EAST WEST UNIVERSITY

CSE347

Information System Analysis and Design

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

Topic: 9

System Architecture
Component Design

INTRODUCTION

- **Behavior diagrams**
  - A type of diagram that depicts behavior of a system

    This includes activity, state machine, and use case diagrams, interaction diagrams
- **Interaction diagrams**
  - A subset of behavior diagrams which emphasize object interactions. This includes collaboration, activity, sequence diagrams
- **Structure diagrams**
  - A type of diagram that depicts the elements of a specification that are irrespective of time. This includes class, composite structure, component, deployment

UML components diagrams are **structure diagrams**

Component Diagram

- A component is an encapsulated, reusable, and replaceable part of your software
- Reducing and defying coupling between software components
- Reusing existing components
- The ability to identify software components (which are encapsulated, reusable and replaceable) supports development strategies that use, e.g., COTS (Commercial- Off-The-Shelf) components.

Component Diagram

- A Component is a self-contained unit that encapsulates the state and behavior of a number of Classifiers.
  - In UML, a Classifier represents a classification of instances according to their Features.
  - ❖ For instance: a category of entities in the domain
  - ❖ A classifier has attributes

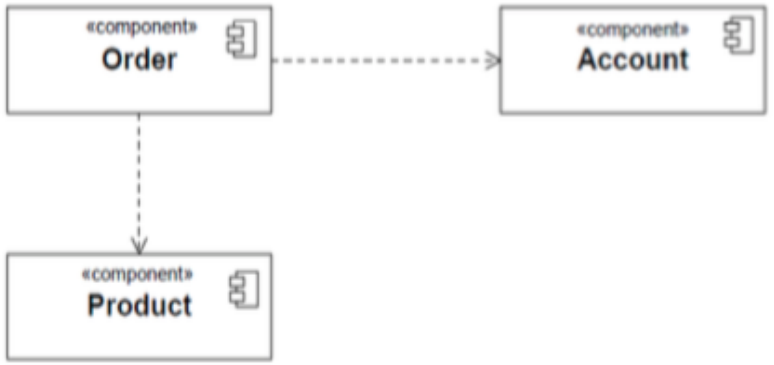


Figure 11.39 Example of an overview diagram showing Components and their general Dependencies

Component diagram

- **Component Diagrams** can show how subsystems relate and which interfaces are implemented by which component.
- A Component Diagram shows one or more interfaces and their relationships to other components.
- A Component Diagram shows the **dependencies** among software components, including source code, binary code and executable components.
- Some components exist at compile time, some exist at link time, and some exist at run time; some exist at more that one time.

Component Modelling

1. Find components and dependencies
2. Identify and level subcomponents
3. Clarify and make explicit the interfaces between components

When to use component diagrams

- Use component diagrams when you are dividing your system into components and want to show their interrelationships through interfaces or the breakdown of components into a lower-level structure.

How to produce component diagrams

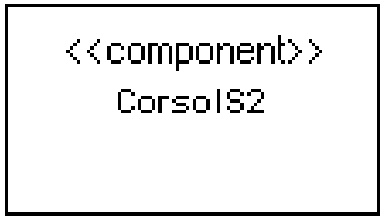
- Decide on the purpose of the diagram
- Add components to the diagram, grouping them within other components if appropriate
- Add other elements to the diagram, such as classes, objects and interfaces
- Add the dependencies between the elements of the diagram

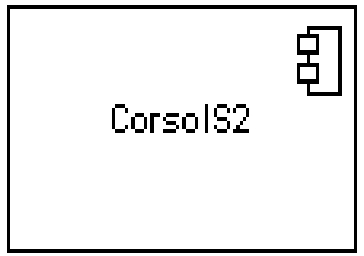
Component notation

- A Component is a physical piece of a system, such as a compiled object file, piece of source code, shared library or Enterprise Java Bean (EJB).
- Note that UML 2.0 uses a new notation for a component. Previous UML versions use the component icon as the main shape.

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. There are a number of standard stereotypes. ex:<<entity>>,<<subsystem>>





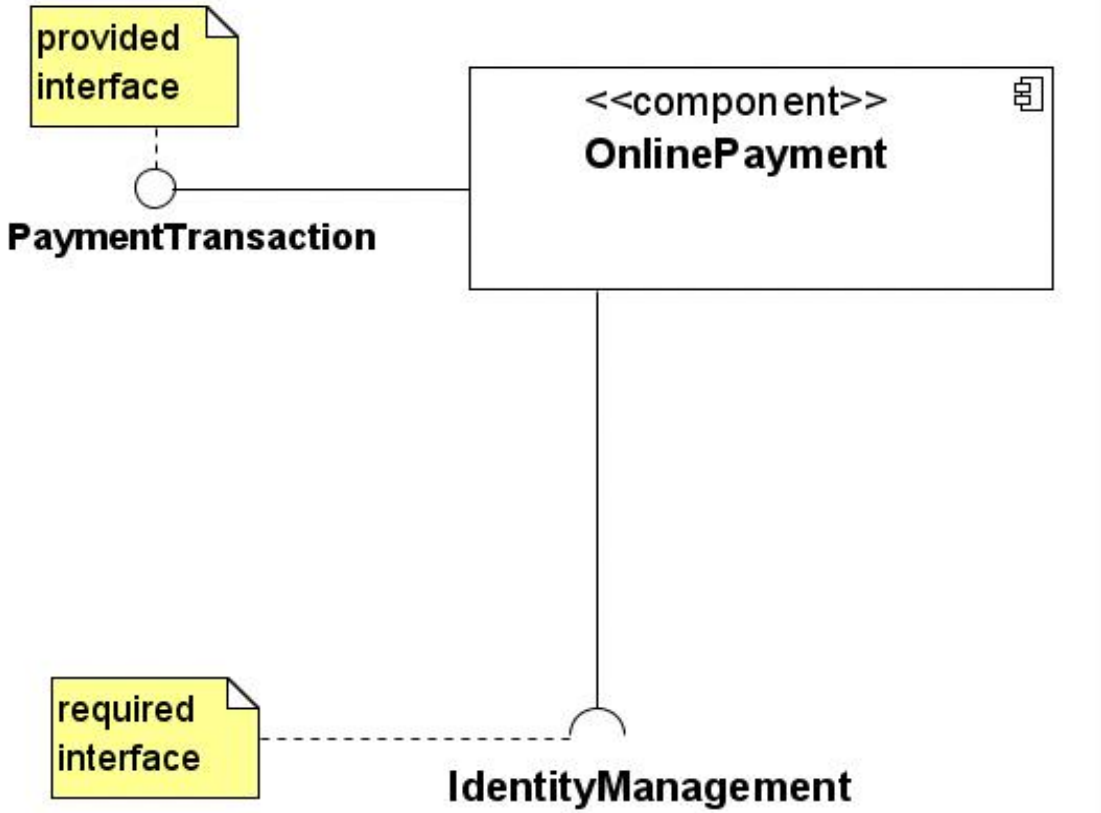
Component ELEMENTS

- A component can have
  - **Interfaces**
    - An interface represents a declaration of a set of operations and obligations
  - **Usage dependencies**
    - A usage dependency is a relationship where one element requires another element for its full implementation
  - **Ports**
    - Port represents an interaction point between a component and its environment
  - **Connectors**
    - Connect two components
    - Connect the external contract of a component to the internal structure

INTERFACE

- A component defines its behaviour in terms of provided and required 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

Component interfaces



Component interfaces

- A provided interface of a component is an interface that the component realizes
- A required interface of a component is an interface that the component needs to function
- The provided and required Interfaces of a Component may be shown by means of ball (lollipop) and socket notation, respectively.

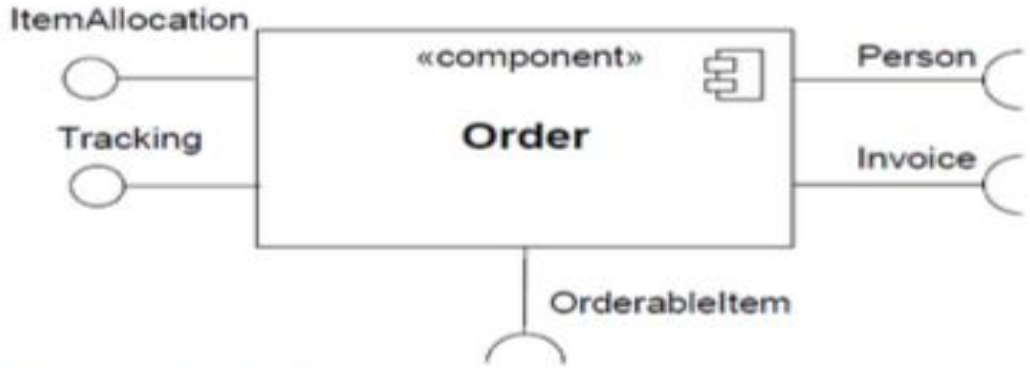


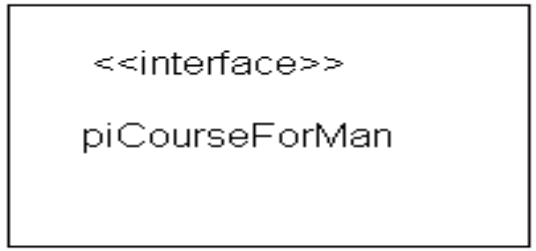
Figure 11.40 A Component with two provided and three required Interfaces

Component interfaces

- Class interfaces have similar notations (definitions).
- Provided interfaces define “a set of public attributes and operations that must be provided by the classes that implement a given interface”.
- Required interfaces define “a set of public attributes and operations that are required by the classes that depend upon a given interface”.
- Java Warnings: Note that these definitions of interfaces differ from the Java definition of interfaces. The Java definition of interfaces does not allow to have attributes, nor hence state.

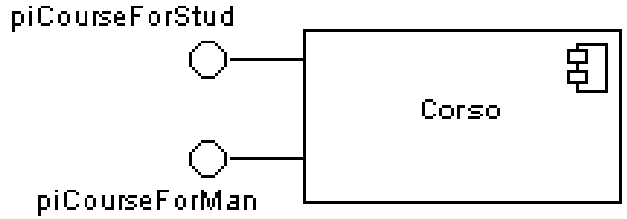
INTERFACE

- 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**

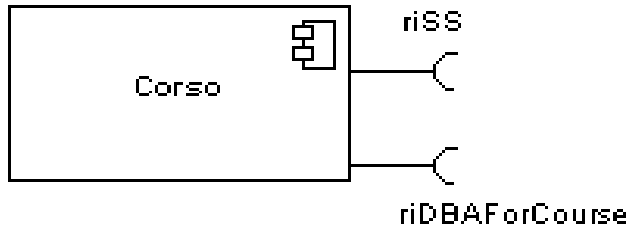


INTERFACE

- A 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

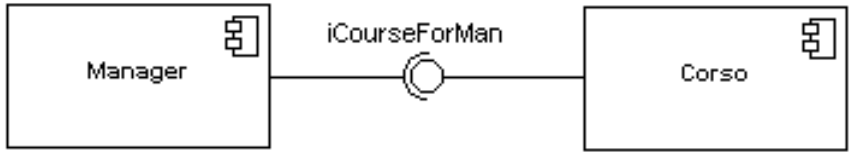


- A required interface
  - Characterize services that the component expects from its environment
  - Is modeled using a socket, labelled with the name, attached by a solid line to the component
  - In UML 1.x were modeled using a dashed arrow



INTERFACE

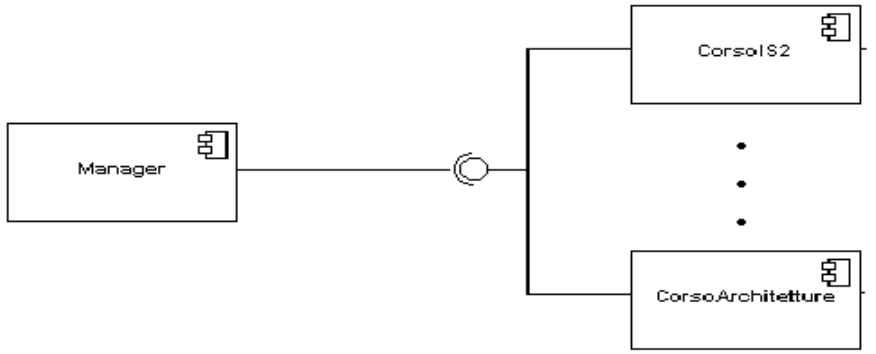
- Where two components/classes provide and require the same interface, these two notations may be combined



  - 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

INTERFACE

- 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

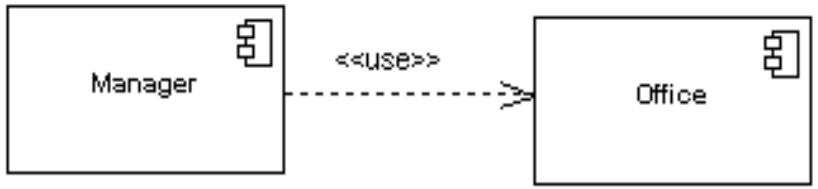


Dependencies

- **Reside Dependencies:** A reside dependency from a component to any UML element indicates that the component is a client of the element, which is considered itself a supplier, and that the element resides in the component.
- **Use Dependencies:** A use dependency from a client component to a supplier component indicates that the client component uses or depends on the supplier component. A use dependency from a client component to a supplier components interface indicates that the client component uses or depends on the interface provided by the supplier component.
- **Deploy Dependency:** A deploy component from a client component to a supplier node indicates that the client components is deployed on the supplier node.

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

Dependencies among components

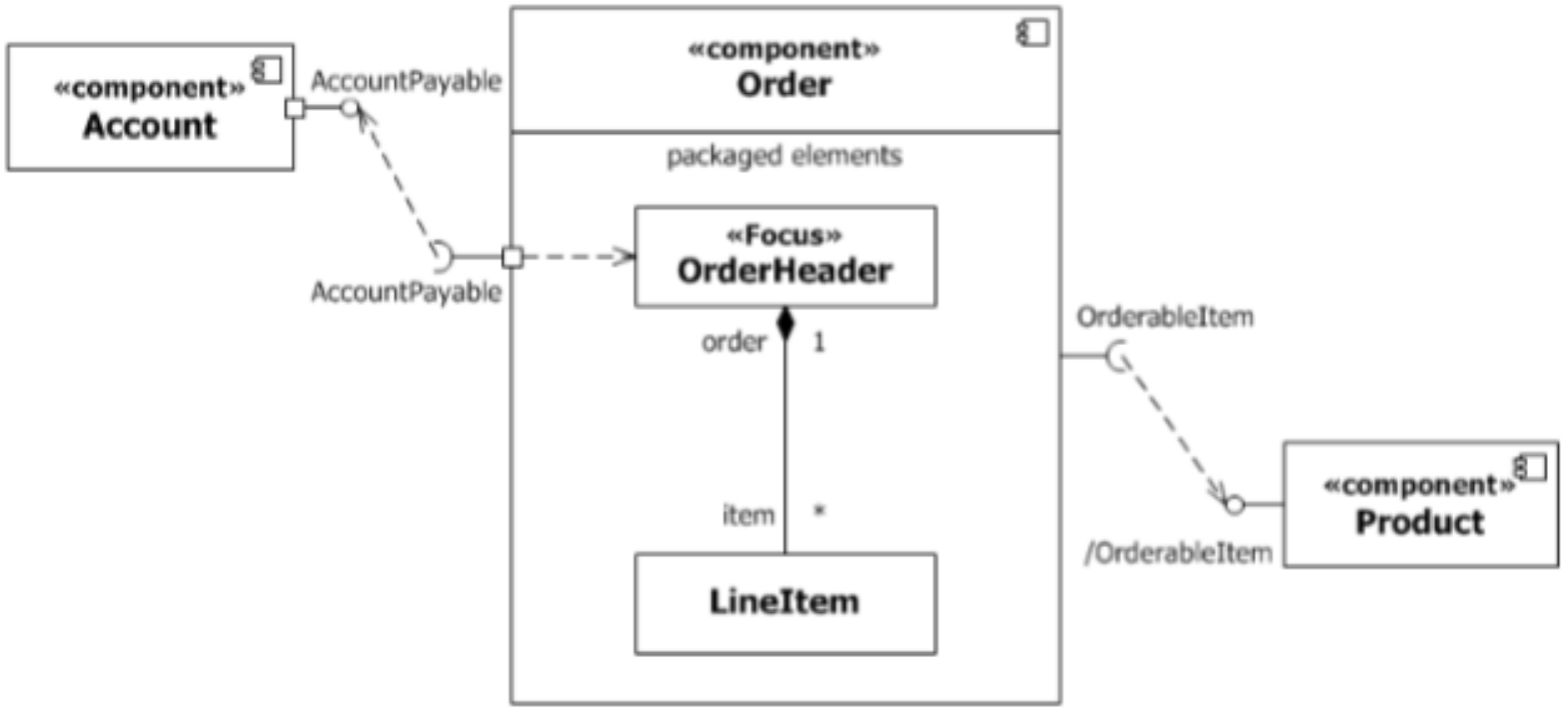
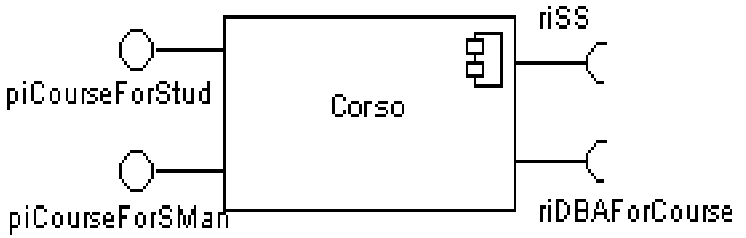
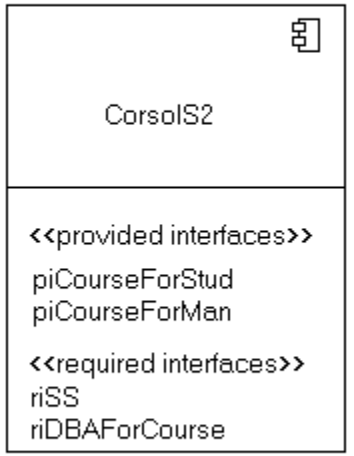


Figure 11.46 Example model of a Component, its provided and required Interfaces, and wiring through Dependencies

EXTERNAL VIEW

- A component has an external view and an internal 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

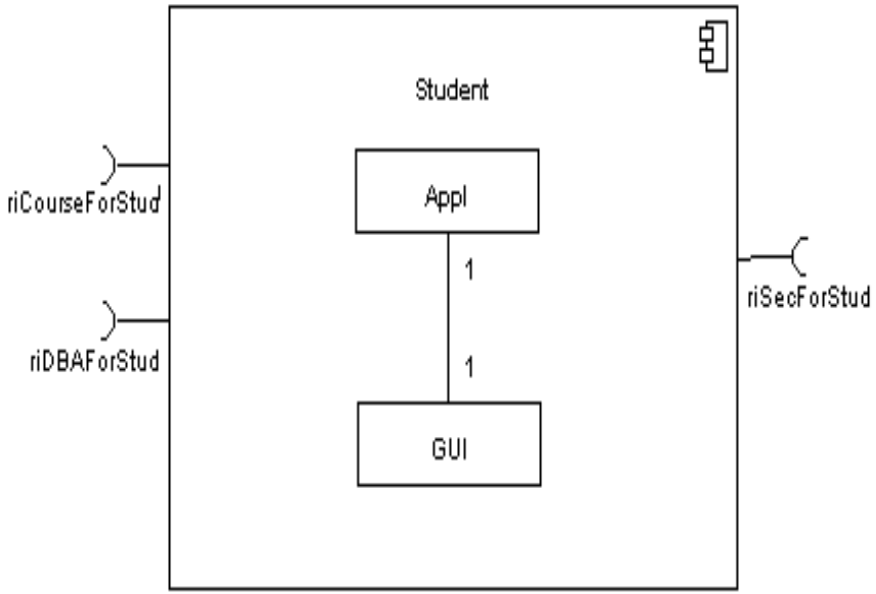




INTERNAL VIEW

- An internal, or white box view of a component is where the realizing classes/components are nested within the component shape

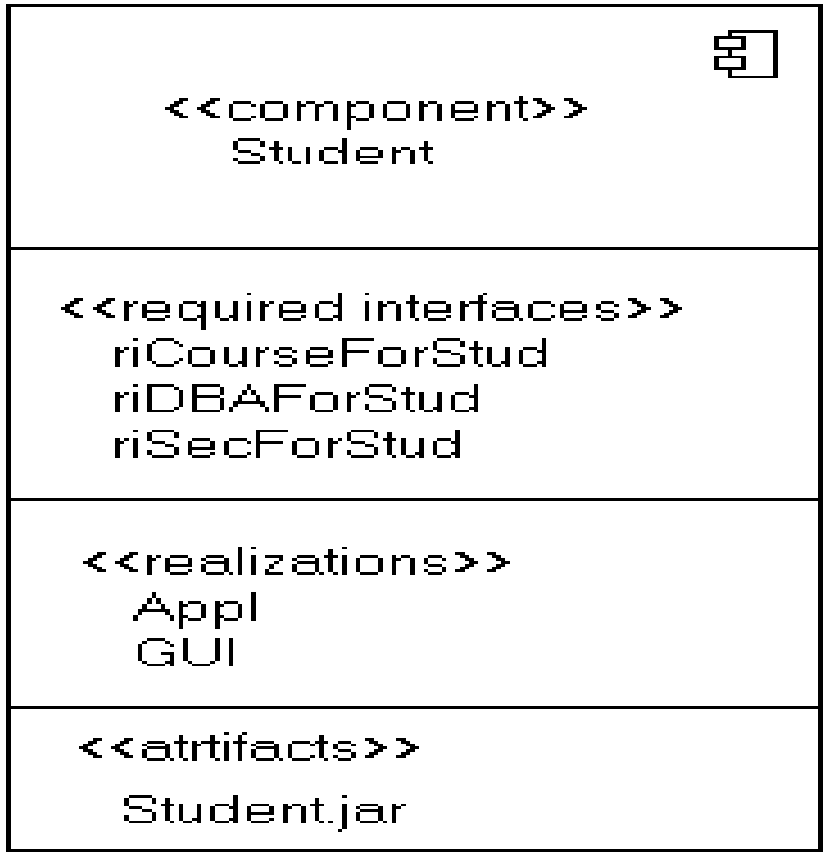
  - Realization is a relationship between two set of model elements
    - One represents a specification
    - The other represent an implementation of the latter



INTERNAL VIEW

- 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 phisycal piece of information



SEMANTICS for an assembly connector

- The semantics for an assembly connector:
  - Are that signals travel along an instance of a connector originating in a required port and delivered to a provided port
  - The interfaces provided and required must be compatible
  - The interface compatibility between provided and required ports that are connected enables an existing component in a system to be replaced

Example

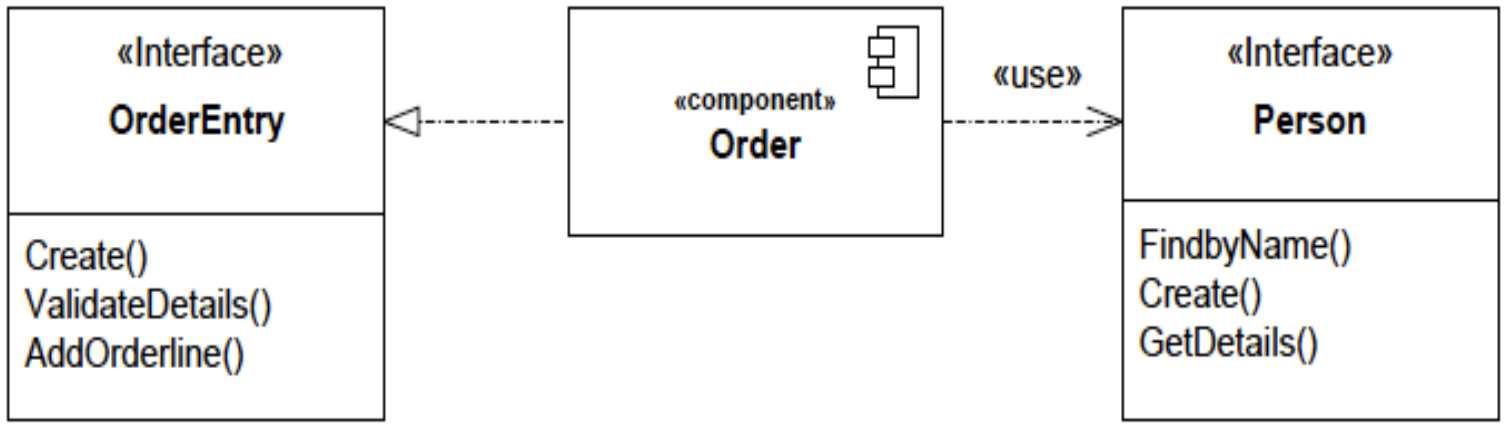


Figure 8.8 - Explicit representation of the provided and required interfaces, allowing interface details such as operation to be displayed (when desired).

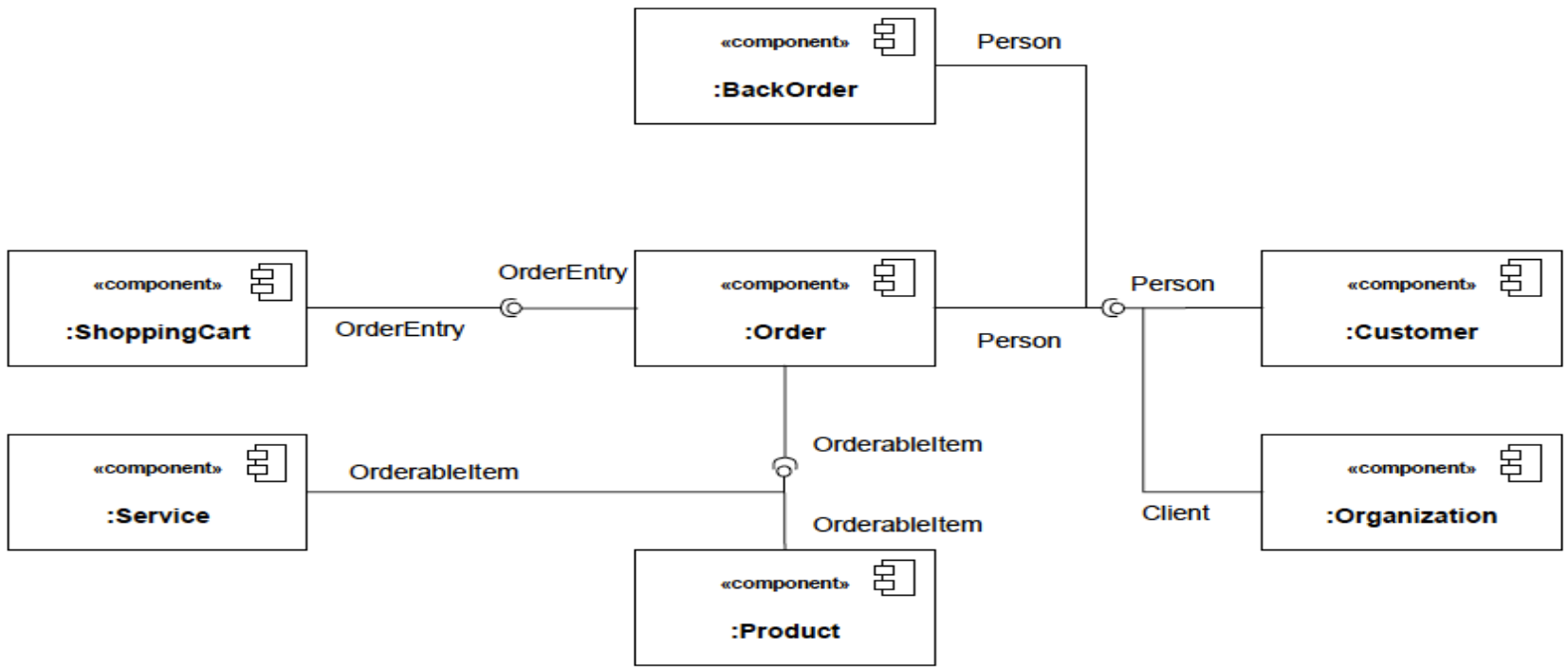


Figure 8.15 -Example of a composite structure of components, with connector wiring between provided and required interfaces of parts (Note: "Client" interface is a subtype of "Person").

Example

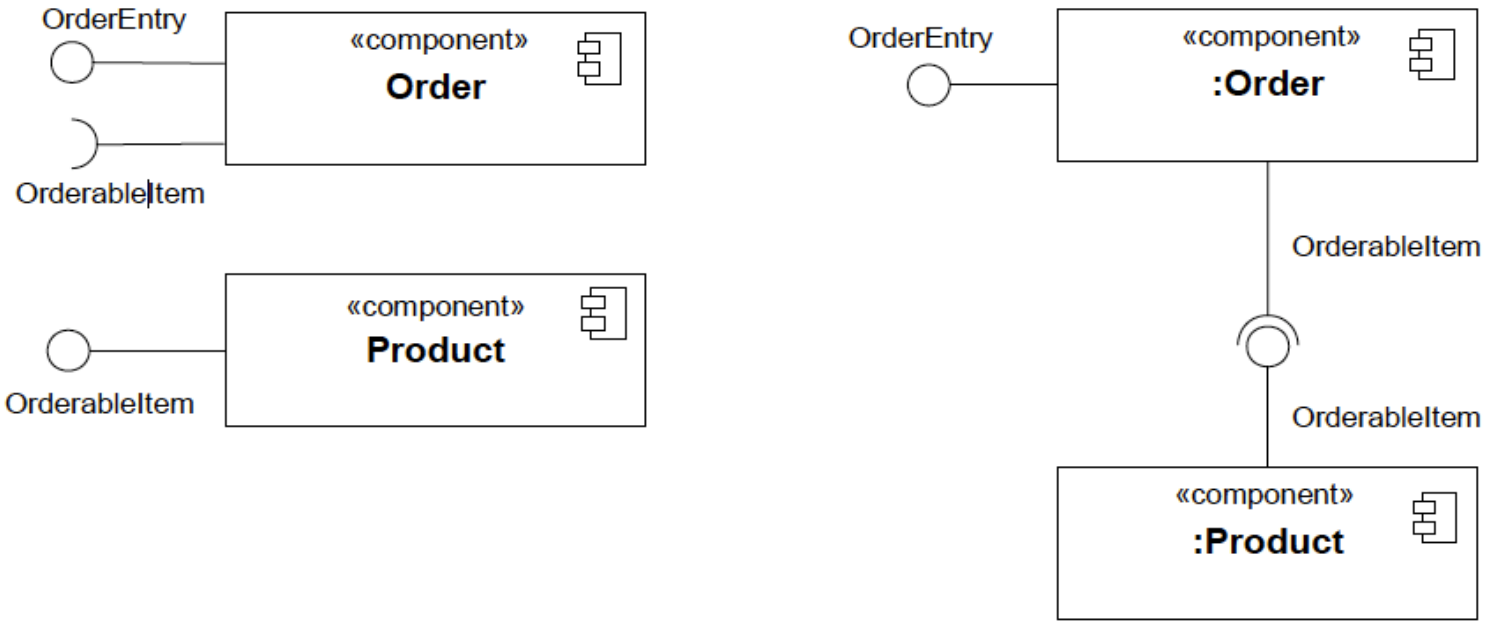


Figure 8.17 - An assembly connector maps a required interface of a component to a provided interface of another component in a certain context (definition of components, e.g., in a library on the left, an assembly of those components on the right).

Example

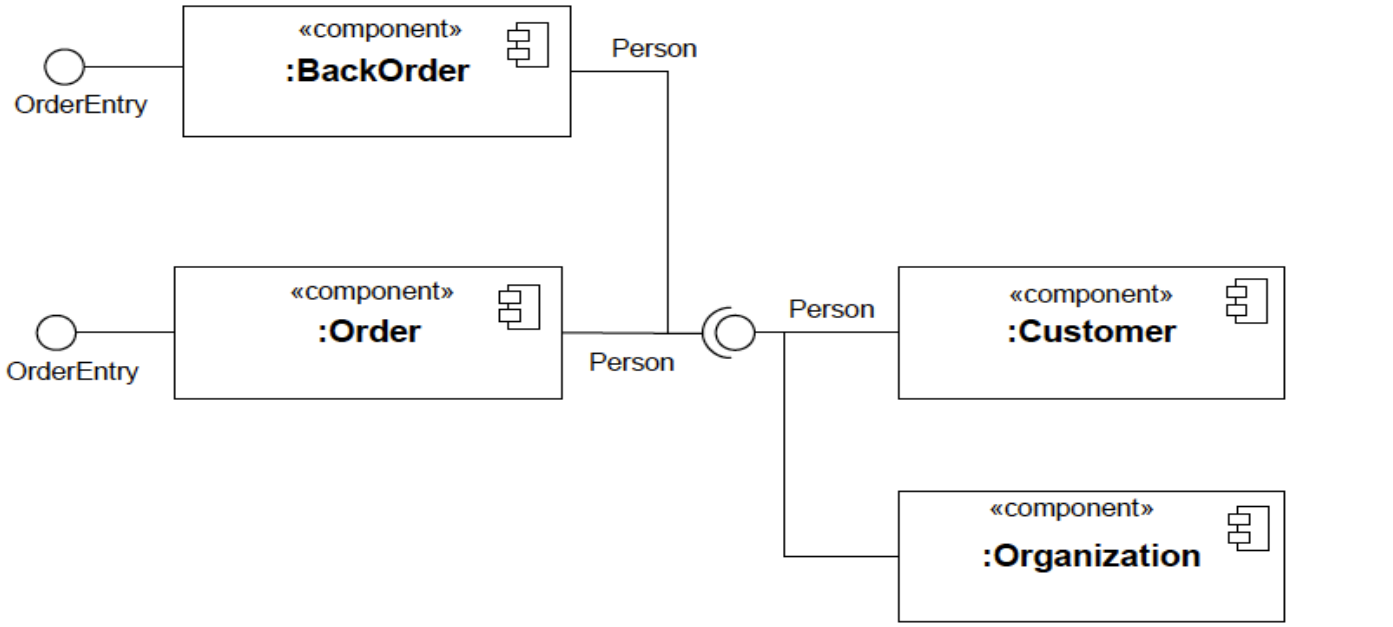


Figure 8.18 - As a notation abstraction, multiple wiring relationships can be visually grouped together in a component assembly.