SE-2002

SOFTWARE DESIGN AND ARCHITECTURE

RUBAB JAFFAR

RUBAB.JAFFAR@NU.EDU.PK

Implementation diagrams
Lecture # 31, 32, 33

TODAY'S OUTLINE

- Implementation Diagrams
  - Component Diagram
  - Deployment Diagram
- Introduction about components
- Components and component diagrams
- Elements of the component
- Component view: black-box view and white-box view

i. Static

a. Use case diagram

b. Class diagram

ii. Dynamic

a. Activity diagram

b. Sequence diagram

c. Object diagram

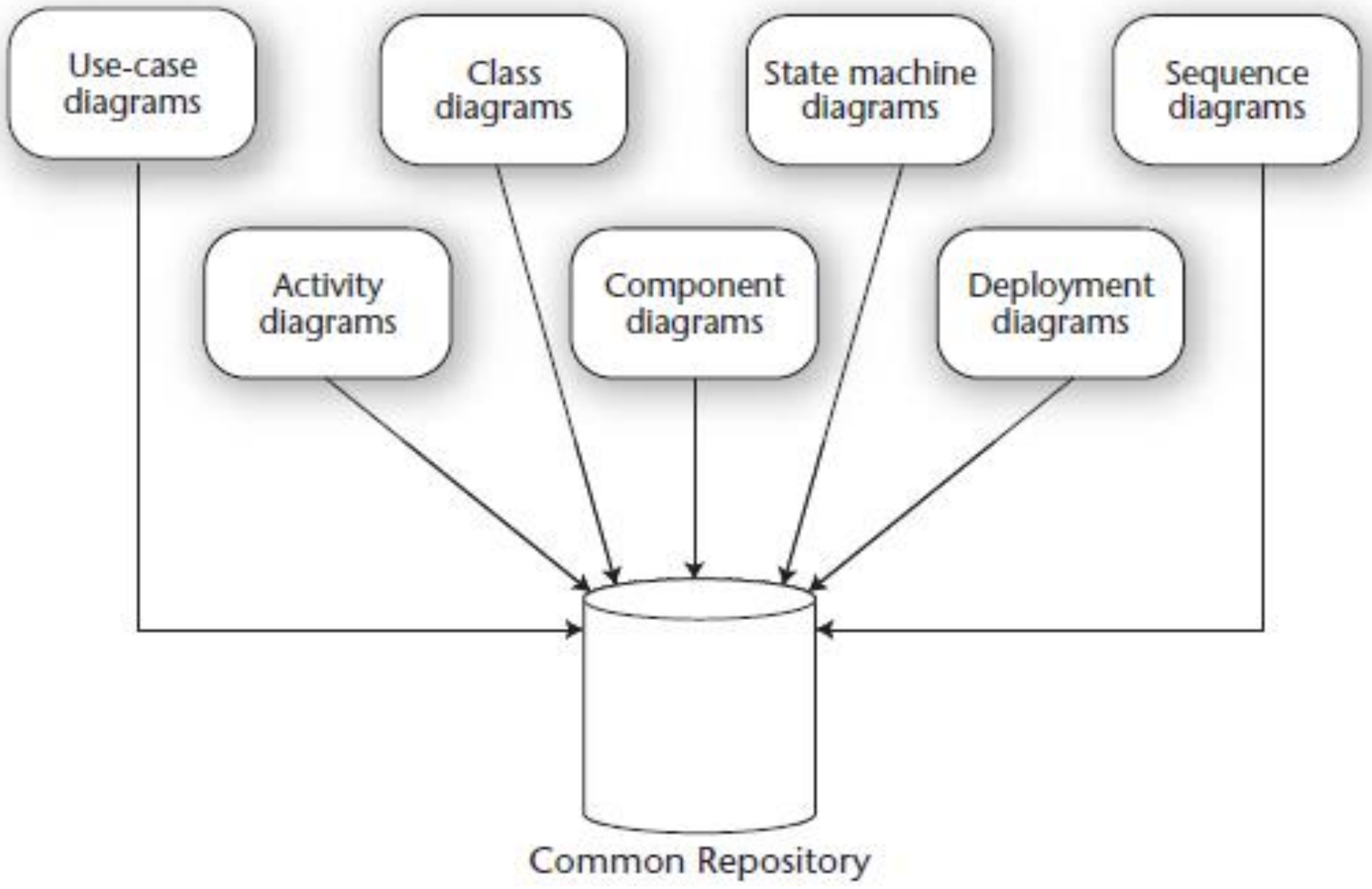
d. State diagram

e. Collaboration diagram

iii. Implementation

a. Component diagram

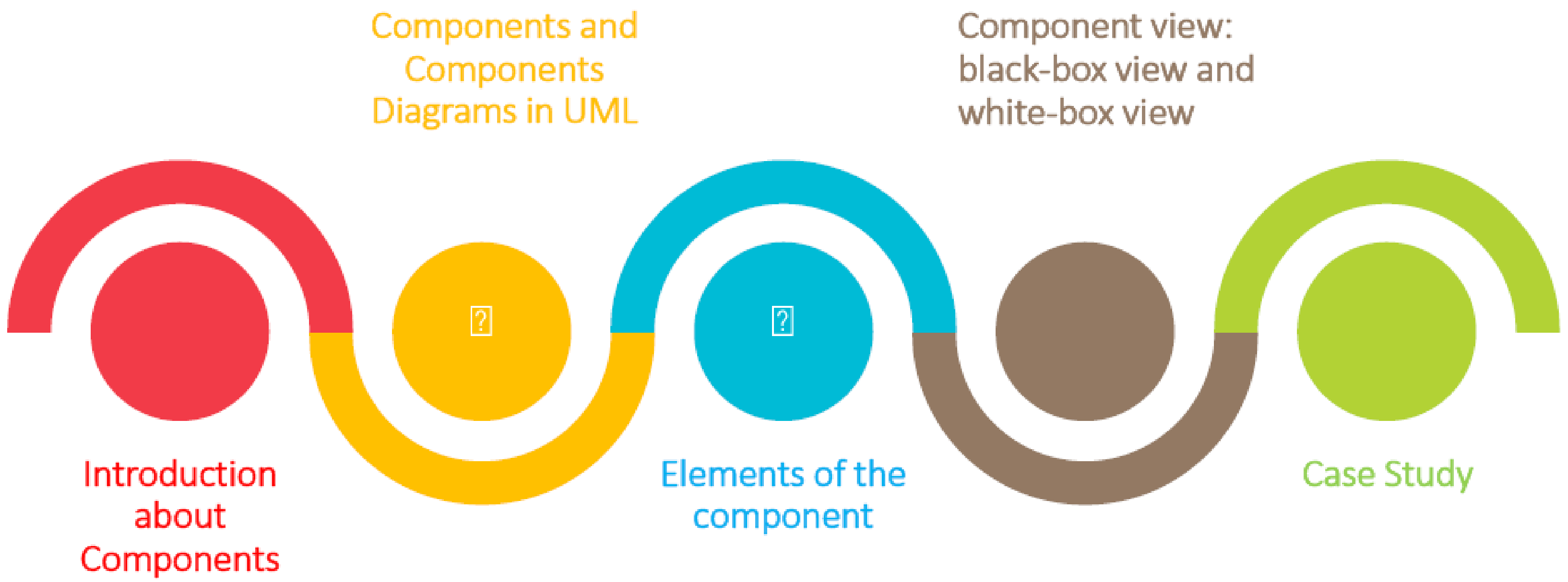
b. Deployment diagram


Use-case diagrams
Class diagrams
State machine diagrams
Sequence diagrams
Activity diagrams
Component diagrams
Deployment diagrams
Common Repository

IMPLEMENTATION DIAGRAMS

- Both are structural diagrams
- **Component Diagrams:**
  - set of components and their relationships
  - Illustrate static implementation view
  - Component maps to one or more classes, interfaces, or Collaborations
- **Deployment Diagrams:**
  - Set of nodes and their relationships
  - Illustrate static deployment view of architecture
  - Node typically encloses one or more components

PLAN OF TALK


Components and Components Diagrams in UML
Component view: black-box view and white-box view
Introduction about Components
Elements of the component
Case Study

WHAT IS COMPONENT?

- A component is an autonomous unit within a system.
- A component is a self-contained unit that encapsulates the state and behavior of a set of classifiers.
- All the contents of the components are private—hidden inside.
- Also unlike a package, a component realizes and requires interfaces.
- The components can be used to describe a software system of arbitrary size and complexity.

COMPONENT DIAGRAM

- A component diagram breaks down the actual system under development into various high levels of functionality.
- Each component is responsible for one clear aim within the entire system and only interacts with other essential elements on a need-to-know basis.

COMPONENT DIAGRAM

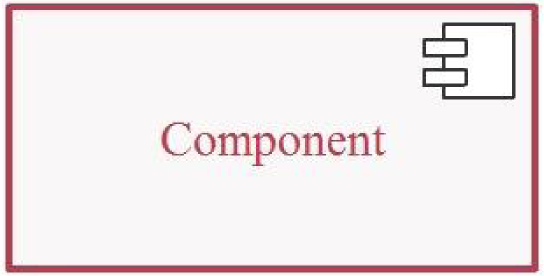
- Models the physical implementation of the software
- Models the high-level software components, and their interfaces
- Elements of component diagram are not much different than what we have seen in class diagram(Classes, interface, relationships)
- Dependencies are designed such that they can be treated as independently as possible
- So, Special kind of class diagram focusing on system's Components.
- Components to use with Component Diagram are:
  - Components required to run the system(library file, etc.)
  - Source code file, and data file
  - Executable file (.exe)

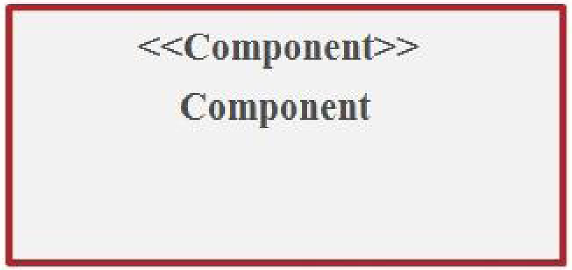
COMPONENT DIAGRAM ELEMENTS

- Component diagram shows
  - components,
  - Provided interface
  - required interfaces,
  - ports, and
  - Relationships between them.

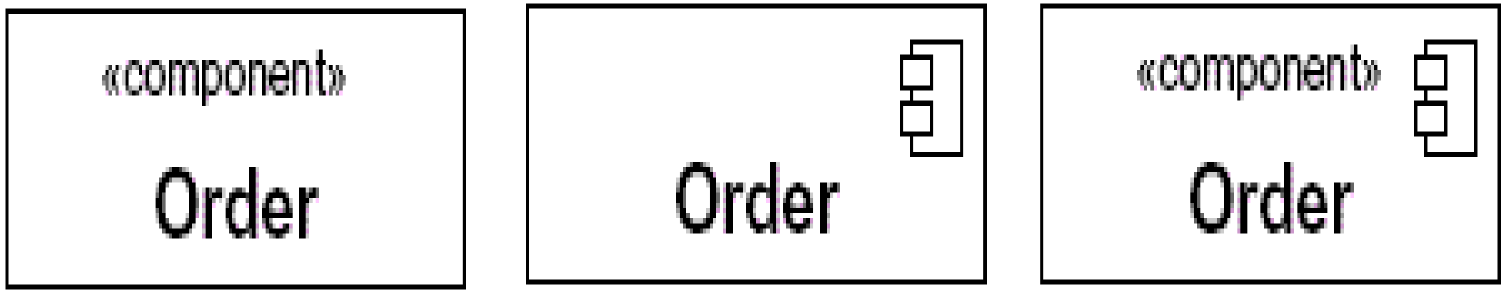
COMPONENT NOTATION

- A component is shown as a rectangle with
  - A keyword <<component>>
  - Optionally, in the right hand corner a component icon can be displayed.
    - A component icon is a rectangle with two smaller rectangles jutting out from the left-hand side
    - This symbol is a visual stereotype
  - The component name
- Components can be labelled with a stereotype
- Th





WAYS TO REPRESENT COMPONENTS

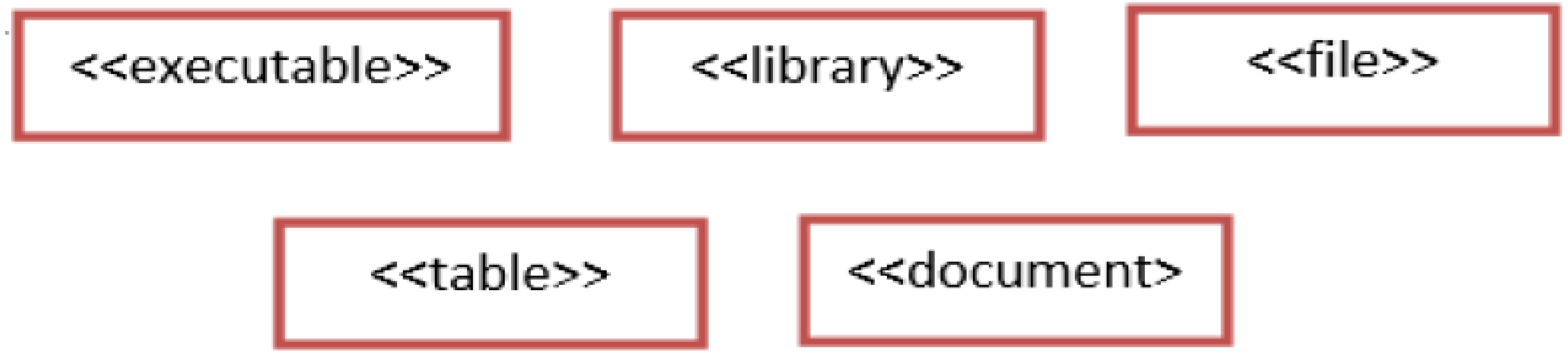


COMPONENT

- A component may be replaced by another if and only if their provided and required interfaces are identical.
- This idea is the underpinning for the plug-and play capability of component-based systems and promotes software reuse.
- UML places no restriction on the granularity of a component. Thus, a component may be as small as a figures-to-words converter, or as large as an entire document management system.
- Such assemblies are illustrated by means of component diagrams.

COMPONENT STEREOTYPES

- Components stereotype provides visual cues about roles played by components in a system. Some of component stereotype are as follows.



- **<<executable>>: executable file (.exe)**
- **<<library>>: references resources (.dll)**
- **<<file>>: text file, source code file, etc.**
- **<<table>>: database file, table file, etc.**
- **<<document>>: document file, web page file, etc.**

COMMON STEREOTYPES

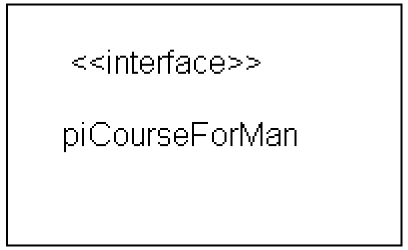
Stereotype	**Indicates**
<<application>>	A “front-end” of your system, such as the collection of HTML pages and ASP/JSPs that work with them for a browser-based system or the collection of screens and controller classes for a GUI-based system.
<<database>>	A hierarchical, relational, object-relational, network, or object-oriented database.
<<document>>	A document. A UML standard stereotype.
<<executable>>	A software component that can be executed on a node. A UML standard stereotype.
<<file>>	A data file. A UML standard stereotype.
<<infrastructure>>	A technical component within your system such as a persistence service or an audit logger.
<<library>>	An object or function library. A UML standard stereotype.
<<source code>>	A source code file, such as a .java file or a .cpp file.
<<table>>	A data table within a database. A UML standard stereotype
<<web service>>	One or more web services.
<<XML DTD>>	An XML DTD.

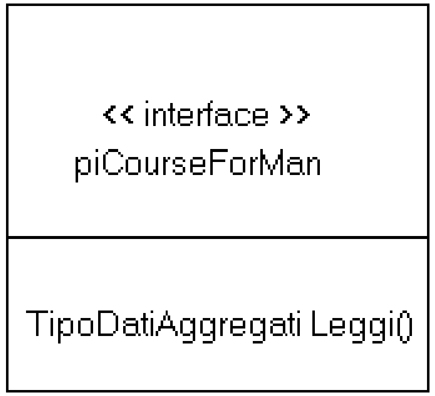
INTERFACES

- **An interface**
  - Is the definition of a collection of one or more operations
  - Provides only the operations but not the implementation
  - Implementation is normally provided by a class/ component
  - In complex systems, the physical implementation is provided by a group of classes rather than a single class
- A class can implement one or more interfaces
- An interface can be implemented by 1 or more classes

INTERFACES

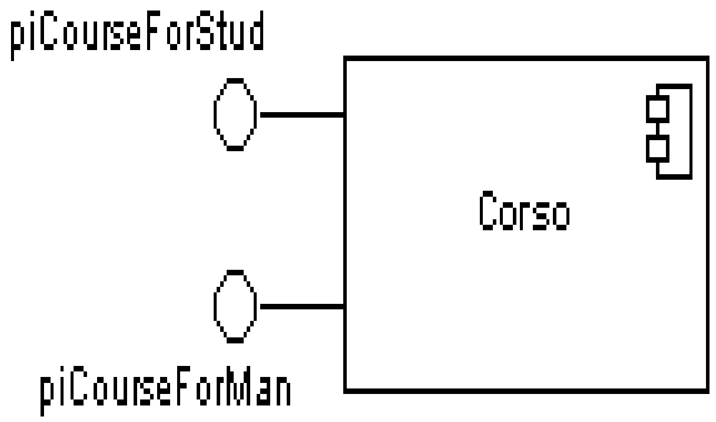
- May be shown using a rectangle symbol with a keyword <<interface>> preceding the name.
- For displaying the full signature, the interface rectangle can be expanded to show details
- Can be
  - Provided
  - Required
- The purpose is:
  - To control dependencies between components
  - To make components swappable





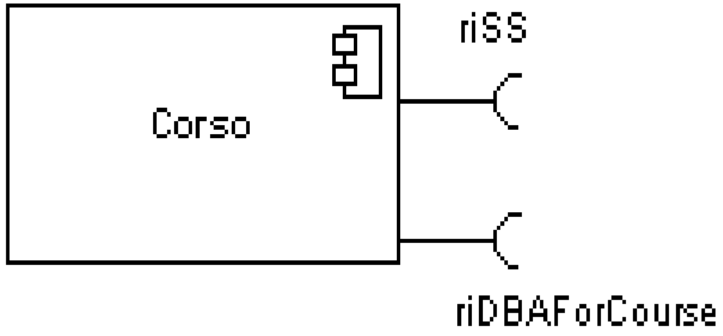
PROVIDED INTERFACES

- **Provided Interface:**
- Characterize services that the component offers to its environment
- Is modeled using a ball, labelled with the name, attached by a solid line to the component. Also known as Lollipop notation.

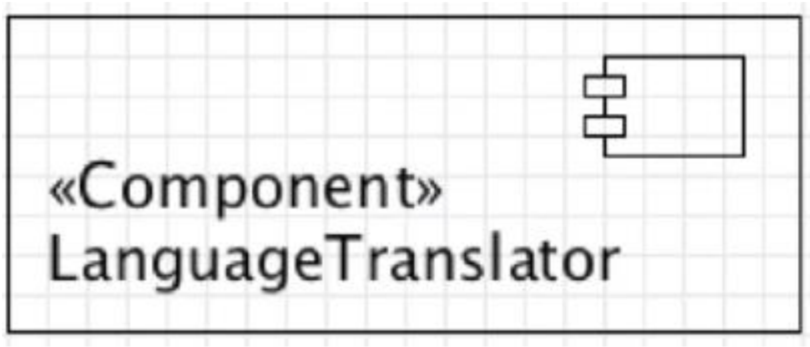


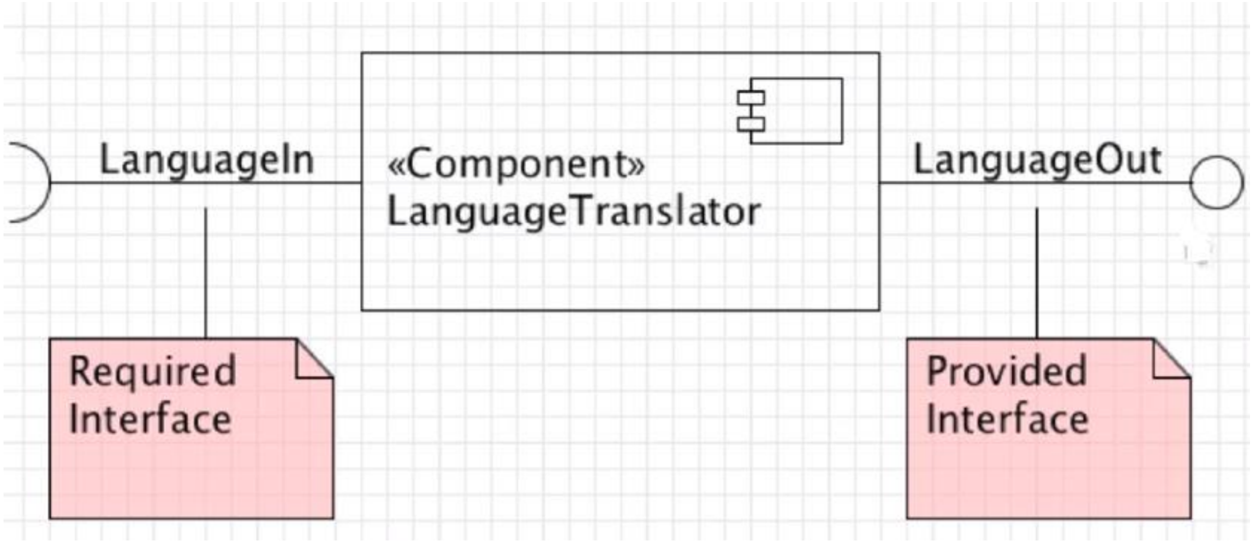
REQUIRED INTERFACES

- **Required Interface:**
- Characterize services that the component expects from its environment
- Is modeled using a socket, labelled with the name, attached by a solid line to the component
- In UML 1.x were modeled using a dashed arrow

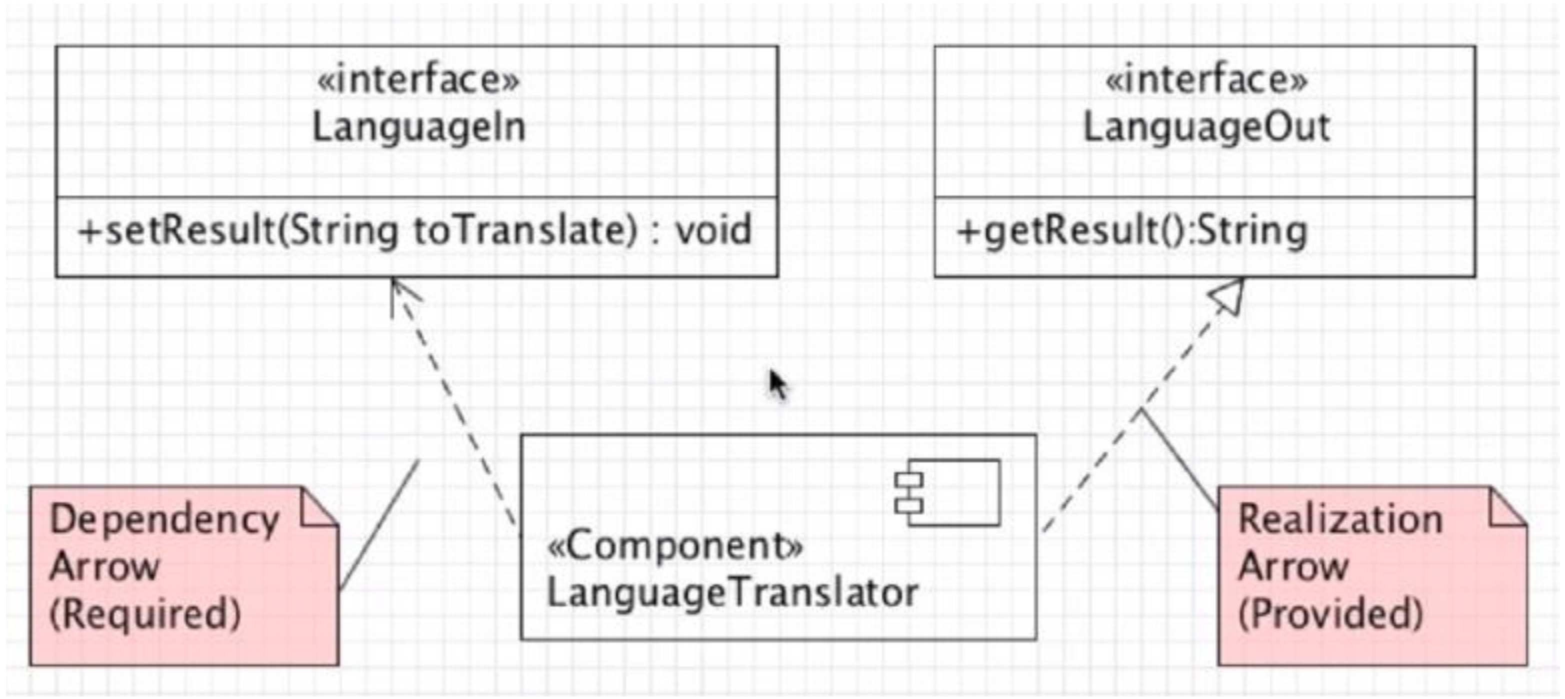


INTERFACE EXAMPLE

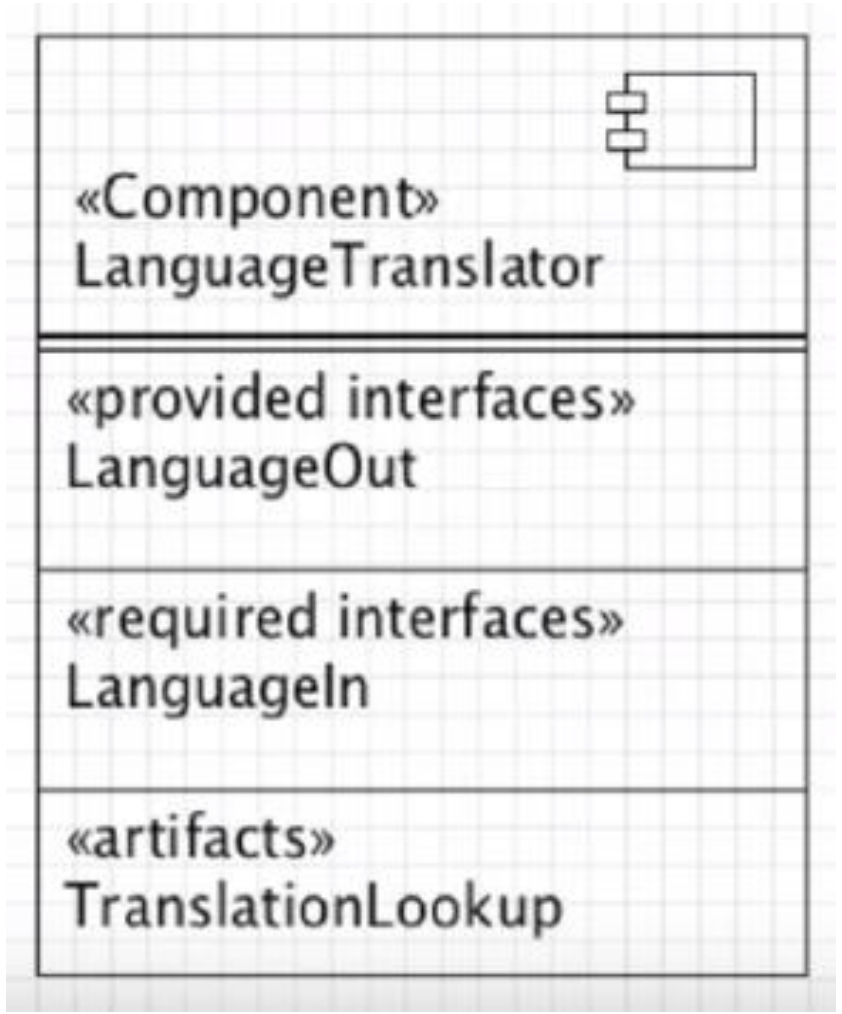


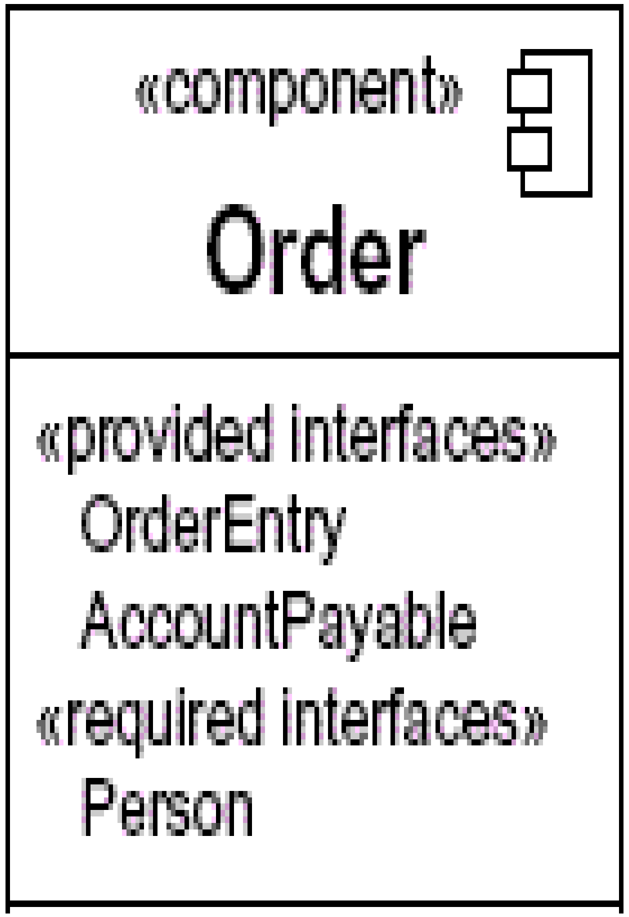


WAYS TO REPRESENT PROVIDED AND REQUIRED INTERFACE


«interface»
LanguageIn
+setResult(String toTranslate) : void
«interface»
LanguageOut
+getResult():String
Dependency Arrow (Required)
«Component»
LanguageTranslator
Realization Arrow (Provided)

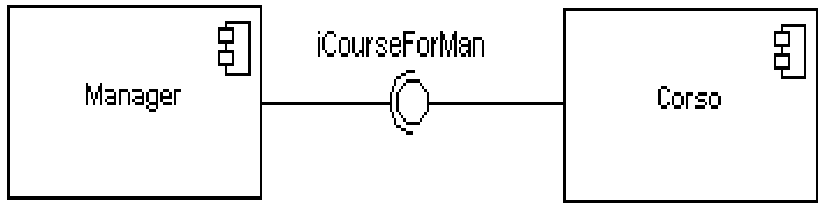
WAYS TO REPRESENT PROVIDED AND REQUIRED INTERFACE



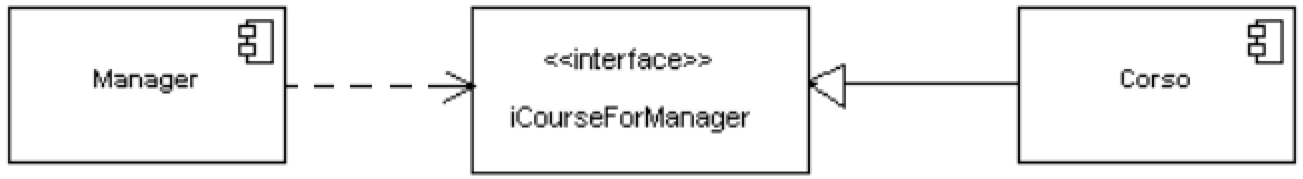


INTERFACES

- Where two components/classes provide and require the same interface, these two notations may be combined.

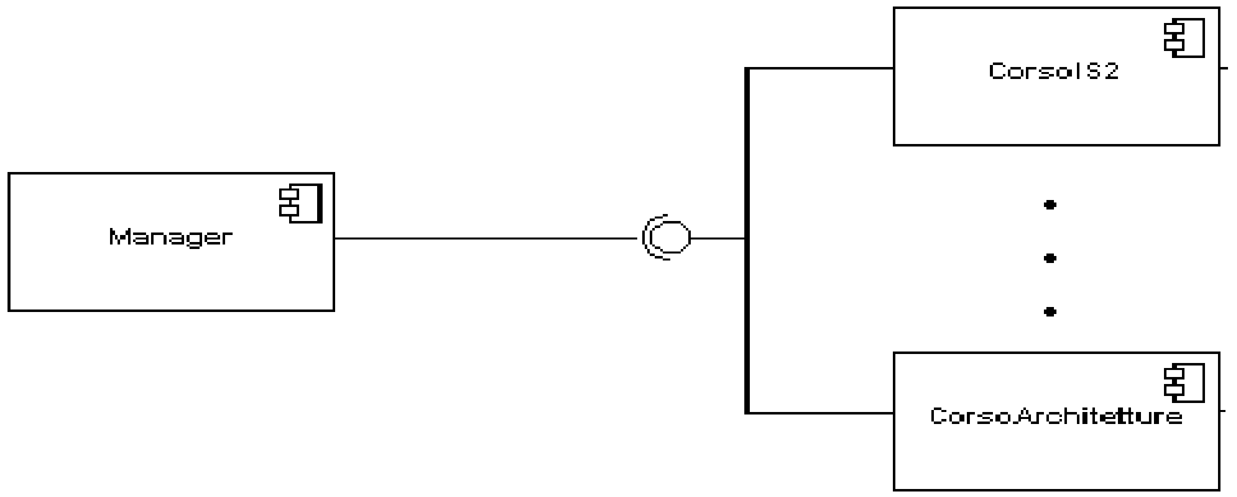


- The ball-and-socket notation hint at that interface in question serves to mediate interactions between the two components
- If an interface is shown using the rectangle symbol, we can use an alternative notation, using dependency arrows

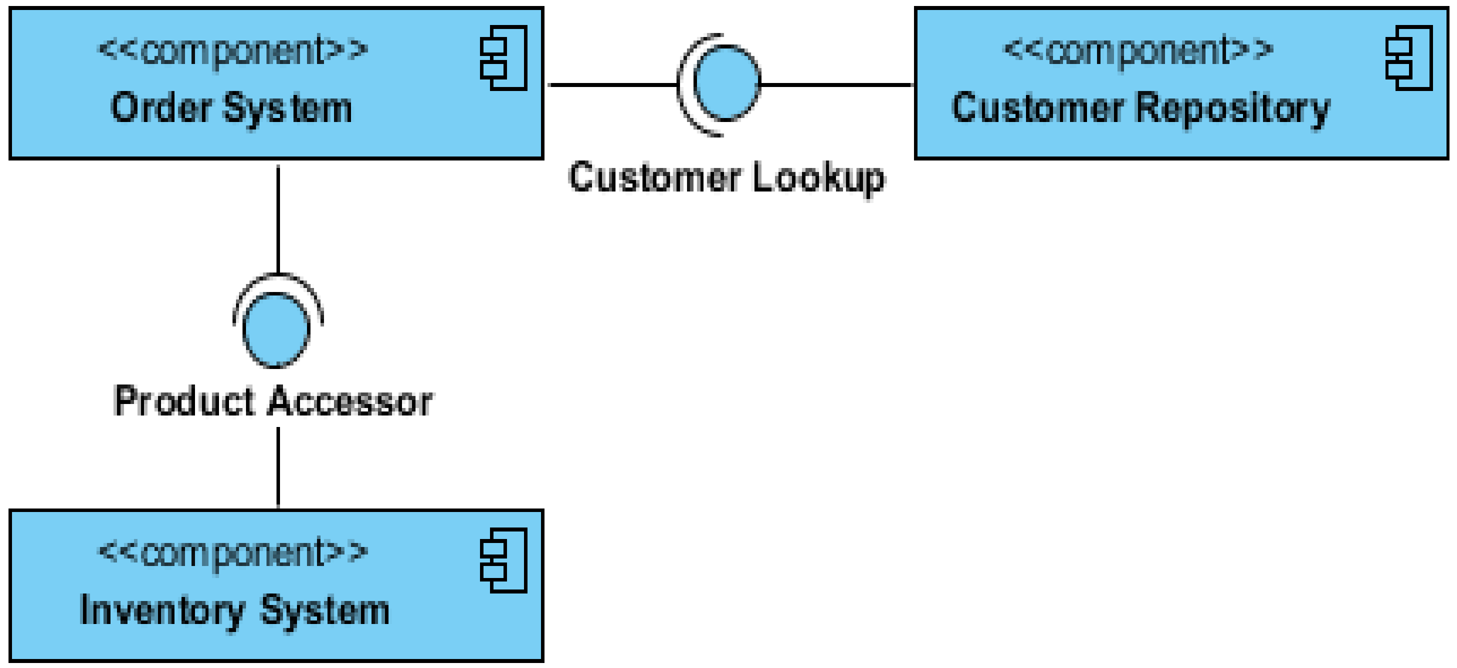


INTERFACES

- In a system context where there are multiple components that require or provide a particular interface, a notation abstraction can be used that combines by joining the interfaces.

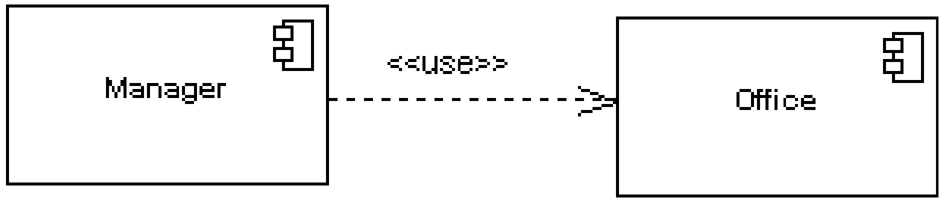


EXAMPLE


<<component>>
Order System
Customer Lookup
<<component>>
Customer Repository
Product Accessor
<<component>>
Inventory System

DEPENDENCIES

- Components can be connected by usage dependencies.

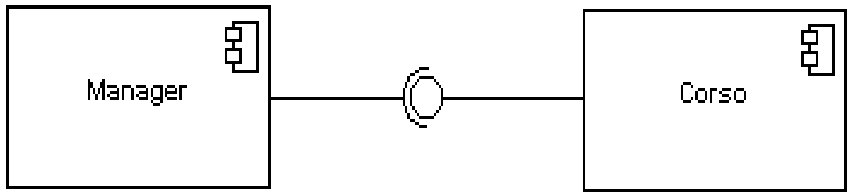


- **Usage Dependency**
- A usage dependency is relationship which one element requires another element for its full implementation
- Is a dependency in which the client requires the presence of the supplier
- Is shown as dashed arrow with a <<use>> keyword
- The arrowhead point from the dependent component to the one of which it is dependent

CONNECTORS

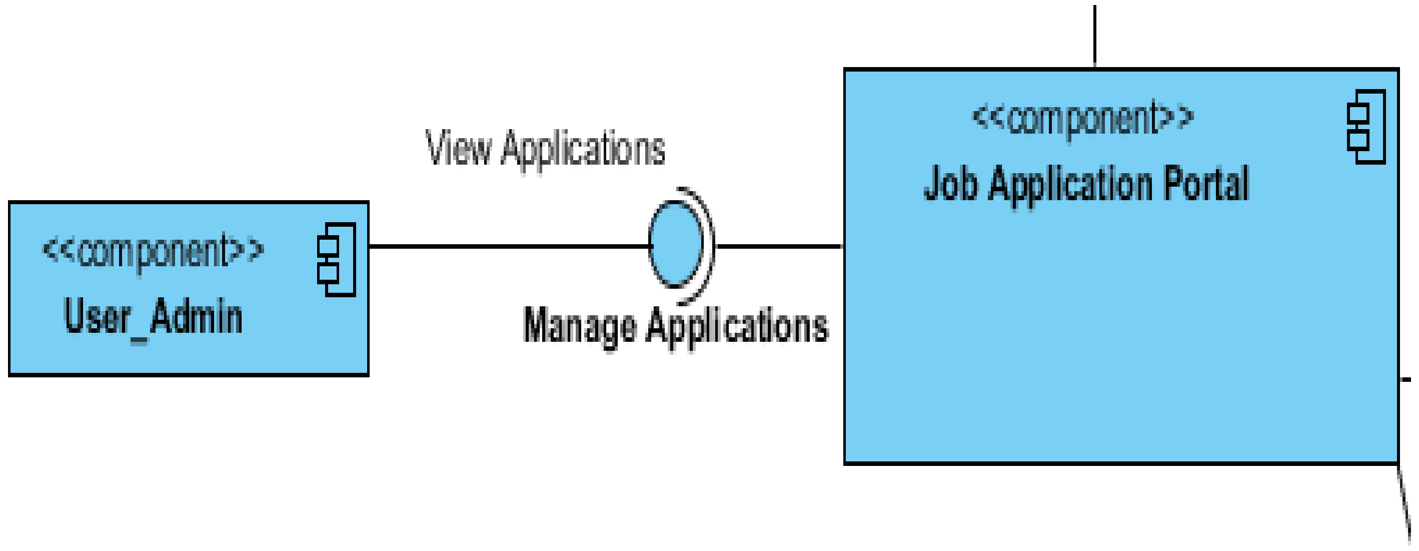
- Two kinds of connectors:
  - Delegation
  - Assembly
- **ASSEMBLY CONNECTOR**
  - A connector between 2 components defines that one component provides the services that another component requires
  - It must only be defined from a required interface to a provided interface
  - An assembly connector is notated by a "ball-and-socket" connection

This notation allows for succint grafical wiring of components

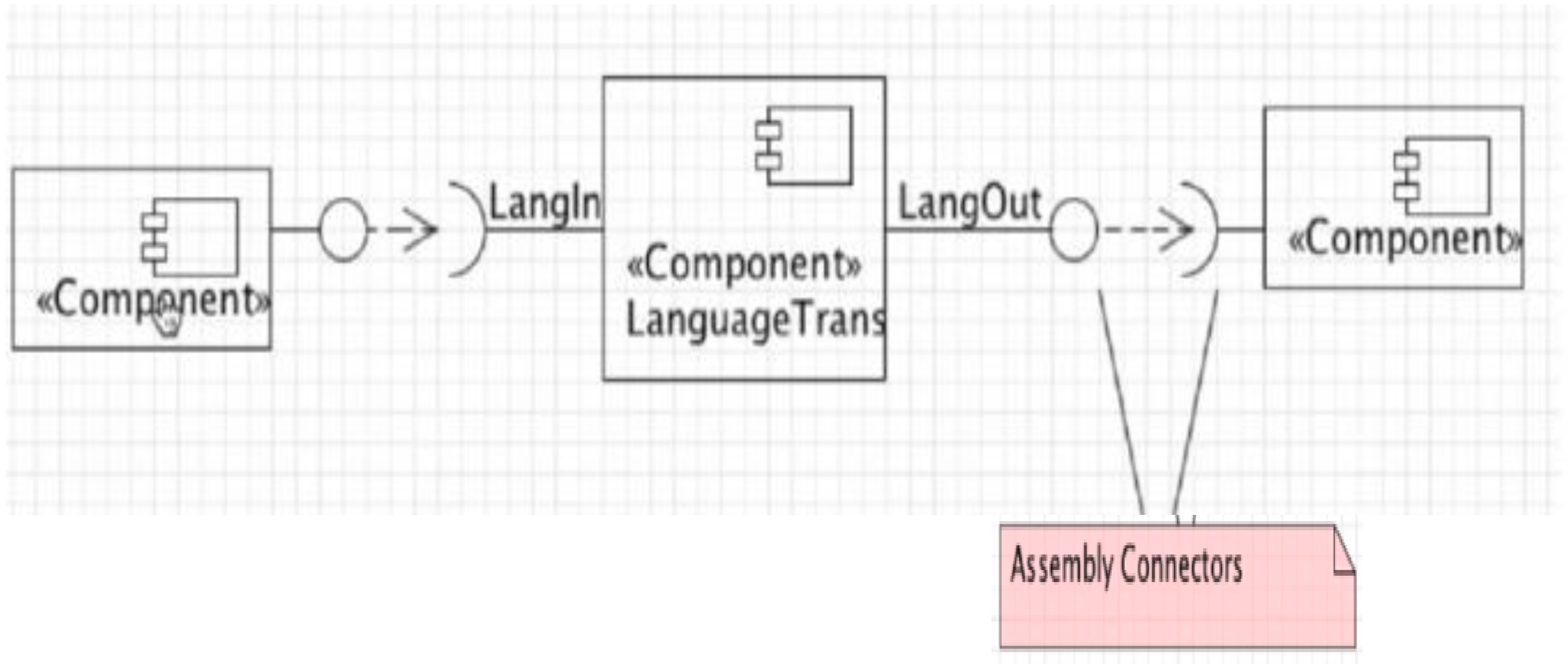


ASSEMBLY CONNECTOR

- The assembly connector bridges a component's required interface (Job Application portal) with the provided interface of another component (User Applicant).

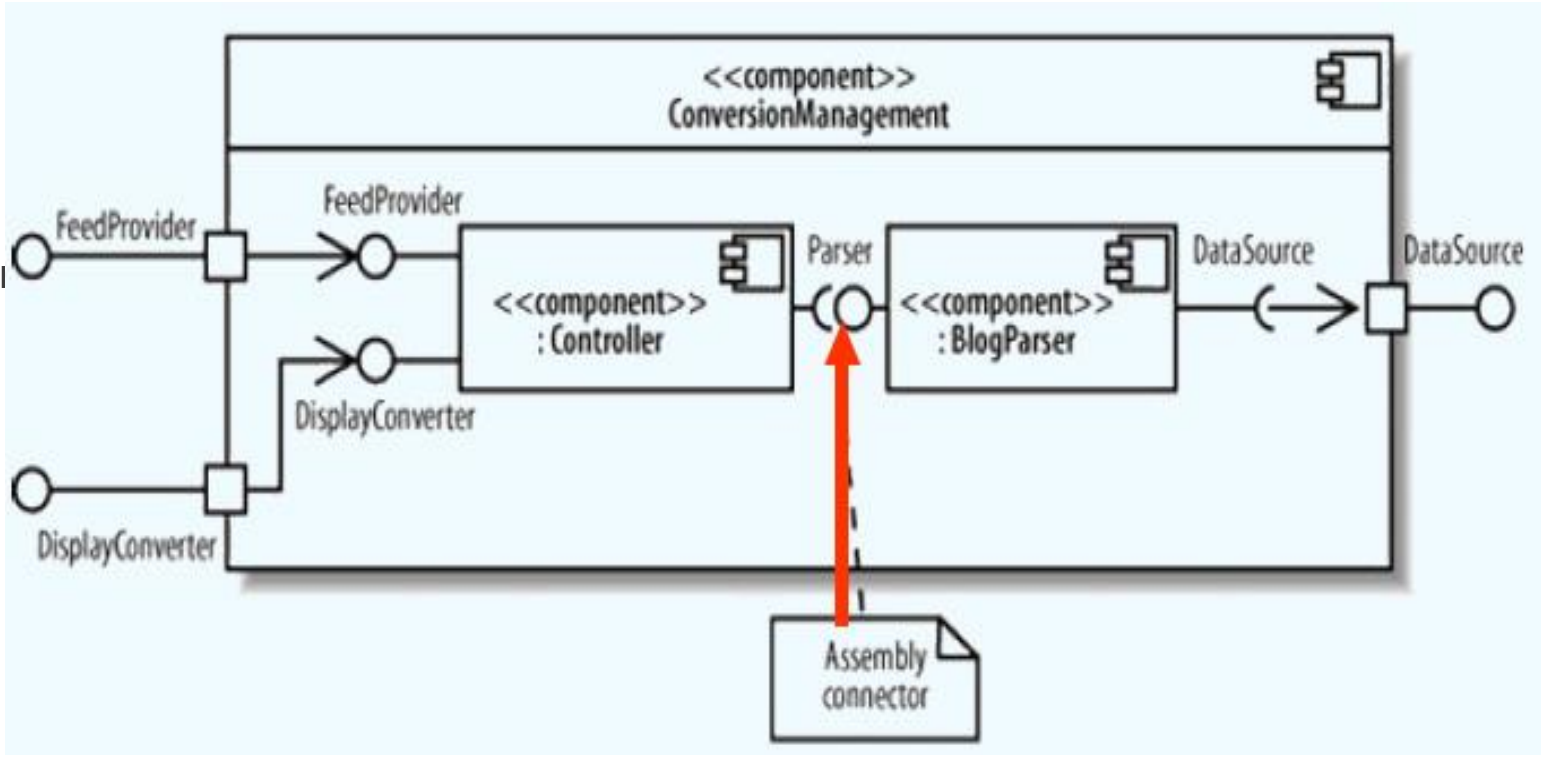


ASSEMBLY CONNECTOR-EXAMPLE



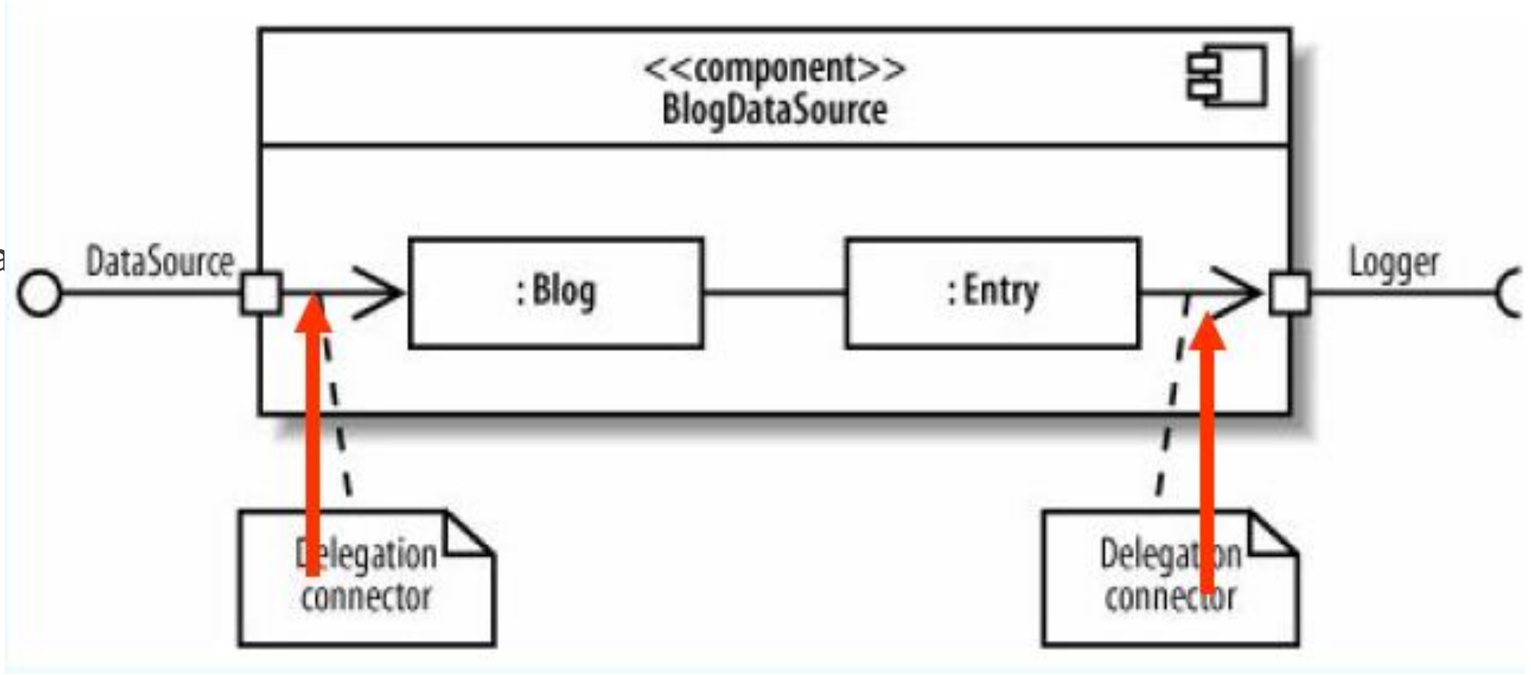
ASSEMBLY CONNECTOR

- Used to show co



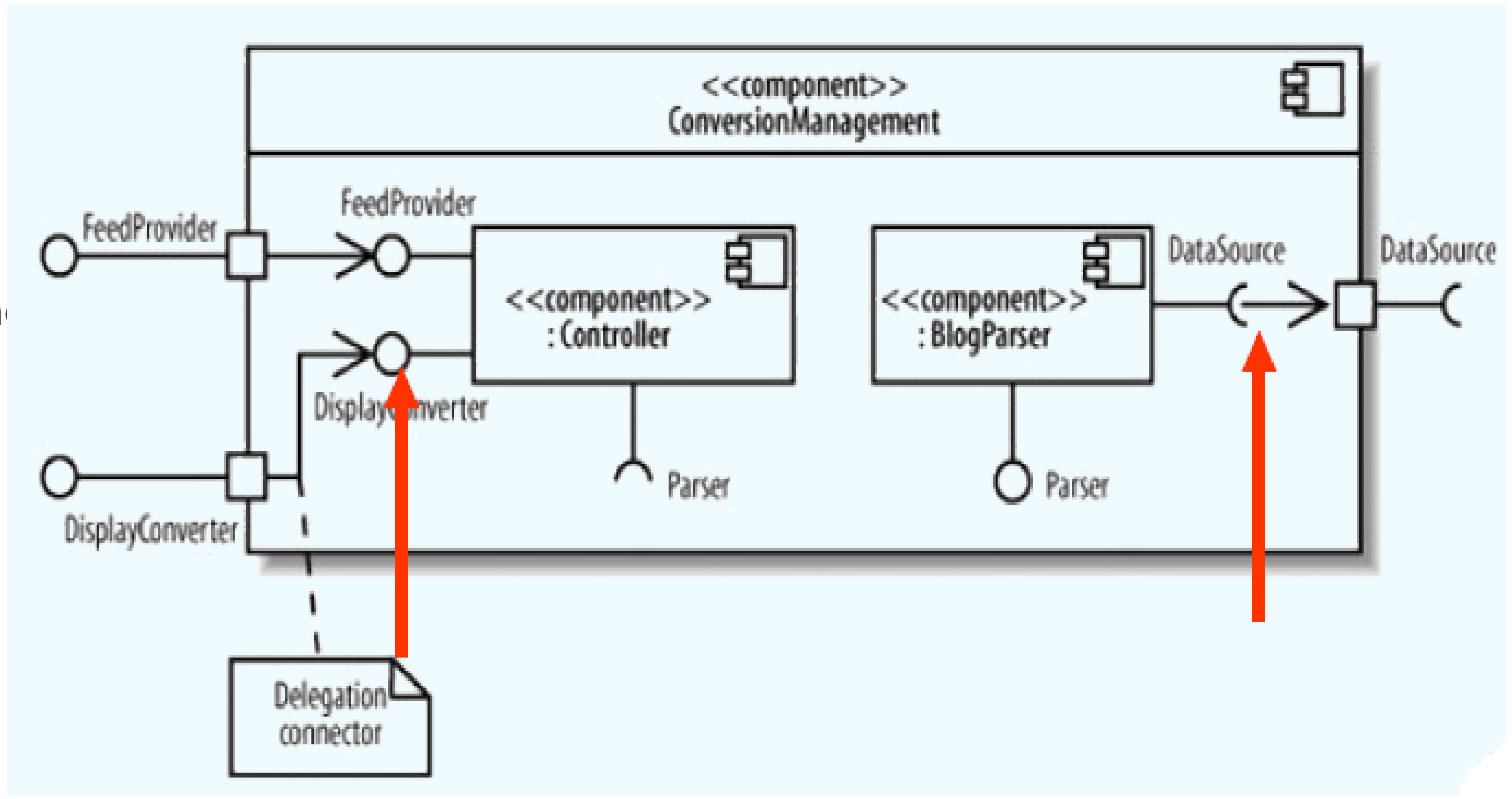
DELEGATION CONNECTOR

- Used to show tha



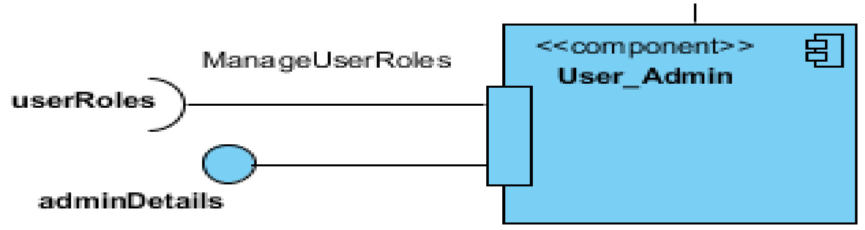
DELEGATION CONNECTOR

- Delegation conn

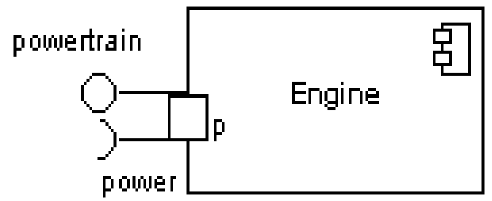


PORT

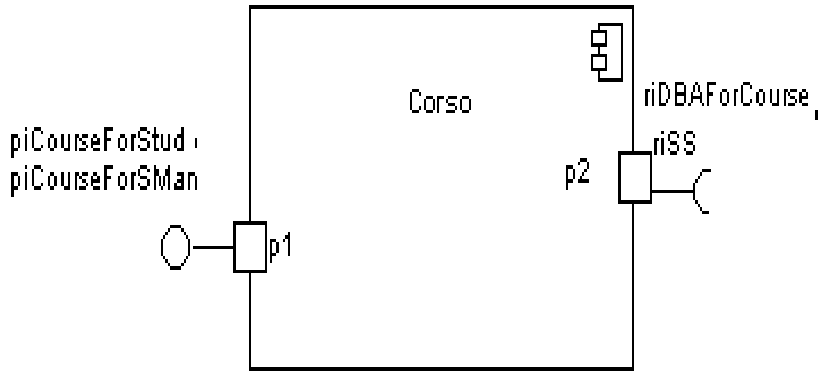
- Specifies a distinct interaction point between that component and its environment –
- Between that component and its internal parts – Is shown as a small square symbol –
- Ports can be named, and the name is placed near the square symbol – Is associated with the interfaces
- Library Services class has port searchPort.



- Ports can support unidirectional communication or bi-directional communication



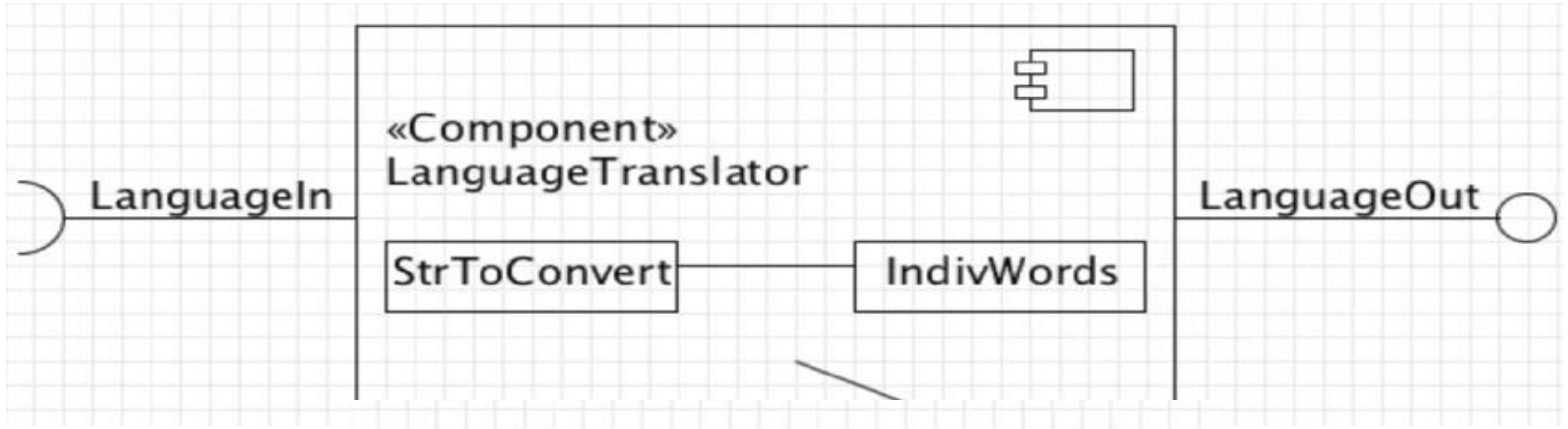
- If there are multiple interfaces associated with a port, these interfaces may be listed with the interface icon, separated by a commas

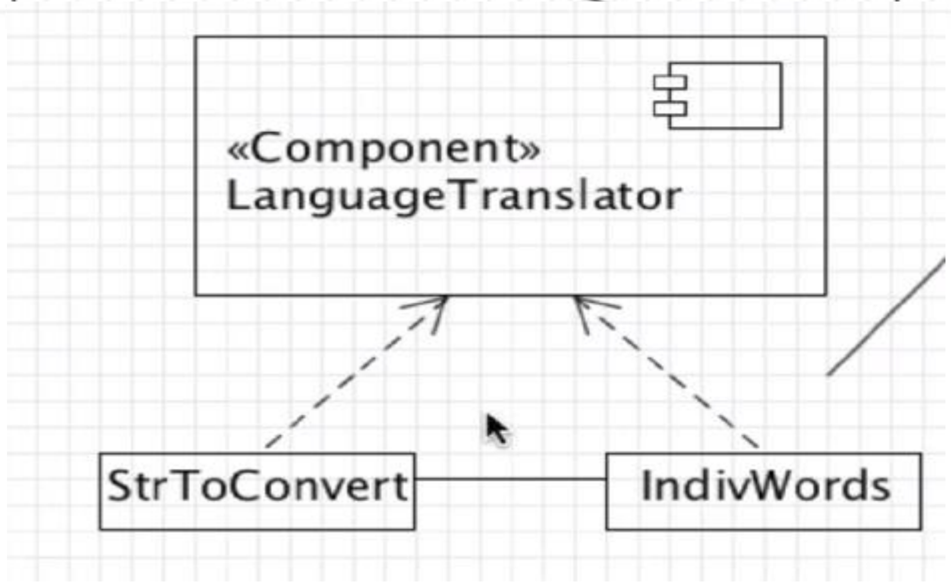


INTERNAL REALIZATION

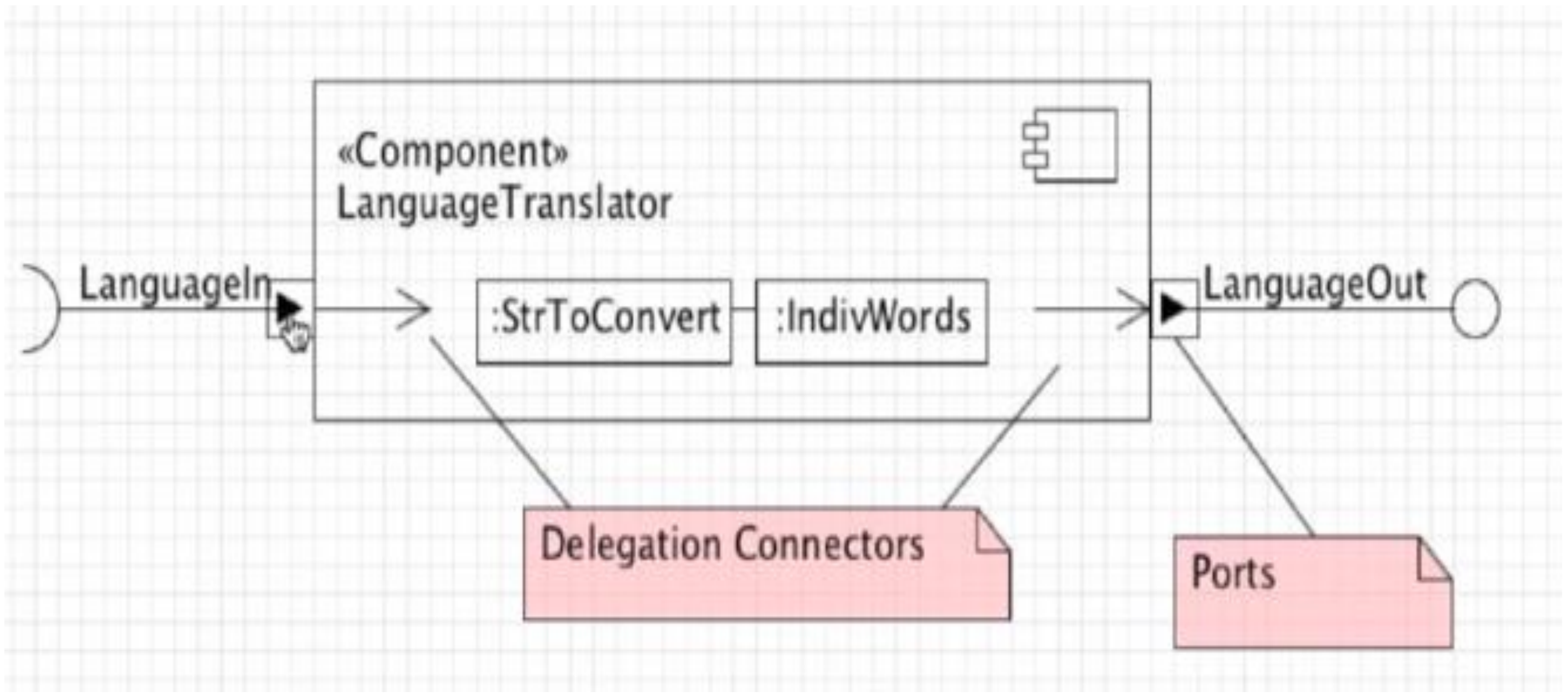
- **A component often contains and uses other classes to implement its functionality.**
- **These classes realize the component**

INTERNAL REALIZATION

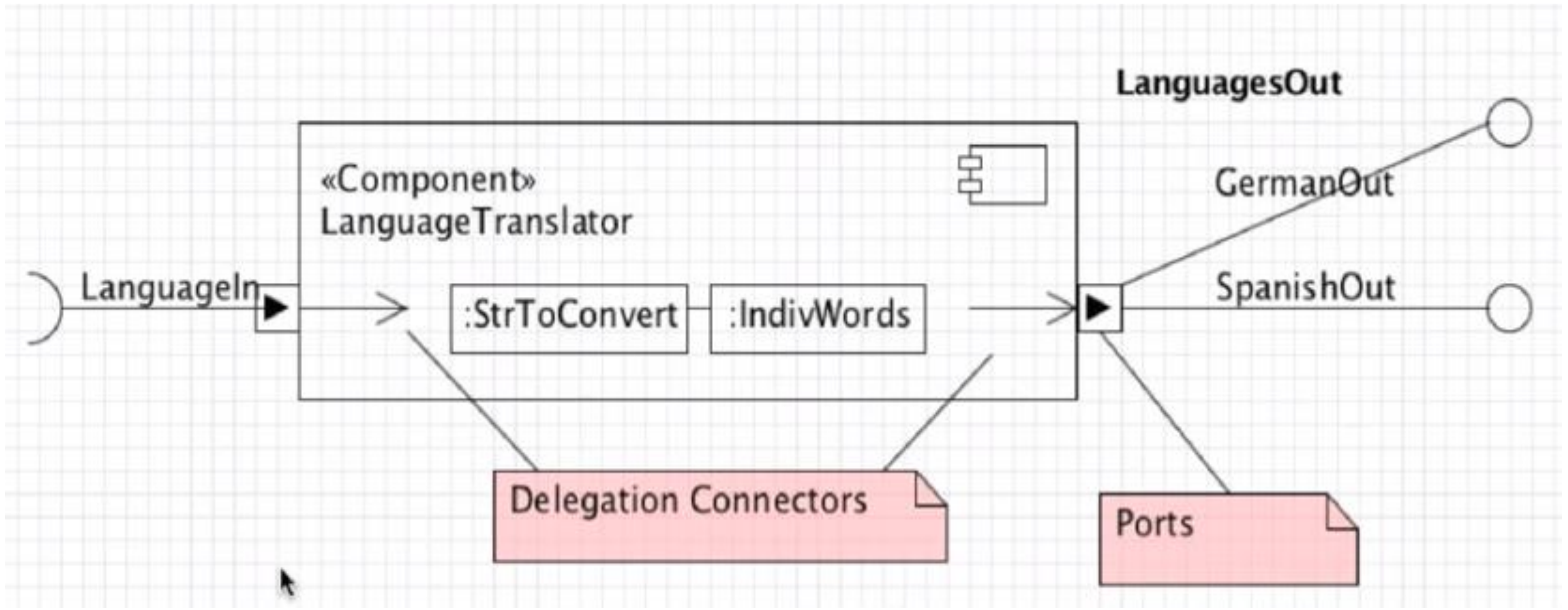


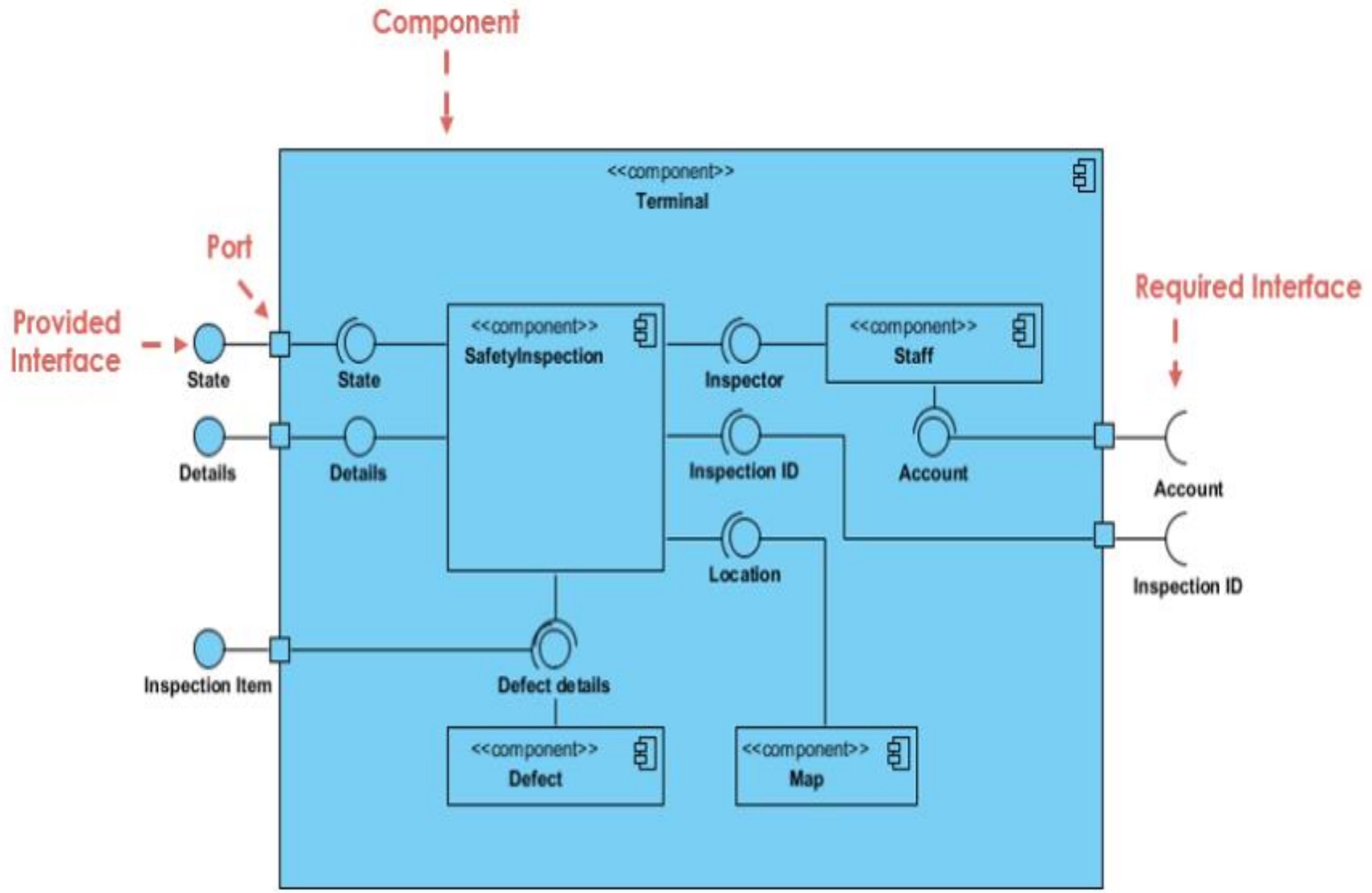


PORTS, DELEGATION CONNECTORS AND INTERNAL REALIZATION



PORTS, DELEGATION CONNECTORS AND INTERNAL REALIZATION




Component
<<component>>
Terminal
Port
Provided Interface
Required Interface
State
State
<<component>>
SafetyInspection
Inspector
<<component>>
Staff
Details
Details
Inspection ID
Account
Account
Location
Inspection ID
Inspection Item
Defect details
<<component>>
Defect
<<component>>
Map

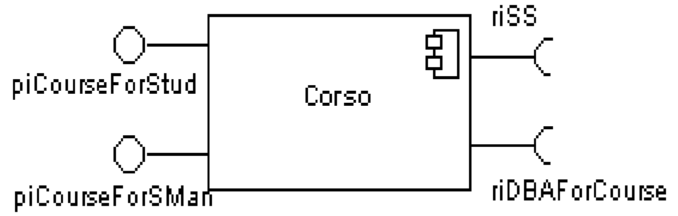
- The example above shows the internal components of a larger component:
- The data (account and inspection ID) flows into the component via the port on the right-hand side and is converted into a format the internal components can use. The interfaces on the right are known as required interfaces, which represents the services the component needed in order to carry out its duty.
- The data then passes to and through several other components via various connections before it is output at the ports on the left. Those interfaces on the left are known as provided interface, which represents the services to deliver by the exhibiting component.
- It is important to note that the internal components are surrounded by a large 'box' which can be the overall system itself (in which case there would not be a component symbol in the top right corner) or a subsystem or component of the overall system (in this case the 'box' is a component itself).

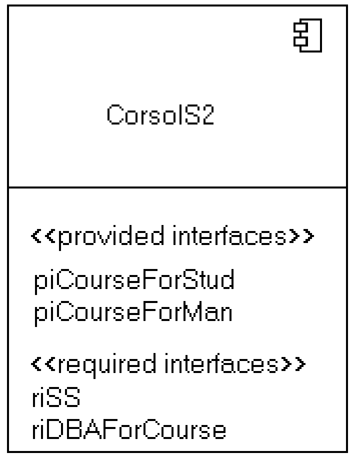
VIEWS OF A COMPONENT

- A component have an
  - external view and
  - an internal view

EXTERNAL VIEW

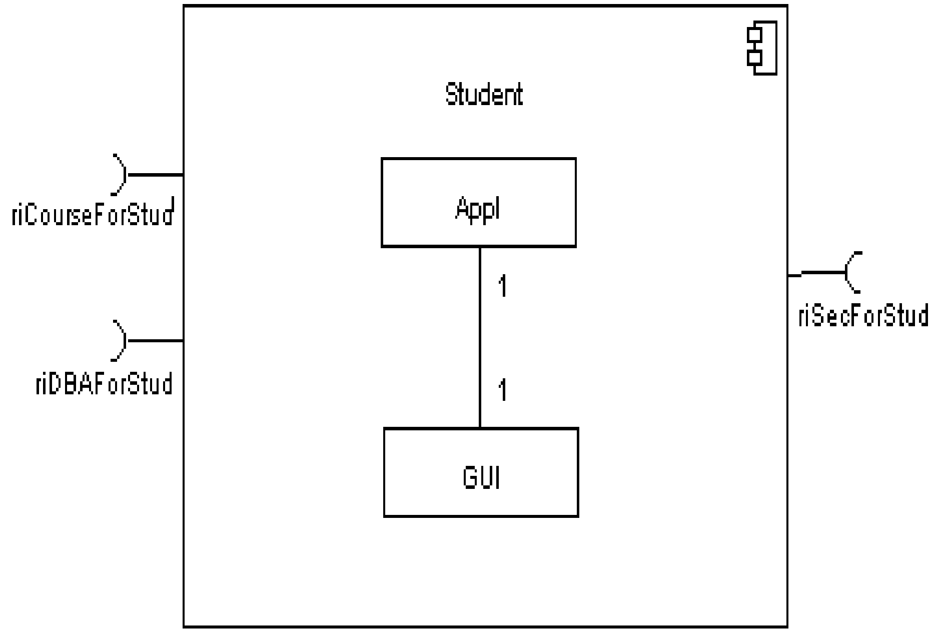
- An external view (or black box view) shows publicly visible properties and operations
- An external view of a component is by means of interface symbols sticking out of the component box
- The interface can be listed in the compartment of a component box

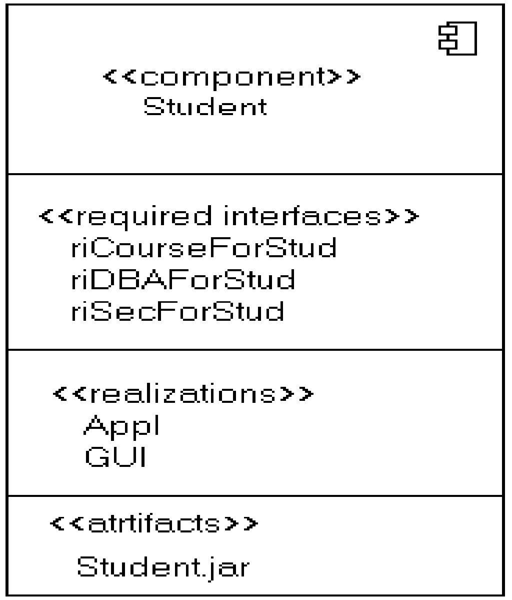




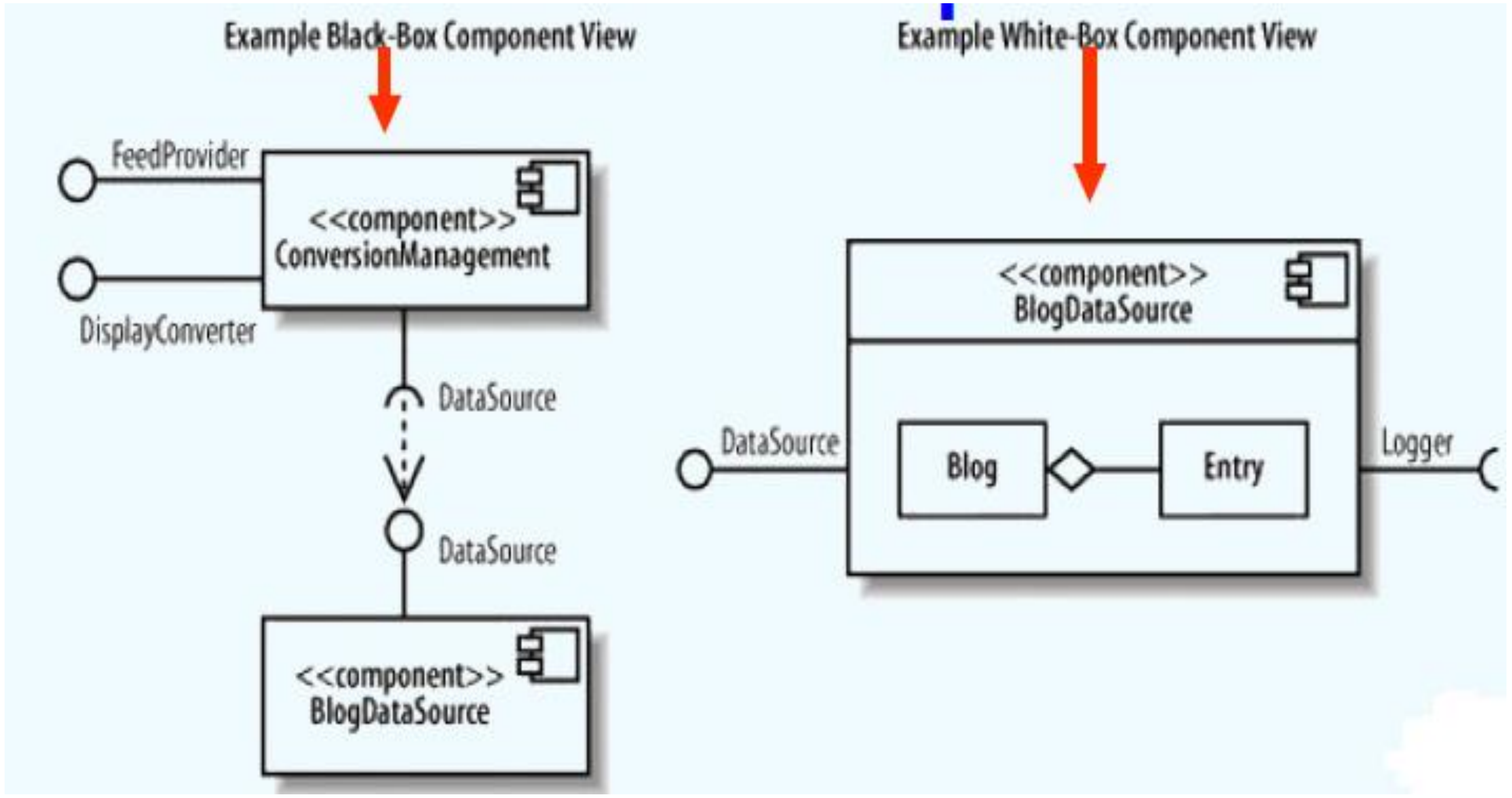
INTERNAL VIEW

- An internal, or white box view of a component is where the realizing classes/components are nested within the component shape
- The internal class that realize the behavior of a component may be displayed in an additional compartment
- Compartments can also be used to display parts, connectors or implementation artifacts
- An artifact is the specification of a physical piece of information

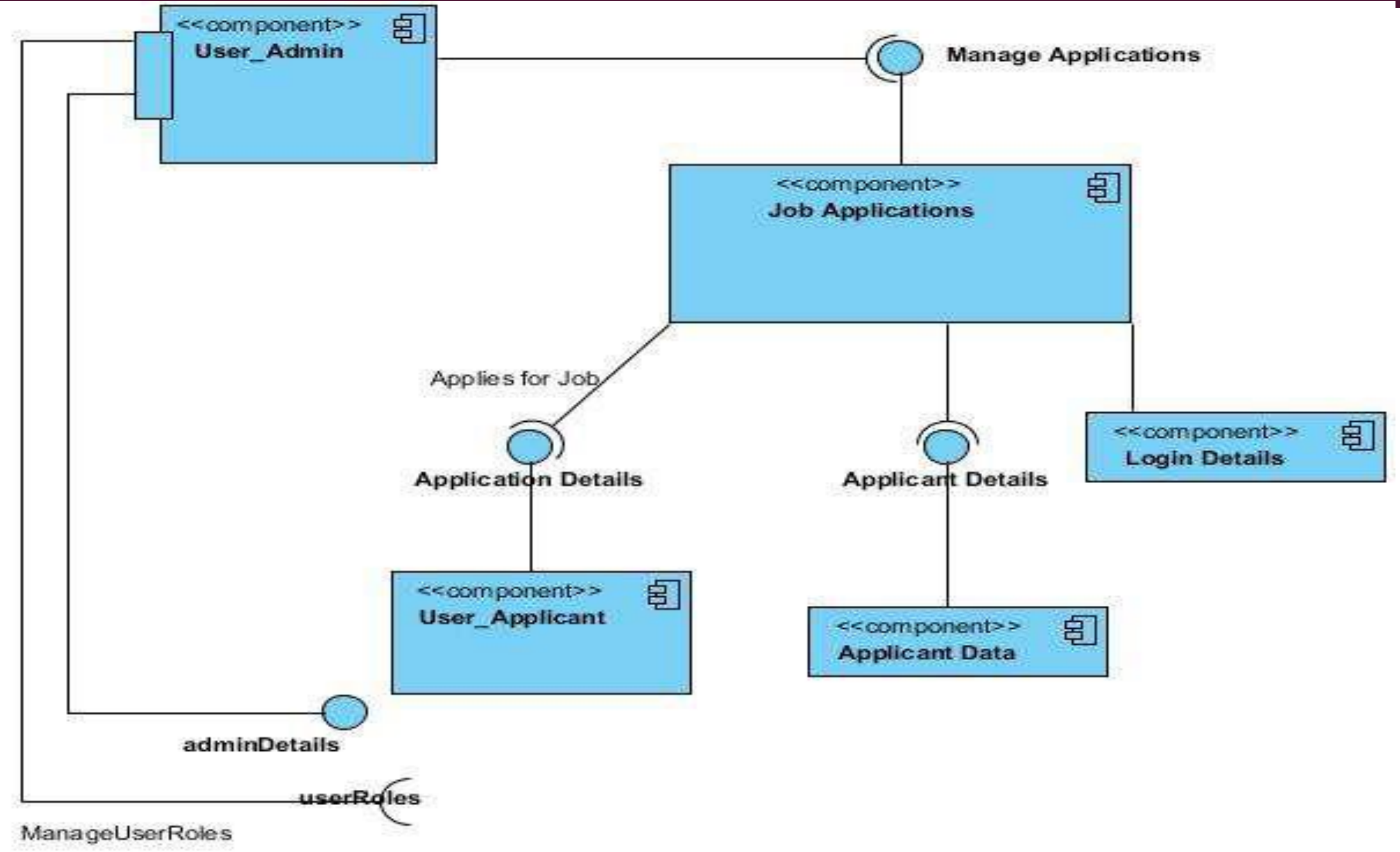




BLACK-BOX AND WHITE-BOX VIEWS



SAMPLE COMPONENT DIAGRAM

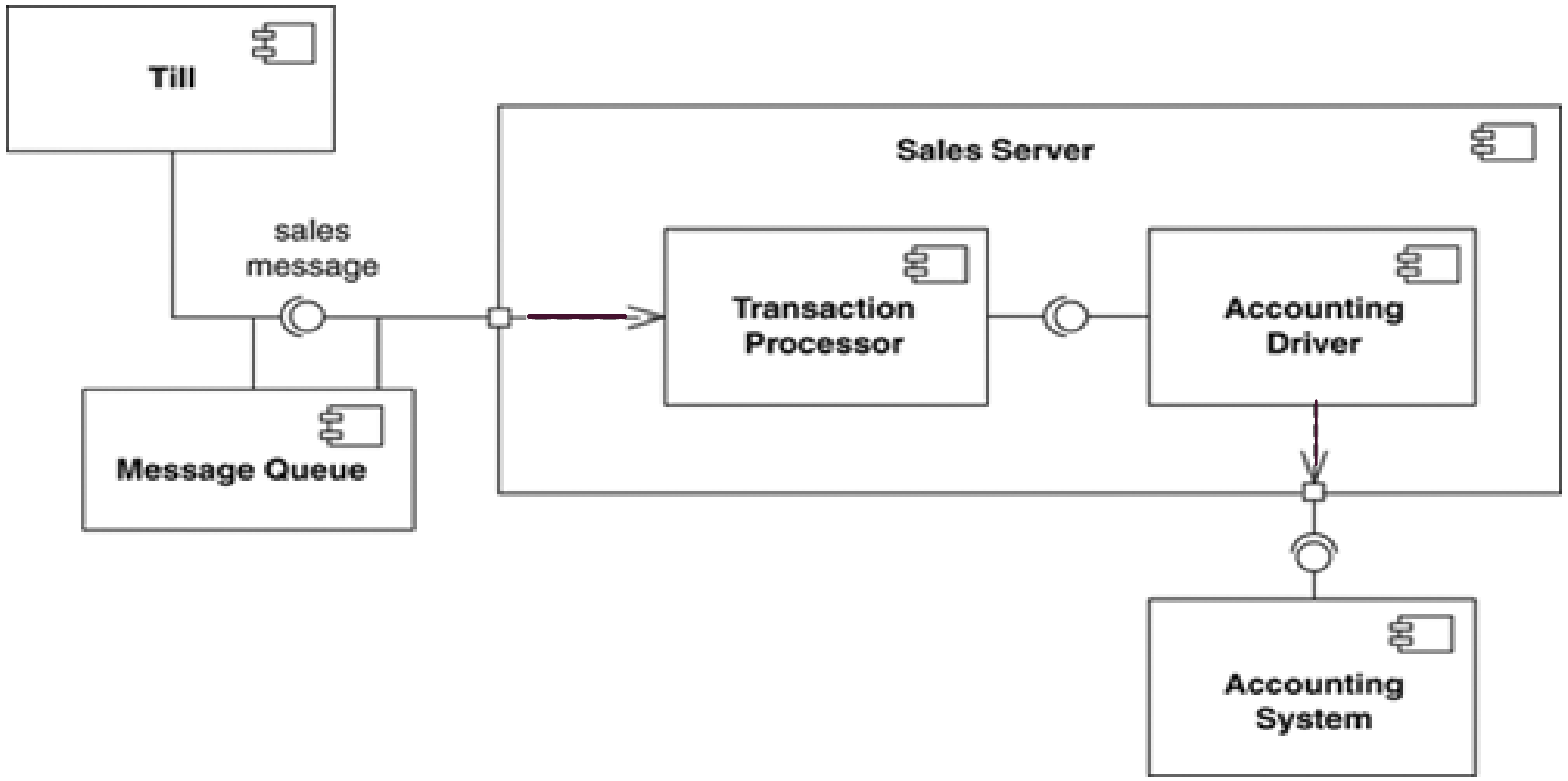


COMPONENT DIAGRAM GUIDELINES

- **Use Descriptive Names for Architectural Components**
  - Use Environment-Specific Naming Conventions for Detailed Design Components
  - Apply Textual Stereotypes to Components Consistently
- **Interfaces**
  - Prefer Lollipop Notation To Indicate Realization of Interfaces By Components
  - Prefer the Left-Hand Side of A Component for Interface Lollipops
  - Show Only Relevant Interfaces
- **Dependencies and Inheritance**
  - Model Dependencies From Left To Right
  - Place Child Components Below Parent Components
  - Components Should Only Depend on Interfaces

EXAMPLE COMPONENT DIAGRAM.

- A sales till can connect to a sales server component, using a sales message interface. Because the network is unreliable, a message queue component is set up so the till can talk to the server when the network is up and talk to a queue when the network is down; the queue will then talk to the server when the network becomes available. As a result, the message queue both supplies the sales message interface to talk with the till and requires that interface to talk with the server. The server is broken down into two major components. The transaction processor realizes the sales message interface, and the accounting driver. Accounting driver talks to the accounting system component for getting the details about the bill information. Accounting driver provides this information to transaction processor for managing the sale.


Till
sales message
Message Queue
Sales Server
Transaction Processor
Accounting Driver
Accounting System

That is all