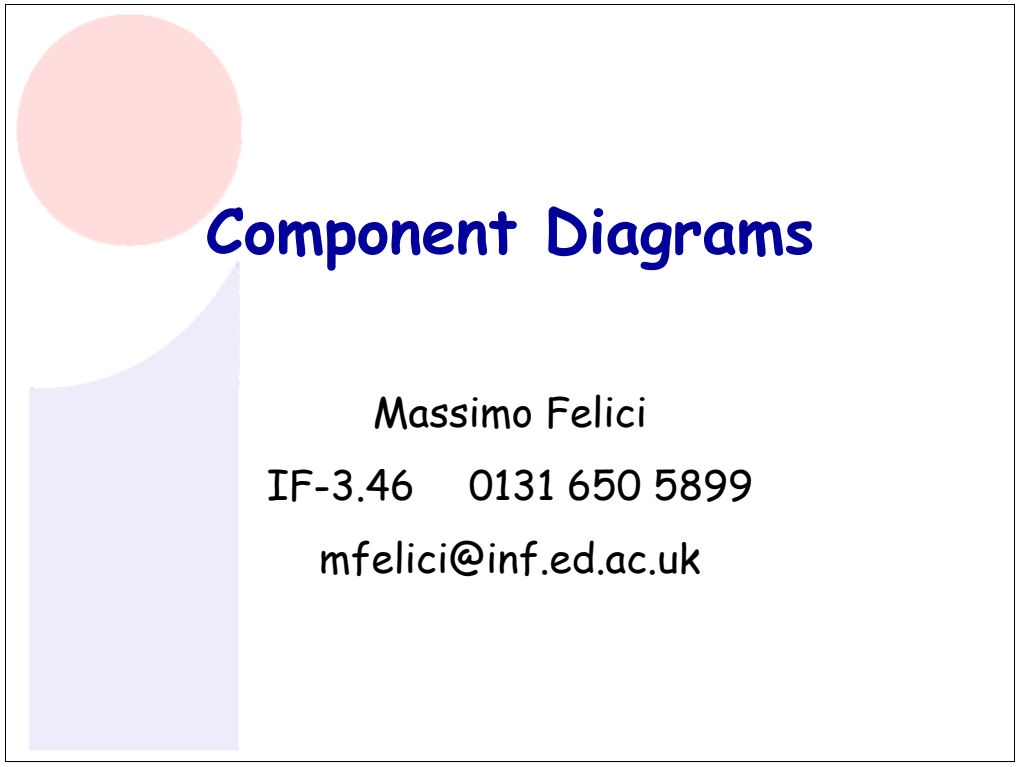

Component Diagrams
Massimo Felici
IF-3.46 0131 650 5899
mfelici@inf.ed.ac.uk

Component Diagrams

- A component is an **encapsulated**, **reusable**, and **replaceable** part of your software

Rationale

- 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.

Readings

• **UML course textbook**

  • Chapter 8 on Components Diagrams.

Component Diagrams

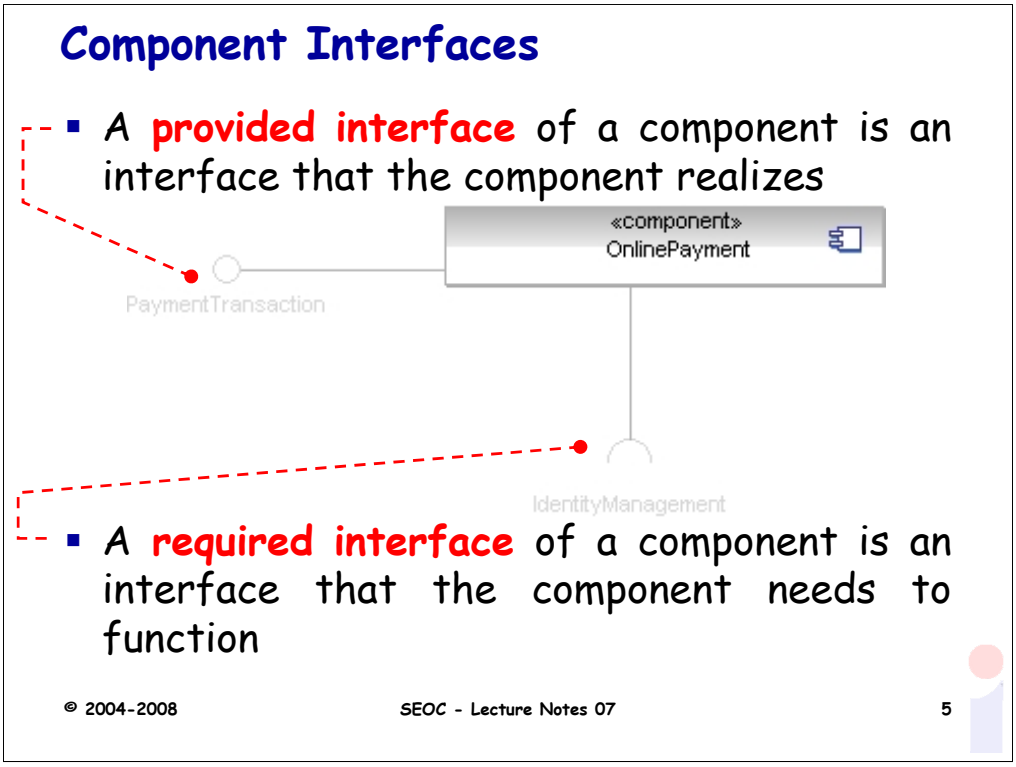
- Model physical software components and the **interfaces** between them
- Show the structure of the code itself
- Can be used to hide the specification detail (i.e., **information hiding**) and focus on the relationship between components
- Model the structure of software releases; show how components integrate with current system design
- Model source code and relationships between files
- Specify the files that are compiled into an executable

Components have **interfaces** and **context dependencies** (i.e., implementation-specific shown on diagram; use-context may be described elsewhere, for example, documentation, use-cases, interaction diagrams, etc.).

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 UML 2.0 uses a new notation for a component. Previous UML versions use the component icon as the main shape.



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".

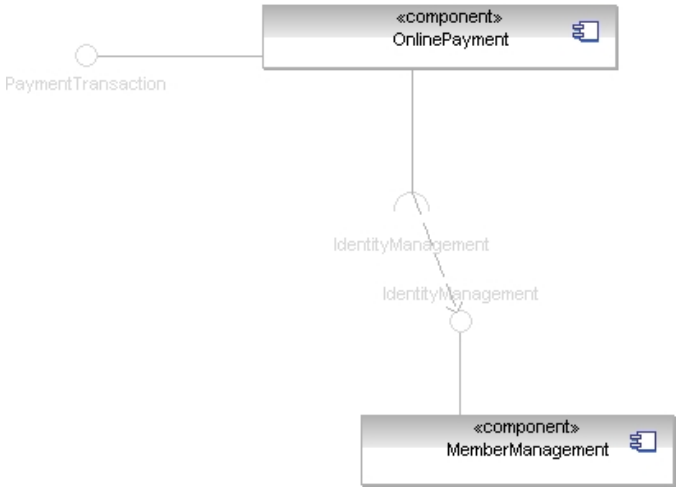
Java Warnings: Note that these definitions of interfaces differ from the Java definition of interfaces. The Java definition of interfaces does not allow to have attributes, nor hence state.

Readings

- **UML course textbook**
  - Chapter 7 on Class Diagram: Other Notations.

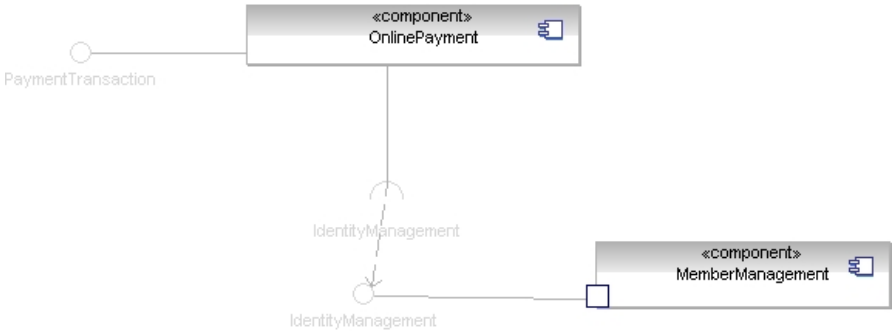
Component Assemblies

- Components can be "wired" together to form subsystems



Ports

- A port (definition) indicates that the component itself does not provide the required interfaces (e.g., required or provided). Instead, the component delegates the interface(s) to an internal class.

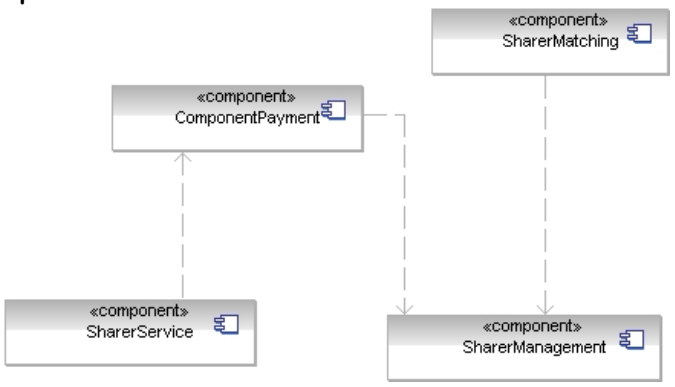


Component Realization. A component might implement (realize) the provided interfaces for the component, or it may delegate that realization to other classes that make up that component. The realization dependency can be shown in three ways: (1) listing the realization classes, (2) using realization dependency relationships, (3) showing containment graphically.

Ports Forwarding and Filtering. Ports connect to the required and provided interfaces on the outsize of the class. They can also connect to the classes of the component itself.

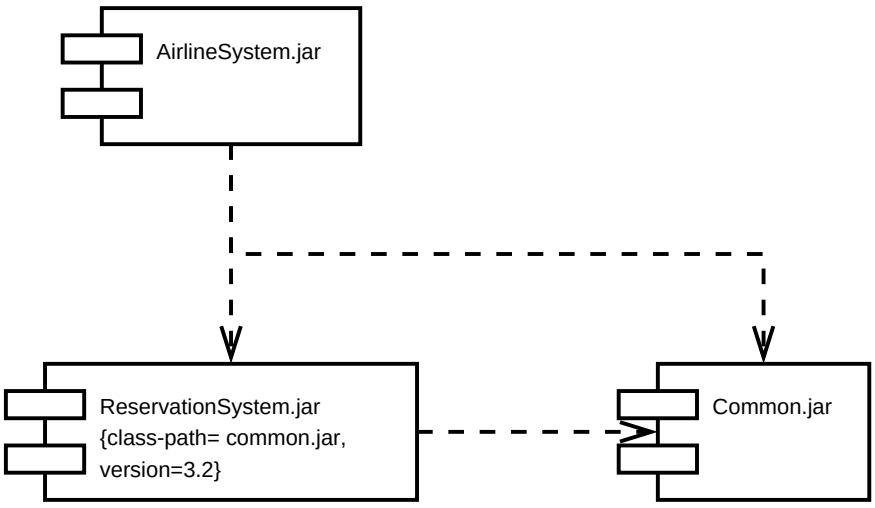
Component Modelling

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



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.

Another example of a component diagram (note the notation complies UML versions earlier than UML 2.0).



Components Diagrams

- 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.

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 component's 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

How to produce component diagrams

1. Decide on the **purpose** of the diagram
2. Add **components** to the diagram, grouping them within other components if appropriate
3. Add other **elements** to the diagram, such as classes, objects and interfaces
4. Add the **dependencies** between the elements of the diagram

Readings

- **UML course textbook**
  - Chapter 7 on Class Diagram: Other Notations.
  - Chapter 8 on Components Diagrams.

Summary

- Component Rationale
- Notation
- Component Diagrams
- Modelling