Component Diagrams

Massimo Felici

Component Diagrams

- A component is an encapsulated, reusable, and replaceable part of your software
- Reducing and defying coupling between software components
- Reusing existing components

Slide 1: Component Diagrams

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.

Required Readings

- UML course textbook, Chapter 8 on Component Diagrams

Component Diagrams

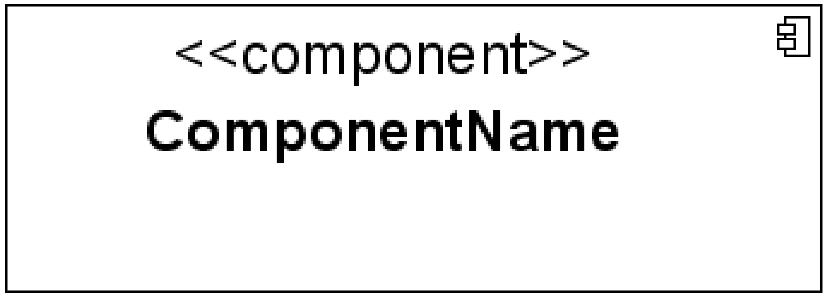
- Model physical software components and the interfaces between them
- Show the structure of the code itself
- Can be used to hide the specification detail (i.e., information hiding) and focus on the relationship between components
- Model the structure of software releases; show how components integrate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

Slide 2: Component Diagrams

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

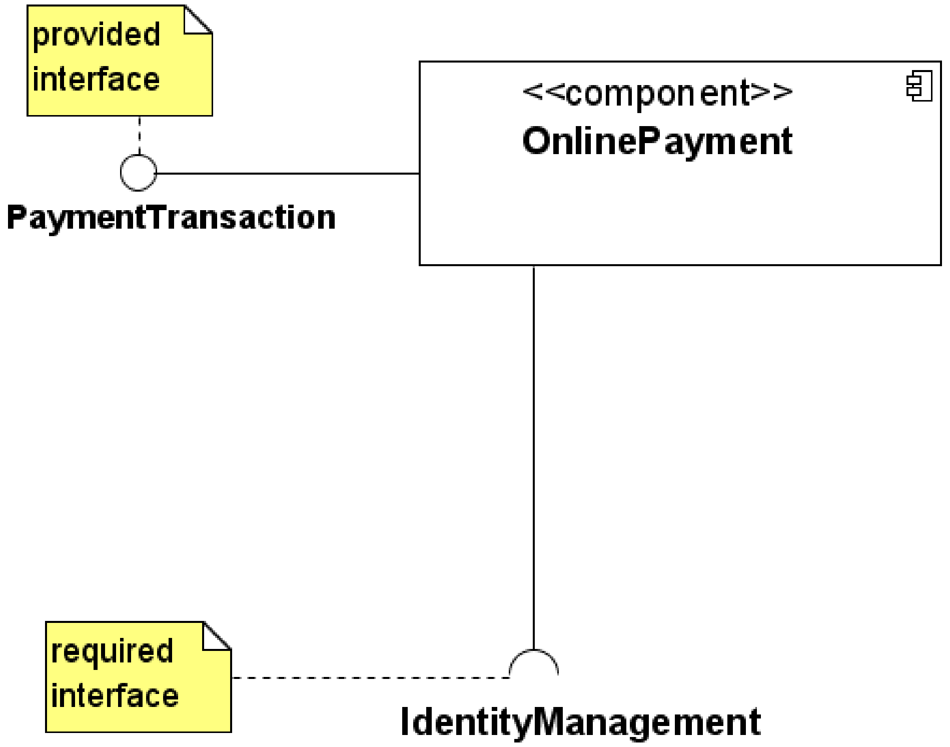
A Component is a physical piece of a system, such as a compiled object file, piece of source code, shared library or Enterprise Java Bean (EJB).



Slide 3: Component Notation

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

Component Interfaces



Slide 4: Component Interfaces

- A **provided interface** of a component is an interface that the component realises

- A **required interface** of a component is an interface that the component needs to function

Slide 4: 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”.

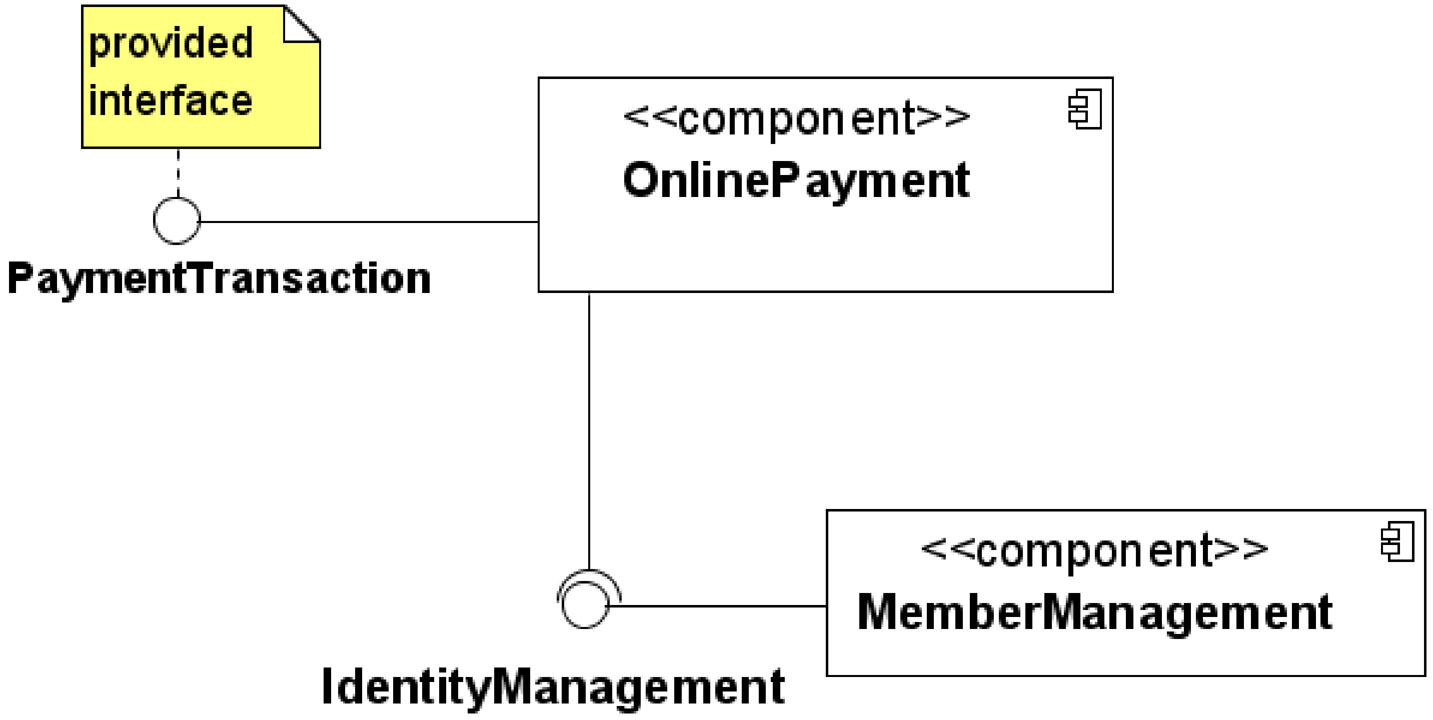
Java Warnings: Note that these definitions of interfaces differ from the Java definition of interfaces. The Java definition of interfaces does not allow to have attributes, nor hence state.

Required Readings

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

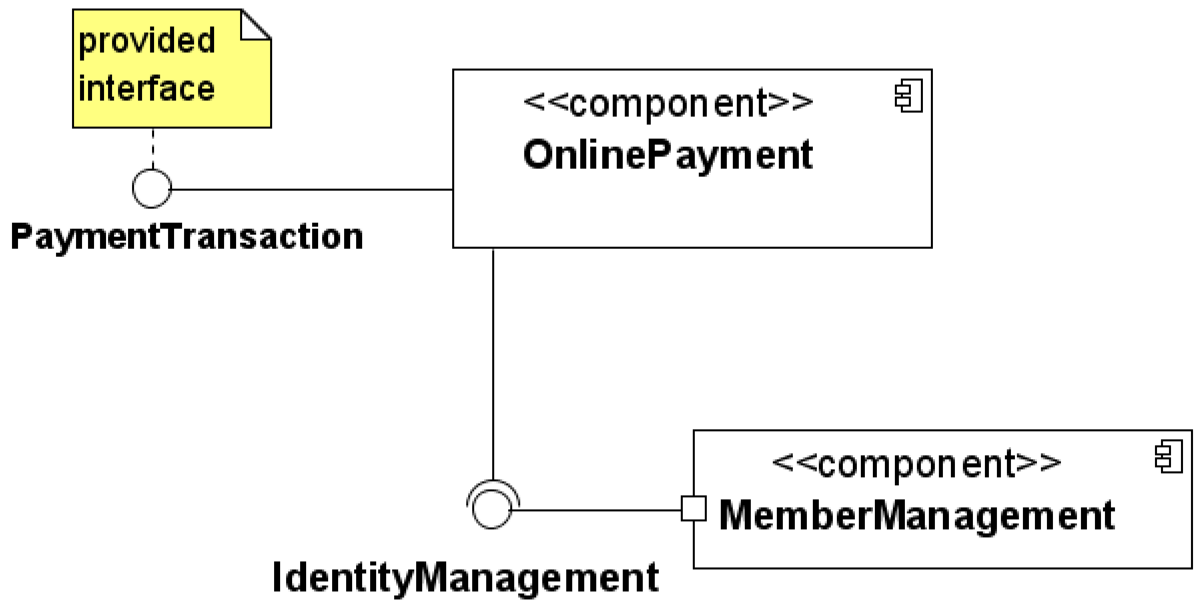
Component Assemblies

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.



Slide 6: Ports

Component Realisation. A component might implement (realise) the provided interfaces for the component, or it may delegate that realisation to other classes that make up that component. The realisation dependency can be shown in three ways:

1. listing the realisation classes
2. using realisation dependency relationships
3. showing containment graphically.

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

Component Diagrams

- **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.

Slide 7: 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.

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

Required Readings

- UML course textbook, Chapter 7 on Class Diagram: Other Notations
- UML course textbook, Chapter 8 on Component Diagrams

Summary

- Component Rationale
- Notation
- Component Diagrams
- Modelling