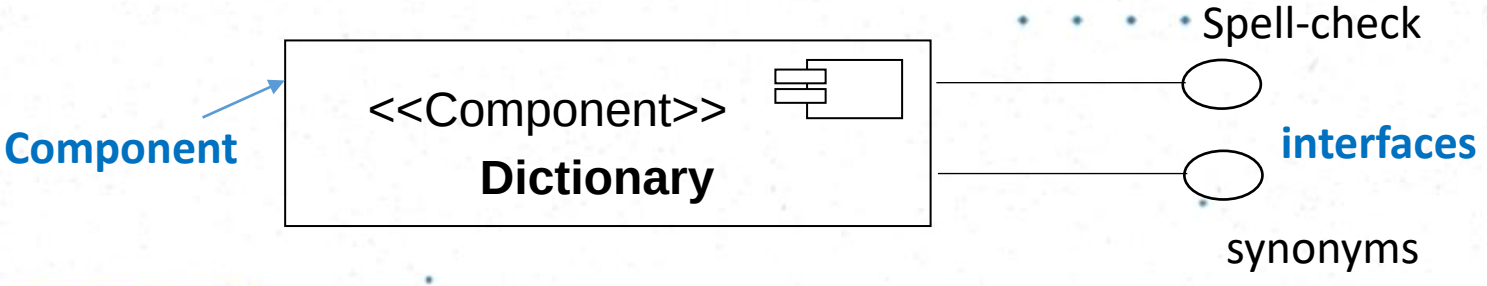
Components

- A Component is a physical unit of implementation with well-defined interfaces that is intended to be used as a replaceable part of a system.
- Components can be represented in different ways.
- Most common way is as **Rectangle**.
- A component is drawn as a rectangle with <<component>> stereotype and a tabbed rectangle icon in the upper right-hand corner.



Component Interfaces

- Component should be loosely coupled, so that they can be changed without changes on the other parts of the system.
- Interfaces are used for this purpose.
- An interface is a list of operations supported by a piece of software or hardware.



Component Interfaces

- Interfaces control dependencies between components and make components swappable.
- Each component realizes (supports) some interfaces and uses other components.
- Accordingly, there are two types of interfaces
  - Provided Interfaces
  - Required Interfaces

Interfaces

Provided Interfaces

- A *provided interface* of a component is an interface that the component realizes.
- Other components and classes interact with a component through its provided interfaces .
- A component's provided interface describes the services provided by the component.

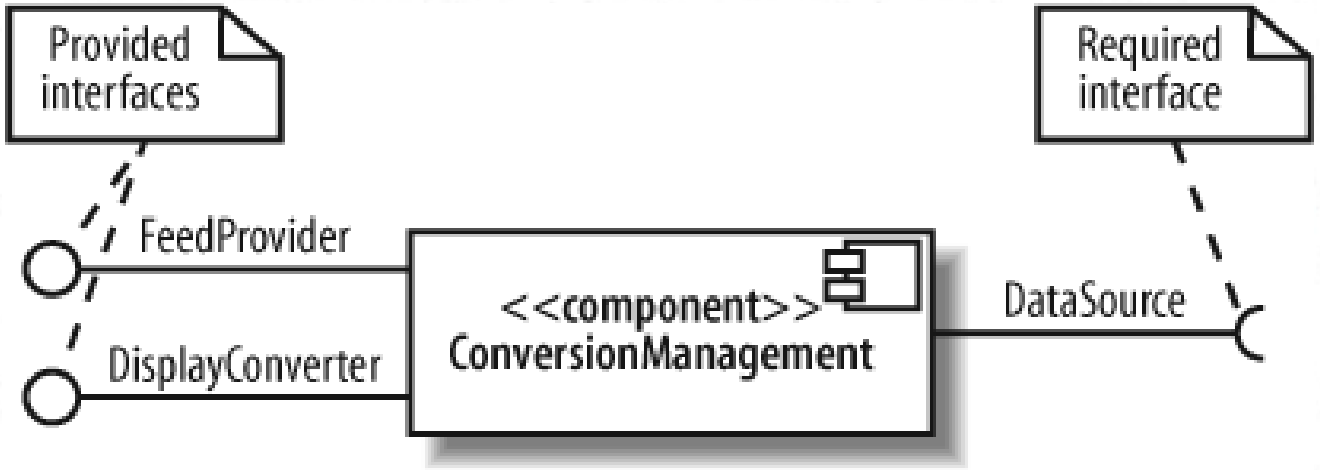
Interfaces

Required Interfaces

- A *required interface* of a component is an interface that the component needs to function.
- More precisely, the component needs another class or component that realizes that interface to function.

Interface Representations

The most common notation for interfaces is known as Ball-and-socket notation



Activity 1:

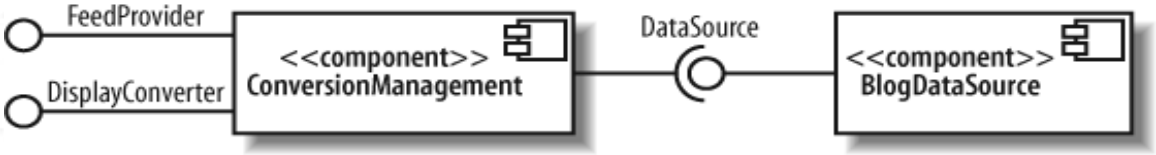
Order is a component which provides three interfaces for the other components to access. They are addLineItem, cancel and pay. It also uses applyDiscount interface to connect with other component.

Connecting Components

- There are two ways to connect components.
  - Using Dependancy relationship (Dependancy arrow)

  - Assembly Connector notation

Snapping the ball and socket together (omitting the dependency arrow)

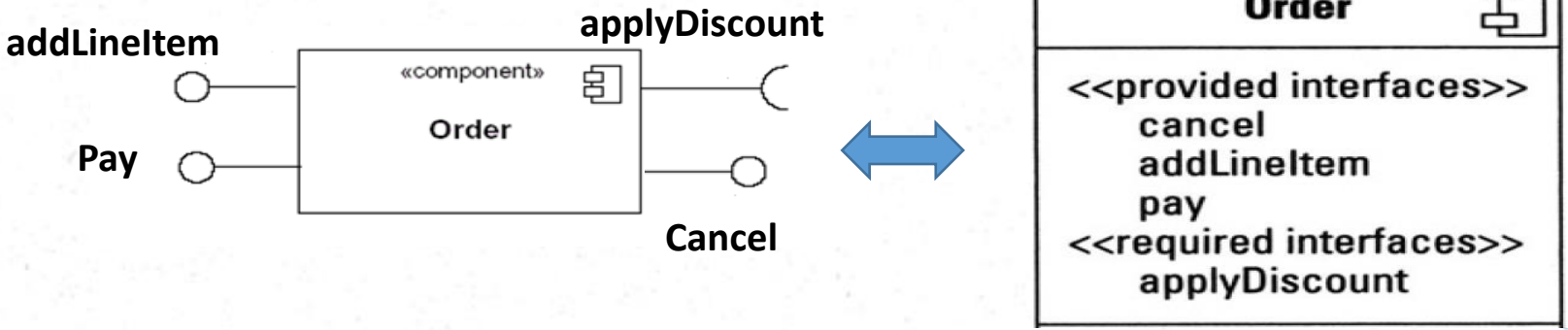


Additional Notations to represent Components and Interfaces

Components as Class Representations

White-box view

- The most compact way of showing required and provided interfaces is to list them inside the component.
- Provided and required interfaces are listed separately.



Activity 2

- Draw a component using white box notation for the given scenario.

- ConversionManagement component provides two interfaces; FeedProvider and Displayconverter. It also uses DataSource interface to access other components.

Additional Interface Representations

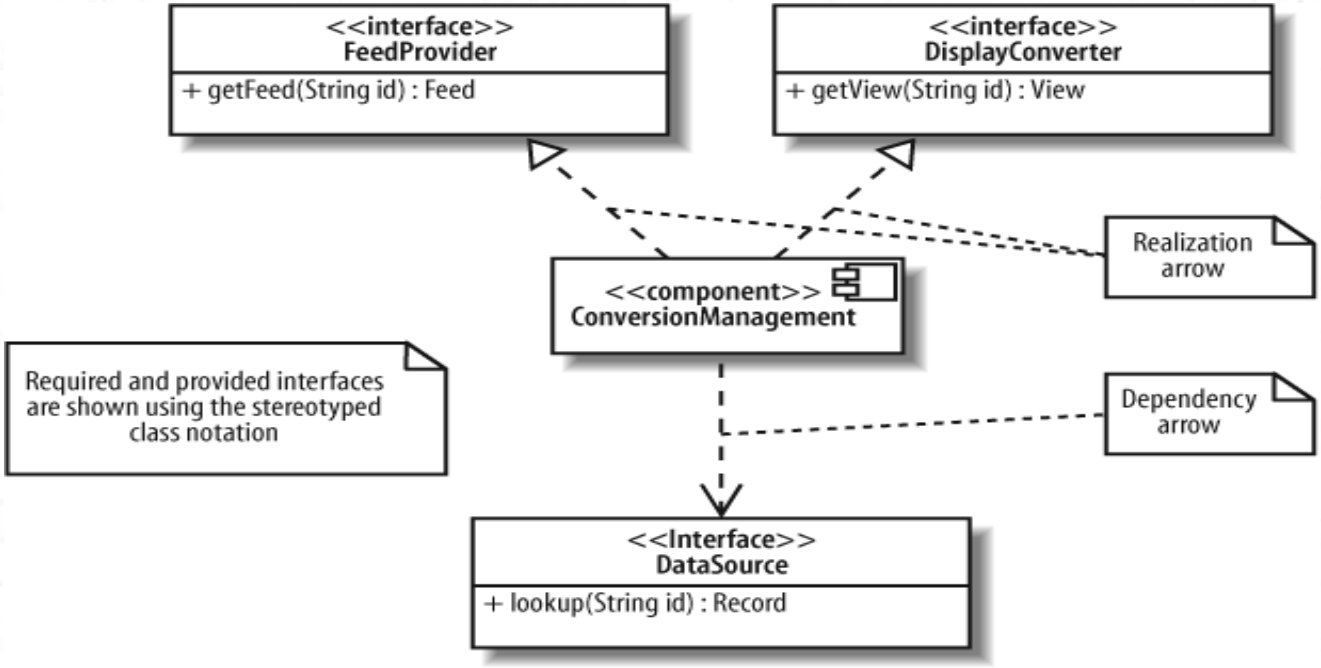
Stereotype Notation

Component's required and provided interfaces are shown by drawing the interfaces with the stereotyped class notation.

- If a component realizes an interface, draw a realization arrow from the component to the interface.
- If a component requires an interface, draw a dependency arrow from the component to the interface

Additional Interface Representations

Stereotype Notation (cont..)



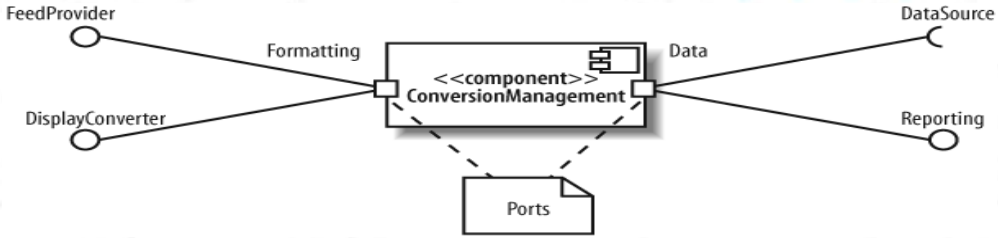
Activity 3

Draw a component diagram for the following.

- Order is a component which provides three interfaces for the other components to access.
- They are addLineItem, cancel and pay.
- Order component also needs to access itemCode, customerDetails, accountDetails and applyDiscount interfaces realized by Product, Customer, Account and Discount components respectively.

Ports

- Specifies a distinct interaction point
  - Between the component and its environment
  - Between the component and its internal parts
- Is shown as a small square symbol.
- Ports can be named, and the name is placed near the square symbol (Optional).

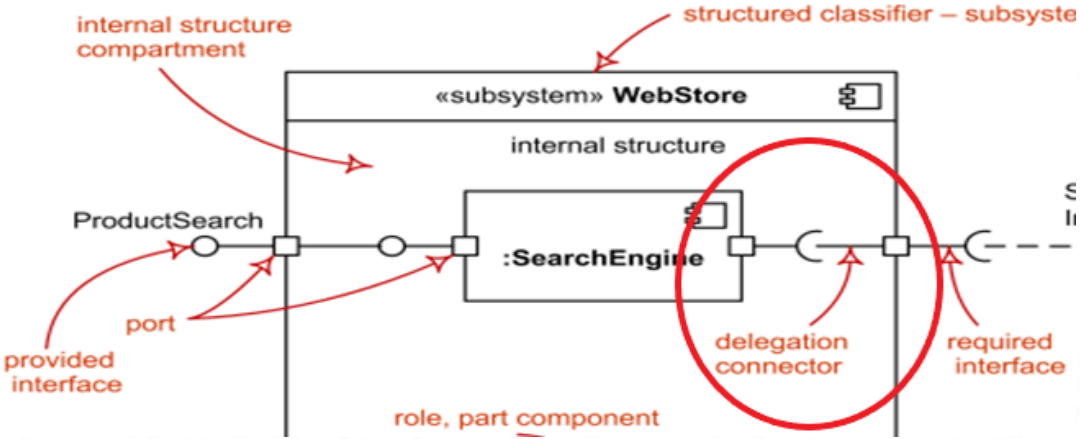


Delegation & Assembly Connectors

- Components have their own unique constructs when showing ports and internal structure called delegation connectors and assembly connectors.
- These are used to show how a component's interfaces match up with its internal parts and how the internal parts work together.

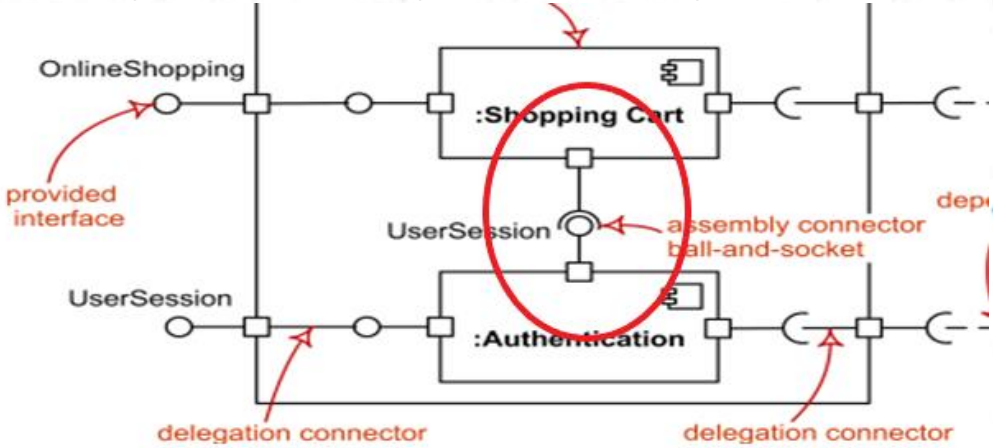
Delegation Connectors

- *Delegation connectors* are used to show that internal parts of the component realize or use the component's interfaces.

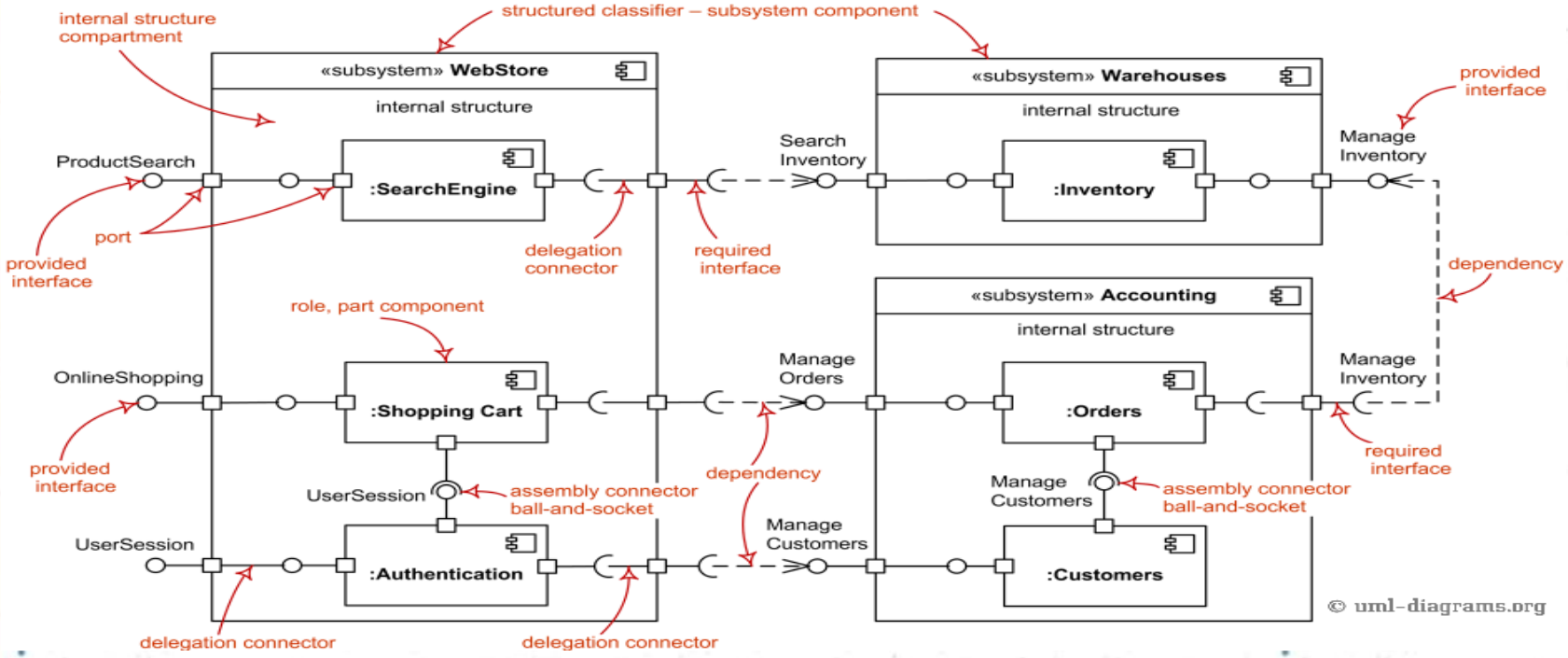


Assembly Connectors

- Assembly connectors are special kinds of connectors that are defined for use when showing composite structure of components.
- Assembly connectors snap together the ball and socket symbols that represent required and provided interfaces.



Example


internal structure compartment
structured classifier – subsystem component
«subsystem» WebStore
internal structure
«subsystem» Warehouses
internal structure
provided interface
ProductSearch
:SearchEngine
Search Inventory
:Inventory
Manage Inventory
port
provided interface
delegation connector
required interface
dependency
role, part component
«subsystem» Accounting
internal structure
OnlineShopping
:Shopping Cart
Manage Orders
:Orders
Manage Inventory
provided interface
UserSession
assembly connector ball-and-socket
dependency
Manage Customers
assembly connector ball-and-socket
required interface
UserSession
:Authentication
Manage Customers
:Customers
delegation connector
delegation connector
© uml-diagrams.org

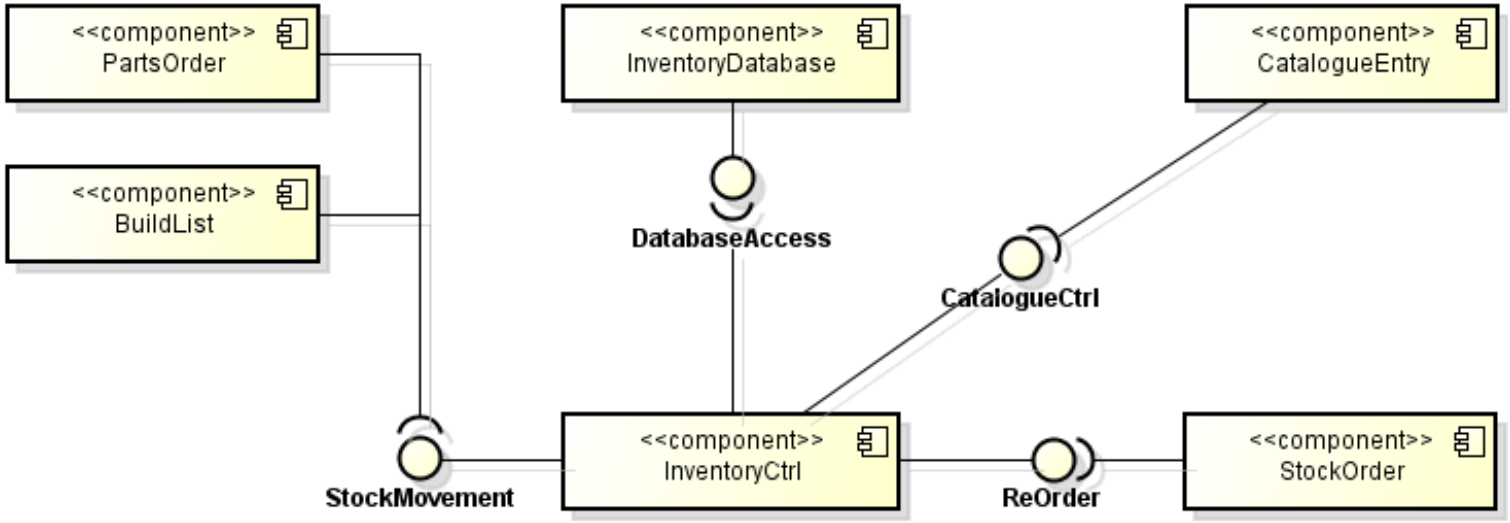
Activity 4

- The Conversion-Management component implements the FeedProvider and DisplayConverter interfaces.
- The Blog-Data-Source component implements the DataSource interface.
- The Log4j component implements the Logger interface.
- The BlogDataSoruce component requires the Logger interface of the Log4j
- The Conversion Management component requires the DataSource interface of Blog-Data-Source.
- The BroadCastEngine component requires the Feed-provider interface of the Conversion-Manamgenet component
- The BlogViewer component requires the DisplayConverter interface of the Conversion-Management component.

Activity 5

- PartsOrder component and BuildList component use InventoryCtrl component through StockMovement interface.
- InventoryDatabase component realizes DatabaseAccess interface which uses by InventoryCtrl.
- Also InventoryCtrl component realizes ReOrder and CatalogueCtrl interfaces that use by StockOrder and CatalogueEntry components respectively.

Activity 5: Answer



Deployment Diagram

Deployment Diagram

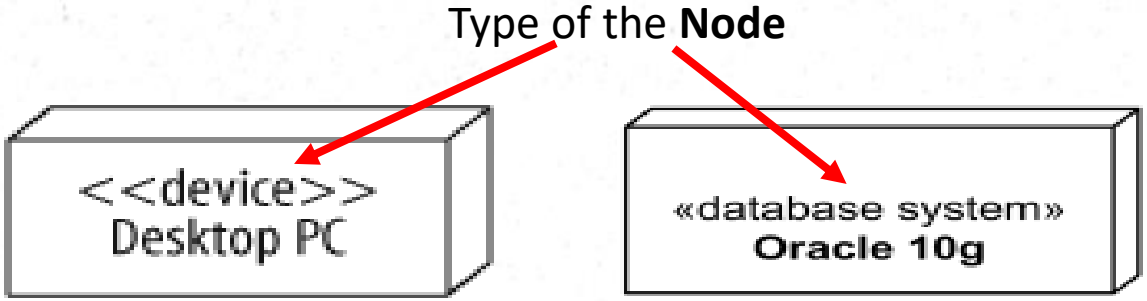
- UML *deployment diagrams* show the physical view of your system, bringing your software into the real world by showing how software gets assigned to hardware and how the pieces communicate

Components of a Deployment Diagram

- Nodes (Hardware / Software units)
- Links between nodes (connections)

Nodes

- A *node* is a hardware or software resource that can host software or related files.
- A node is drawn as a cube with its type written inside.



- There are two types of nodes;
  - Hardware nodes
  - Software nodes (Execution environments)

Hardware Nodes

- Hardware nodes use to show computer hardware.

- It’s labeled with the stereotype <<device>> to specify that this is a hardware node.
- The following items are common examples of hardware nodes:
  - Hardware Server
  - Desktop PC
  - Disk drives

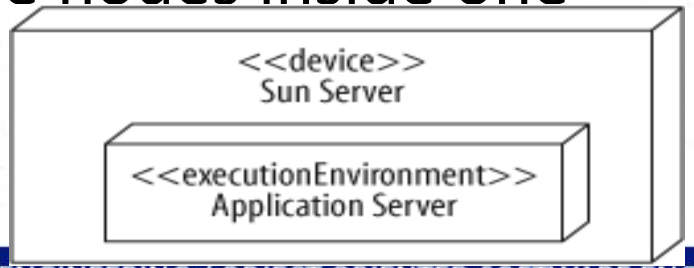
Software Nodes

- Also known as software environments.
- A software node is an application context; not parts of the software going to developed, but a third-party environment that provides services to software going to develop.
- The following items are examples of execution environment nodes:
  - Operating system
  - Web server
  - Application server
  - workflow engine
  - database system
  - web browser
  - J2EE container

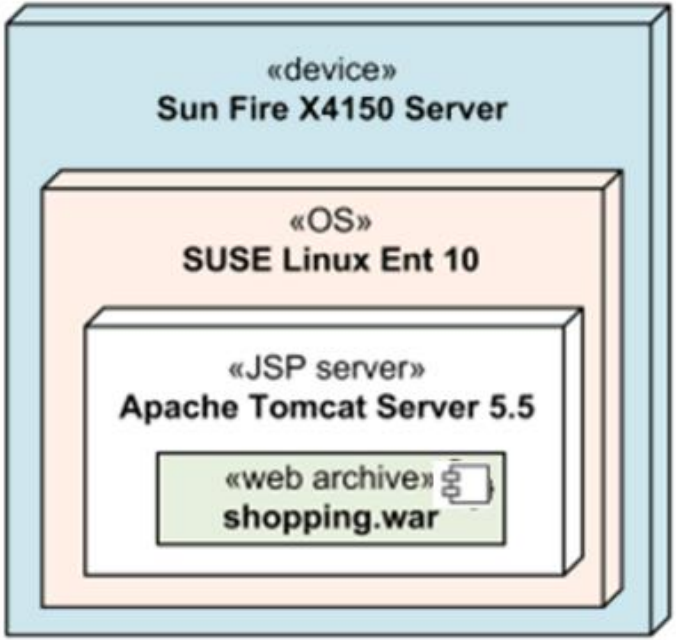
- It's labeled with the stereotype <<executionEnvironment>> to specify that this is a software node.

Nested Nodes

- Execution environments do not exist on their own, they run on hardware.
- For example, an operating system needs computer hardware to run on.
- Execution environment resides on a particular device indicate by placing the nodes inside one another, nesting them.



Nested Node: Example



Activity 6

Draw a nested node according to the following description.

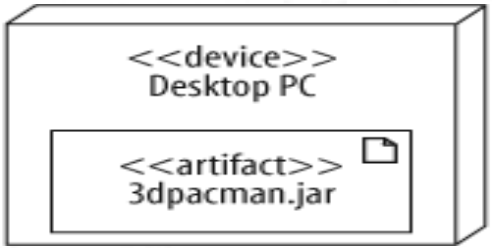
A Hardware server contains a processor. A J2EEServer execution node resides inside the processor. A PerformanceEJB component is installed in the J2EEServer execution node .

Artifacts

- *Artifacts* are physical files that execute or are used by the software.
- Common artifacts you'll encounter include:
  - Executable files, such as *.exe* or *.jar* files
  - Library files, such as *.dll*s (or support *.jar* files)
  - Source files, such as *.java* or *.cpp* files
  - Configuration files that are used by software at runtime, in formats such as *.xml*, *.properties*, or *.txt*
- An artifact is shown as a rectangle with the stereotype <<artifact>>, or the document icon in the upper right hand corner, or both.

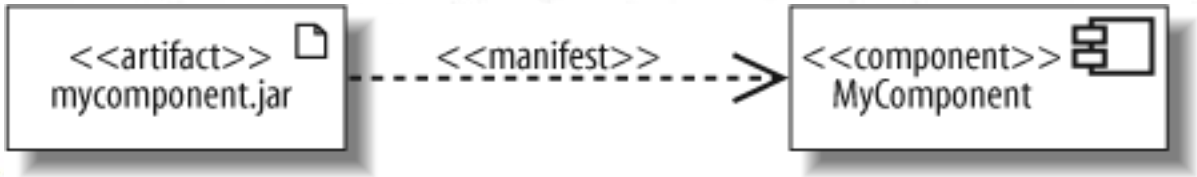
Artifacts

- Deploying an Artifact to a Node.



- Binding an Artifact to a Component.

If an artifact is the physical actualization of a component, then the artifact *manifests* that component.



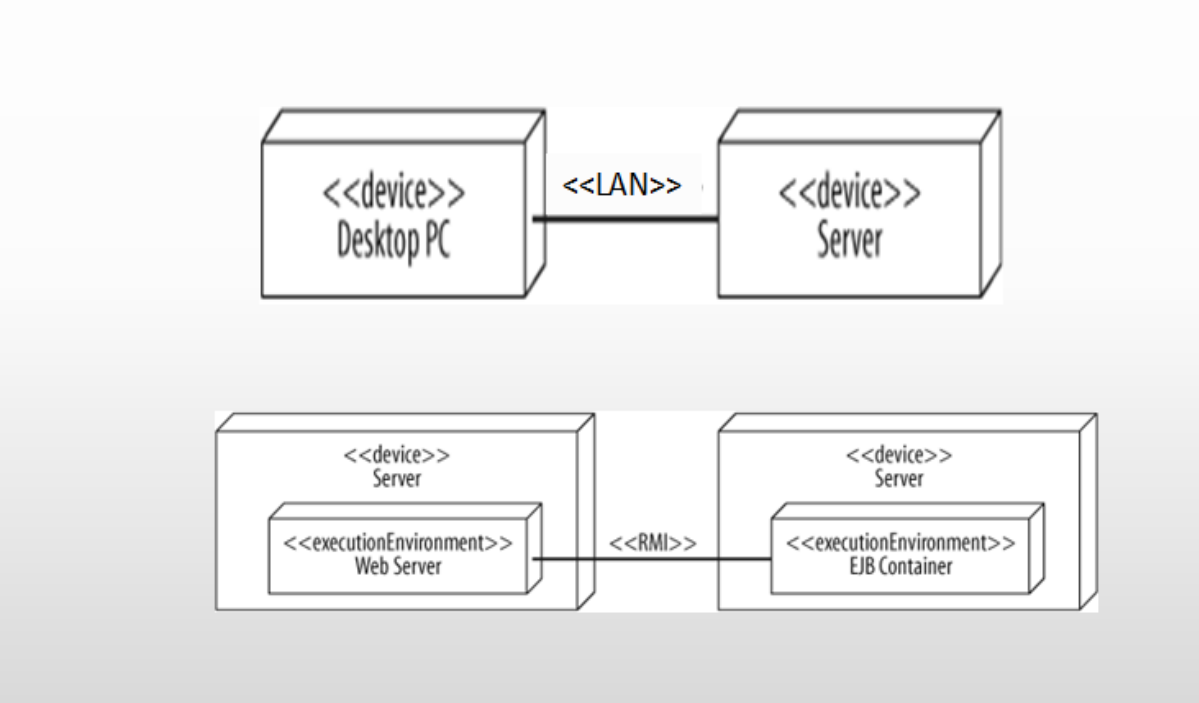
Communication paths / Links

- To get its job done, a node may need to communicate with other nodes.
- For example, a client application running on a desktop PC may retrieve data from a server using TCP/IP.
- *Communication paths / links* are used to show that nodes communicate with each other at runtime.
- Communication paths can be a physical connection (ex: LAN / WAN) or a protocol such as TCP/IP.

Communication paths / Links

- A communication path is drawn as a solid line connecting two nodes. The type of communication is shown by adding a stereotype to the path.
- communication paths also can be shown in between execution environment nodes.
- Example: A web server communicating with an EJB container through RMI
- When deployment targets are **execution environments**, communication path will typically represent some protocol.

Communication paths / Links



Activity 7

- Given below is a partial description of a web-based application developed for an online banking system “Smart-E-Banking”. Model a physical diagram according the given description.
- UI Component contains all the user interfaces of this system. It is installed in the main web server called UI_SEB WebServer. This web server runs in a Lenovo ThinkServer hardware server. It is installed windows OS. UI component is responsible to create i_SEB interface, which used by the SmartBanking_Webclient web application of the desktop.

Activity 7 cont...

- The desktop which is having windows OS, access the system through “SmartBanking_Webclient” web application connected to the UI component.
- There is a connection in between browser and Lenovo ThinkServer OS via HTTP.

References

- The Unified Modeling Language Reference Manual, Second Edition.
- Learning UML 2.0 by Kim Hamilton, Russ Miles.