Subject:

Object-oriented analysis and design

Chapter 6: Component diagrams

Lê Văn Vinh, PhD
Department of Software Engineering
Faculty of Information Technology
University of Technical Education HCMC

Contents

1. **Component diagram**
   a. Introduction
   b. Component: defining and notation
   c. Types of Interfaces
   d. Connecting components
   e. Classes within a component
   f. Ports and internal structures
   g. Types of Connectors
   h. Black-box and White-box views
2. **Architectural styles**
3. **Decomposing system**

Introduction

- **It’s hard to define the classes directly from the requirements.**
- **We need to plan out the high-level pieces of the system to establish the architecture.**
- **Component diagrams form the part of the development view by showing the components of the system.**

What is a component?

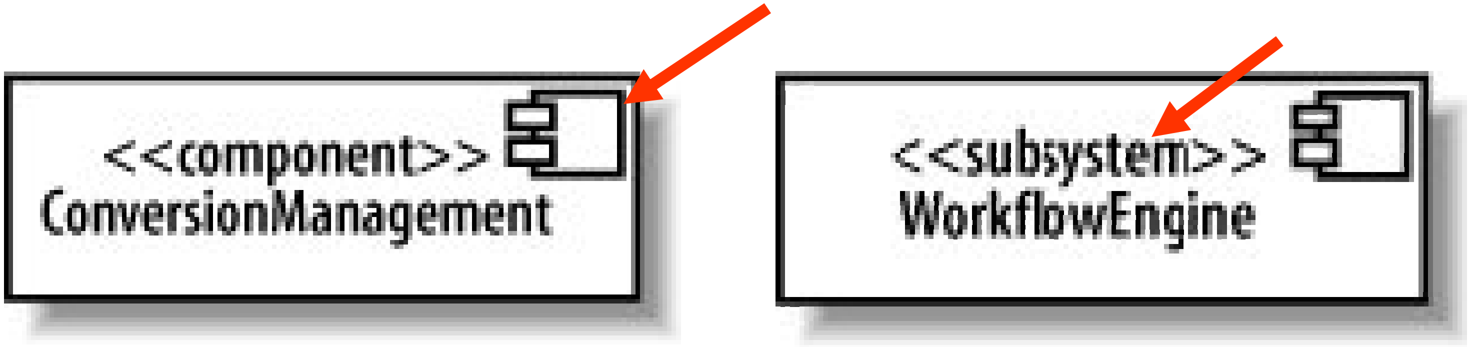
- A component is an **encapsulated**, **reusable**, and **replaceable** part of the software system

- Components can range in size from relatively **small**, about the size of a class, up to a **large** subsystem

- Each component usually performs a **key functionality** in the system

Component notation

- Represented by a rectangle with the <<component>> stereotype
- The tabbed rectangle icon in the upper righthand corner may be optional <<subsystem>> stereotype may be used



Provided and Required Interfaces

- Components interact with each other through provided and required interfaces.
- The purpose is:
  - To control dependencies between components
  - To make components swappable

Provided and Required Interfaces

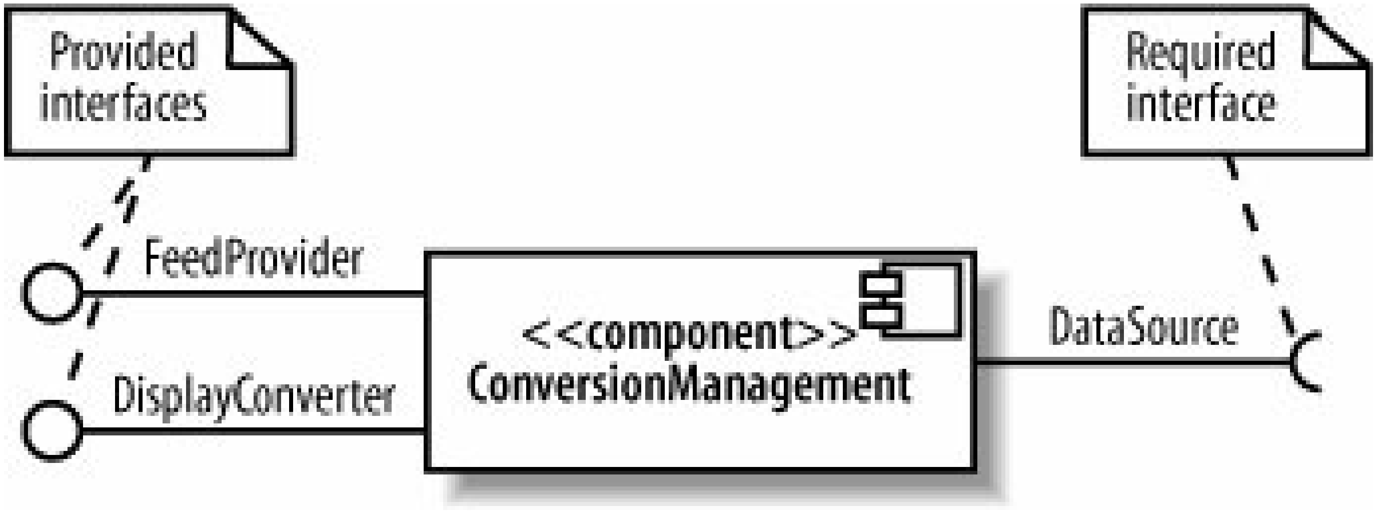
- A **provided interface** of a component is an interface that the component **realizes**.
- Other components and classes interact with a component through its provided interfaces .
- A component's provided interface describes the services provided by the component.

Provided and Required Interfaces

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

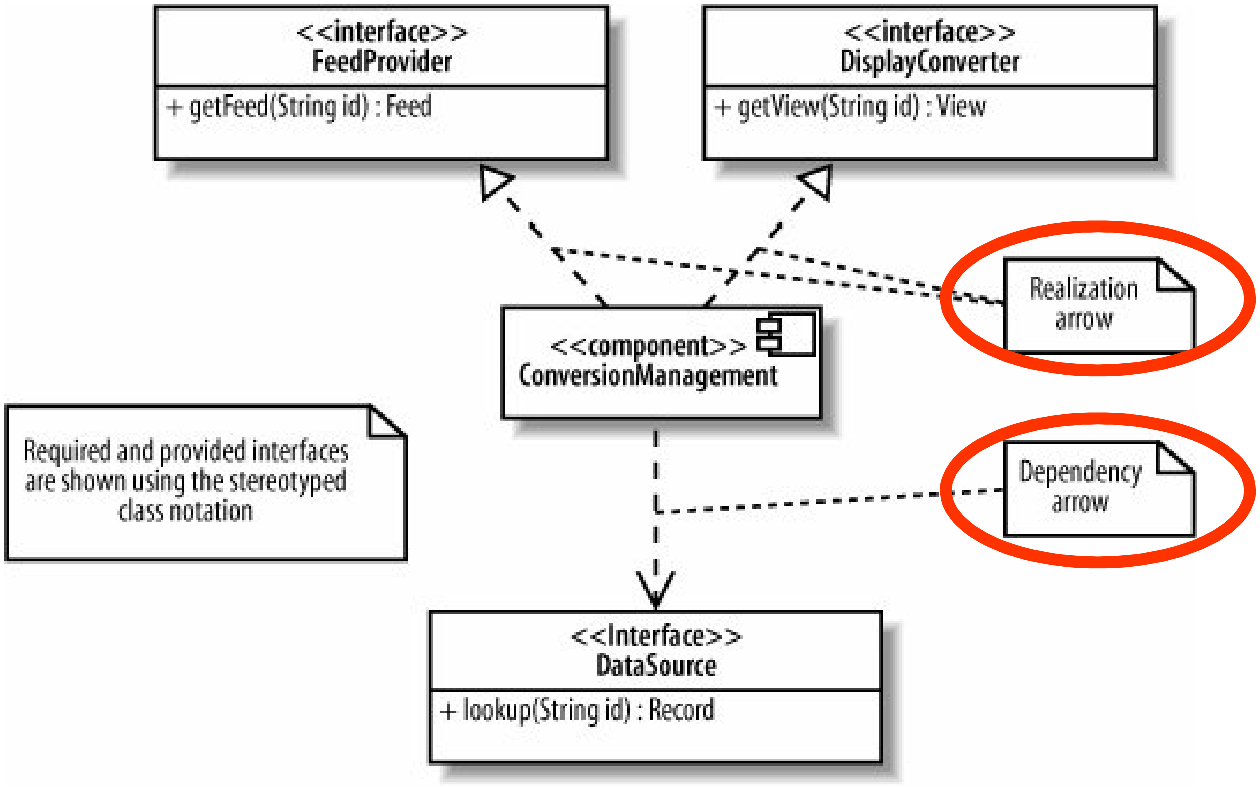
Ball and Socket Notation

- **Balls represent provided interfaces**
- **“Sockets” – half of a circle: represent required interfaces**



Stereotype Notation

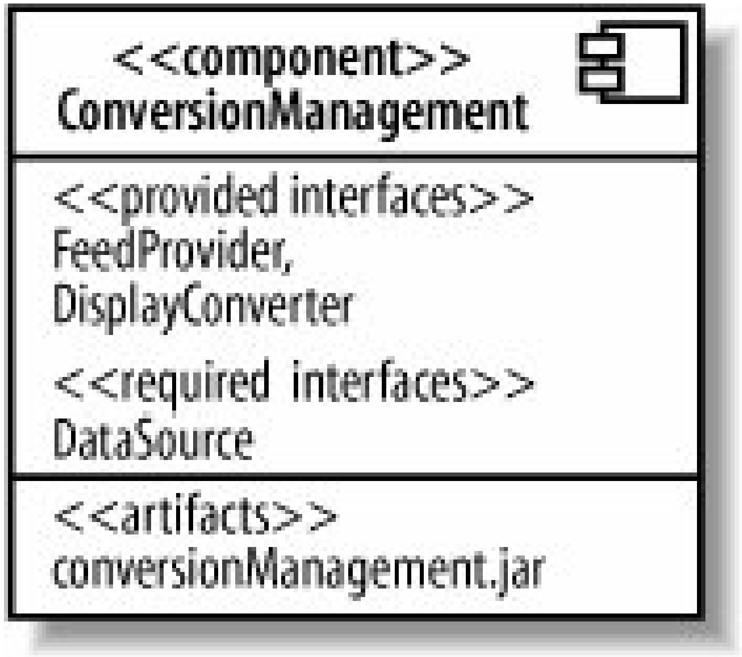
- This way is helpful to **show the operations** of interfaces.



<<interface>>
FeedProvider
+ getFeed(String id) : Feed
<<interface>>
DisplayConverter
+ getView(String id) : View
Realization arrow
<<component>>
ConversionManagement
Required and provided interfaces are shown using the stereotyped class notation
Dependency arrow
<<Interface>>
DataSource
+ lookup(String id) : Record

Listing Component Interfaces

- **Provided & required interfaces are listed separately.**
- **The <<artifacts>> section lists the artifacts, or physical files, manifesting the component.**



<<component>>
ConversionManagement

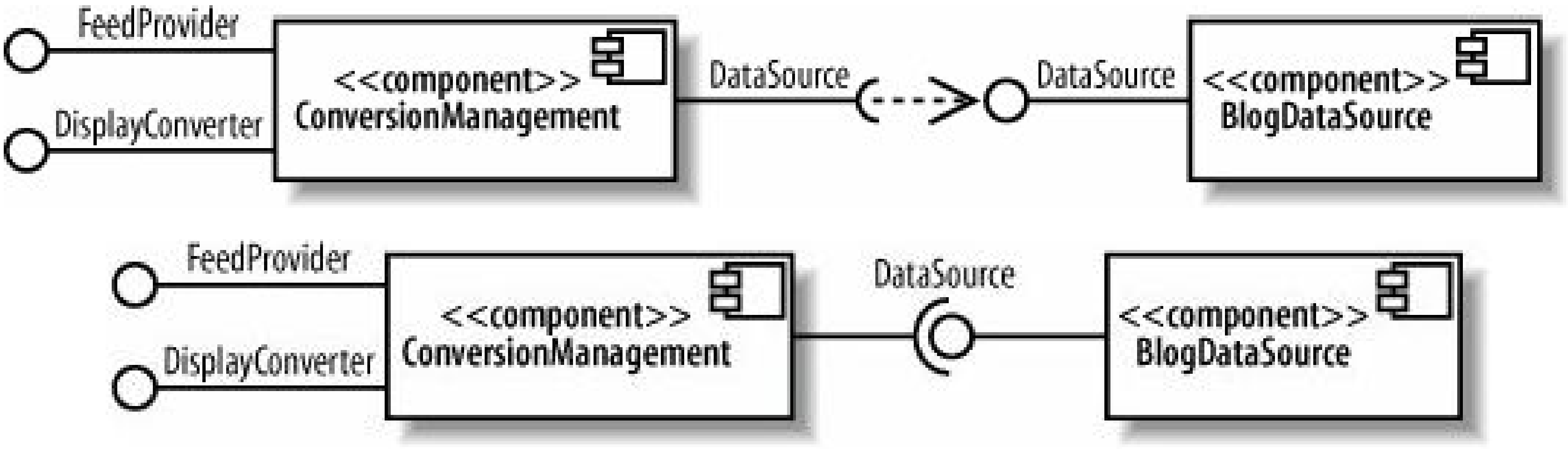
<<provided interfaces>>
FeedProvider,
DisplayConverter

<<required interfaces>>
DataSource

<<artifacts>>
conversionManagement.jar

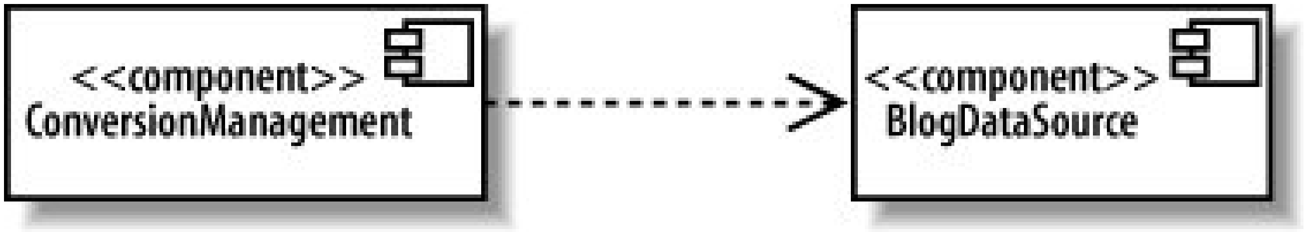
Connecting components

- **The dependency arrow is used to connect from the socket of one component to the ball of another component.**

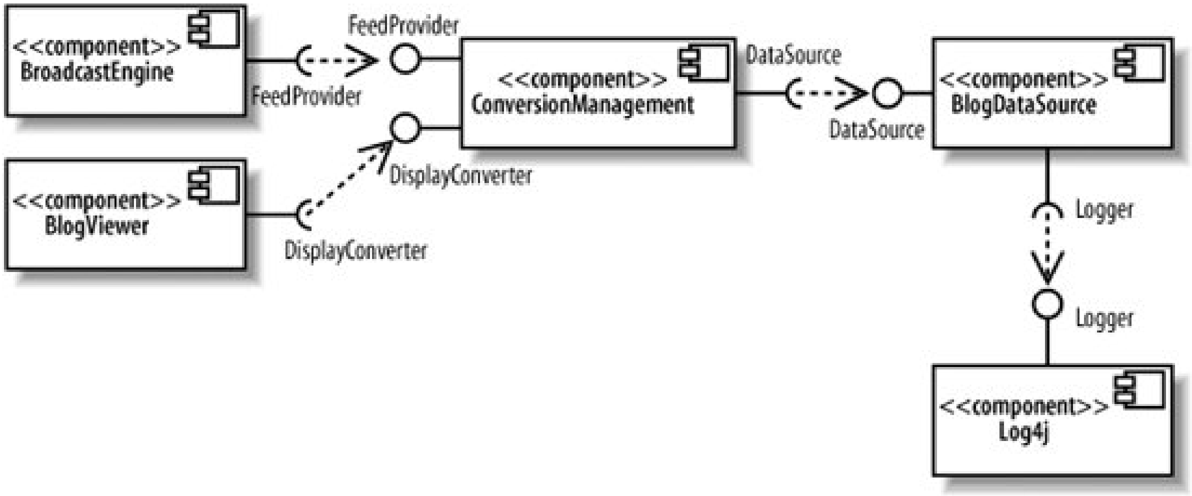


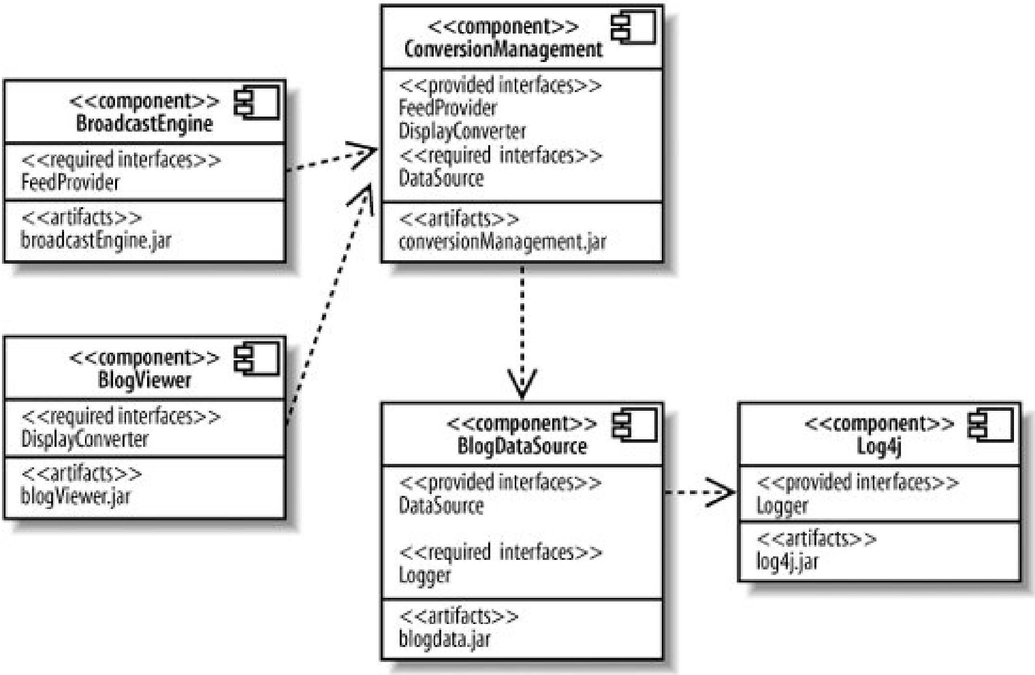
Connecting components (2)

- **This notation is useful in showing simplified higher level views of component dependencies.**



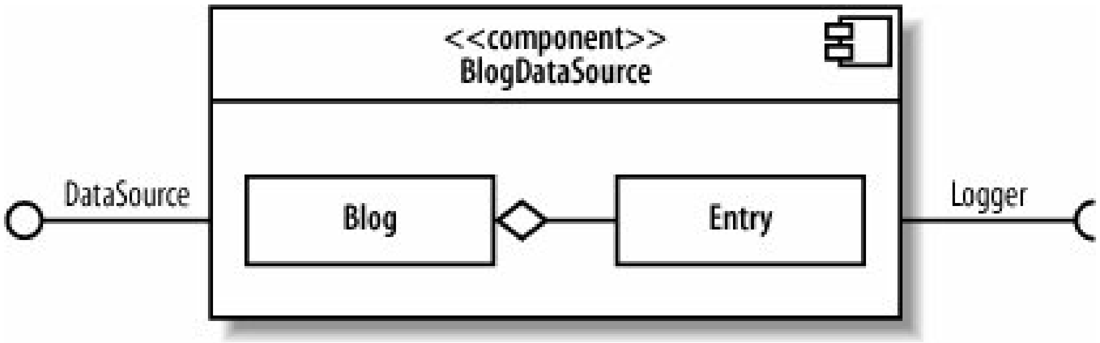
Connecting components (3)

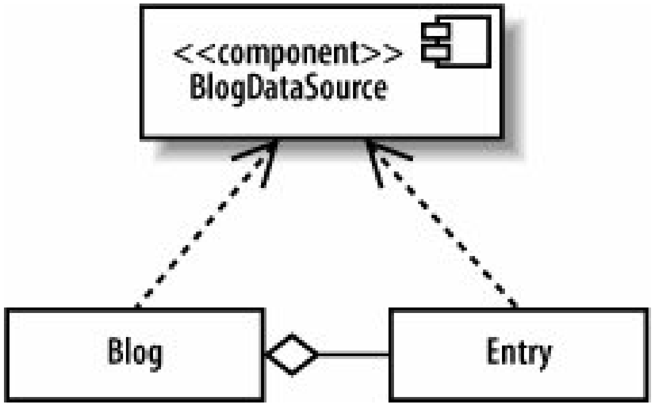




Classes That Realize a Component

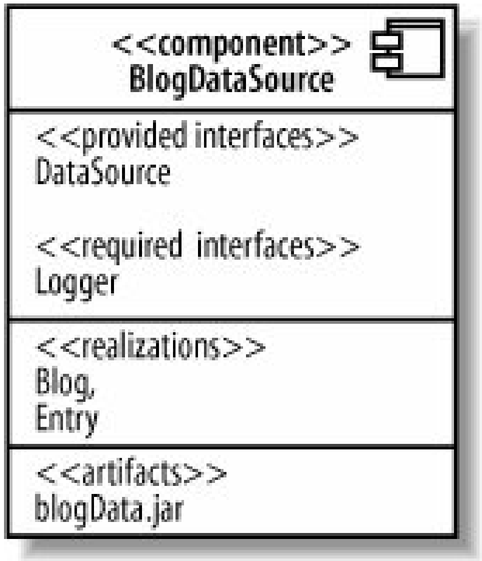
- A component often contains and uses other classes to implement its functionality.
- These classes realize the component





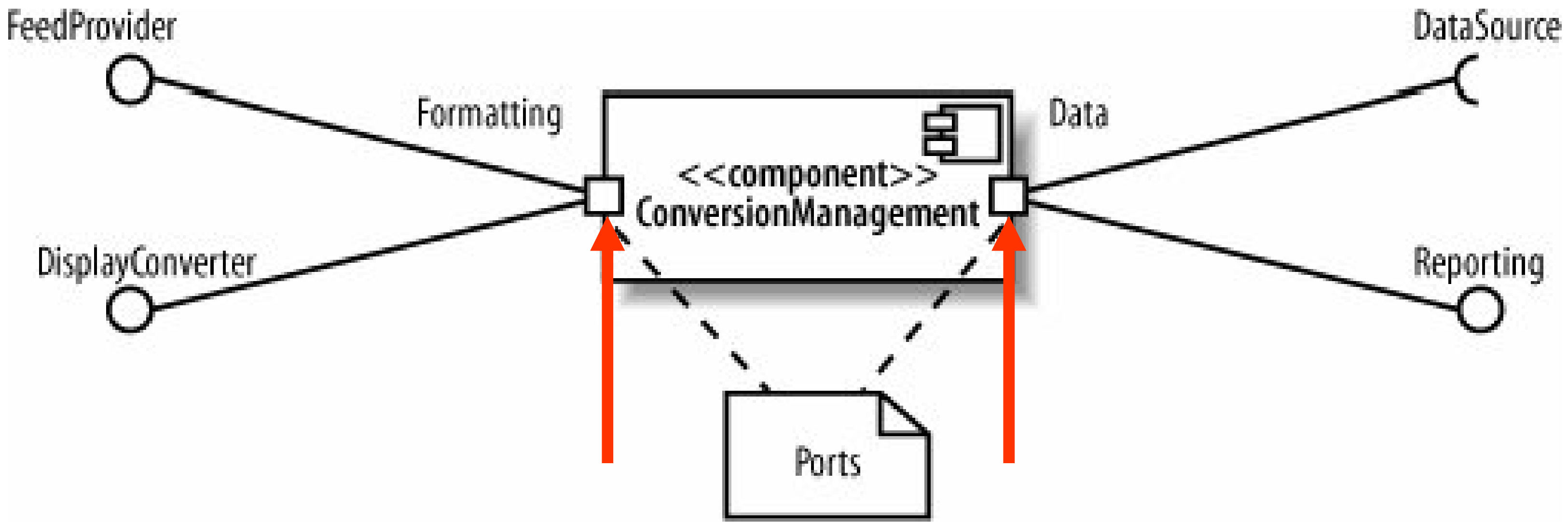
Classes That Realize a Component

- An alternate way to represent: more **compact.**
- However, it **can't show relationships** between the realizing classes.



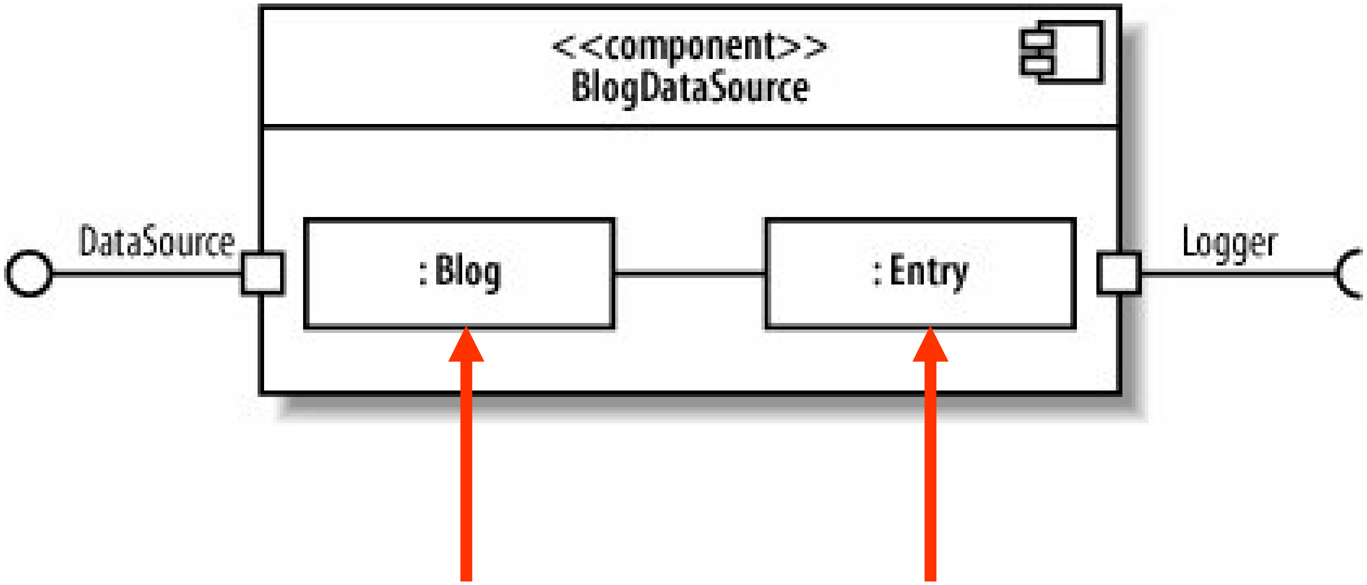
Ports used in a component

- **Used to model distinct ways that a component can be used with related interfaces attached to the port**



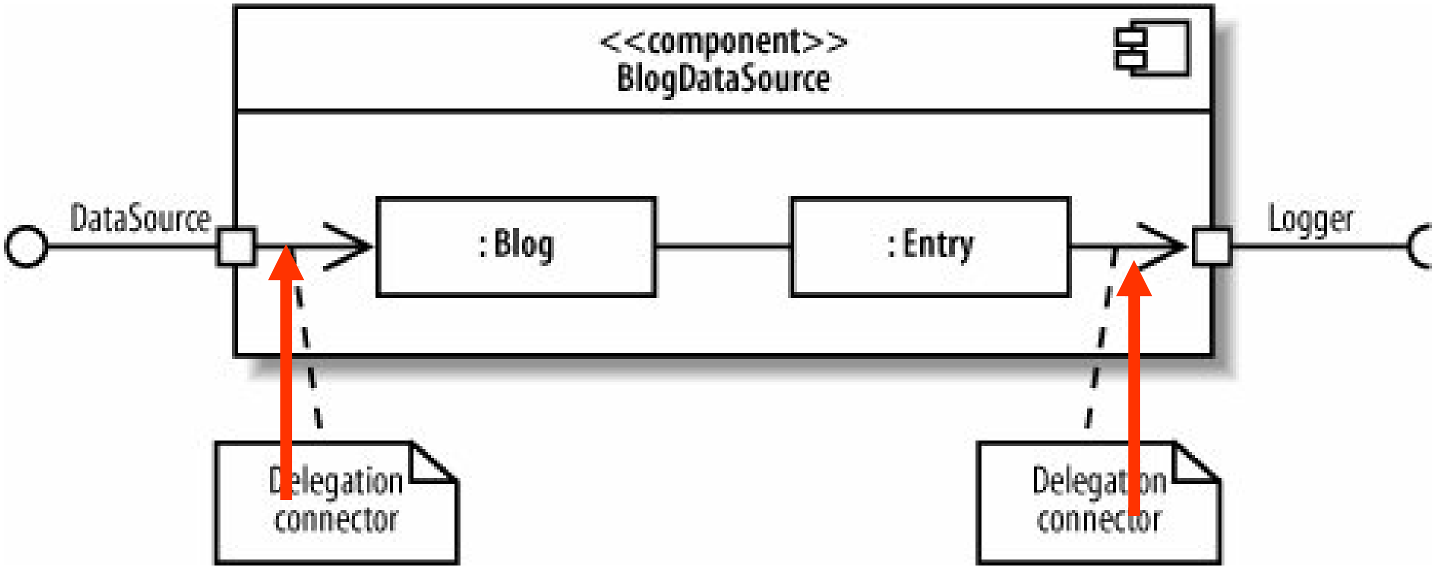
Internal structure

- **Used to model the parts, properties, and connectors within a component.**



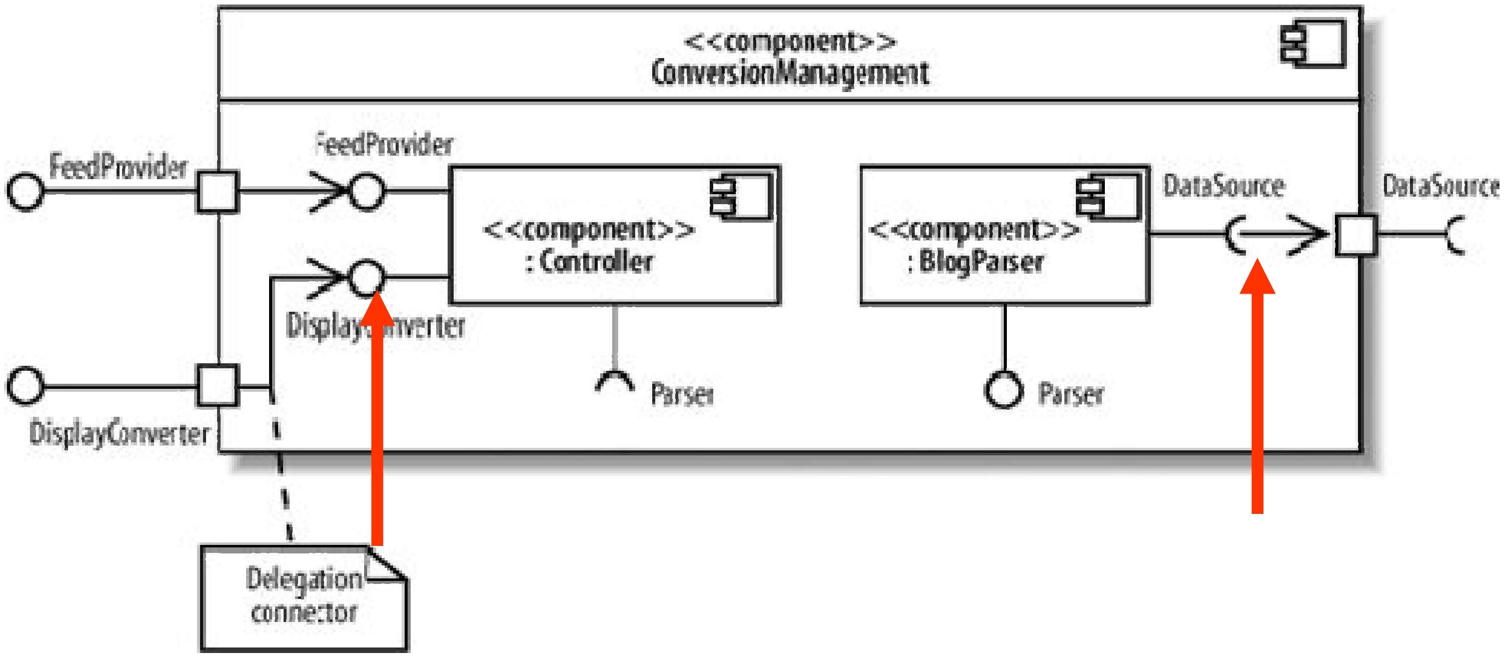
Delegation Connectors

- **Used to show that internal parts realize or use the component's interfaces.**



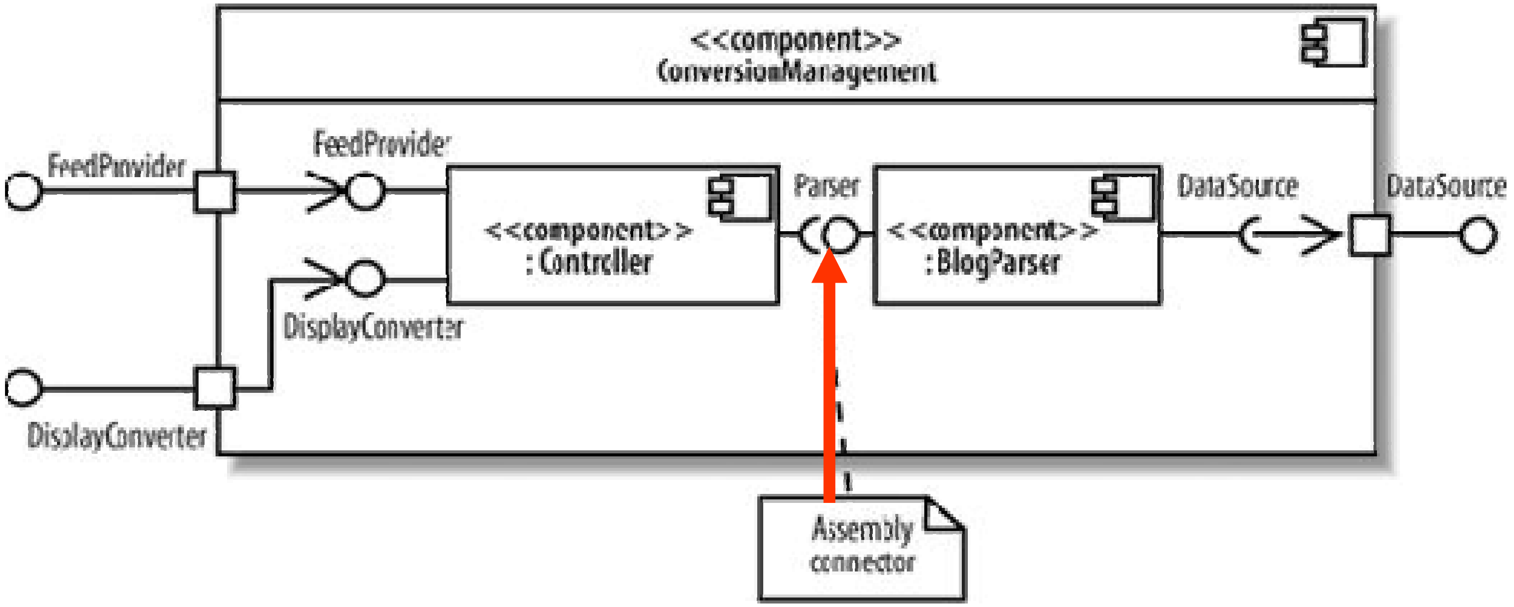
Delegation Connectors (2)

- **Delegation connectors can also connect interfaces of internal parts with ports**



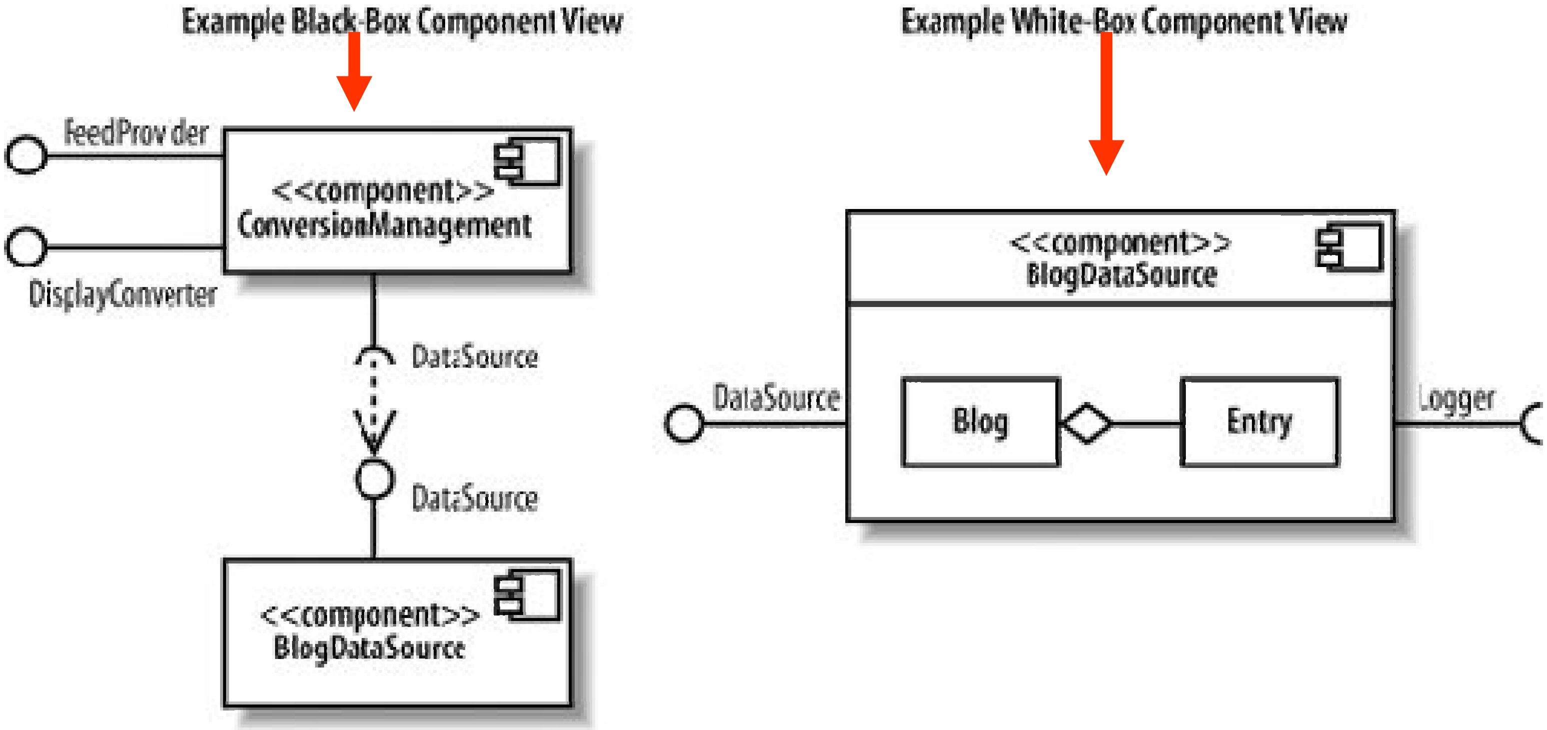
Assembly Connectors

- **Used to show components within another component working together through interfaces.**



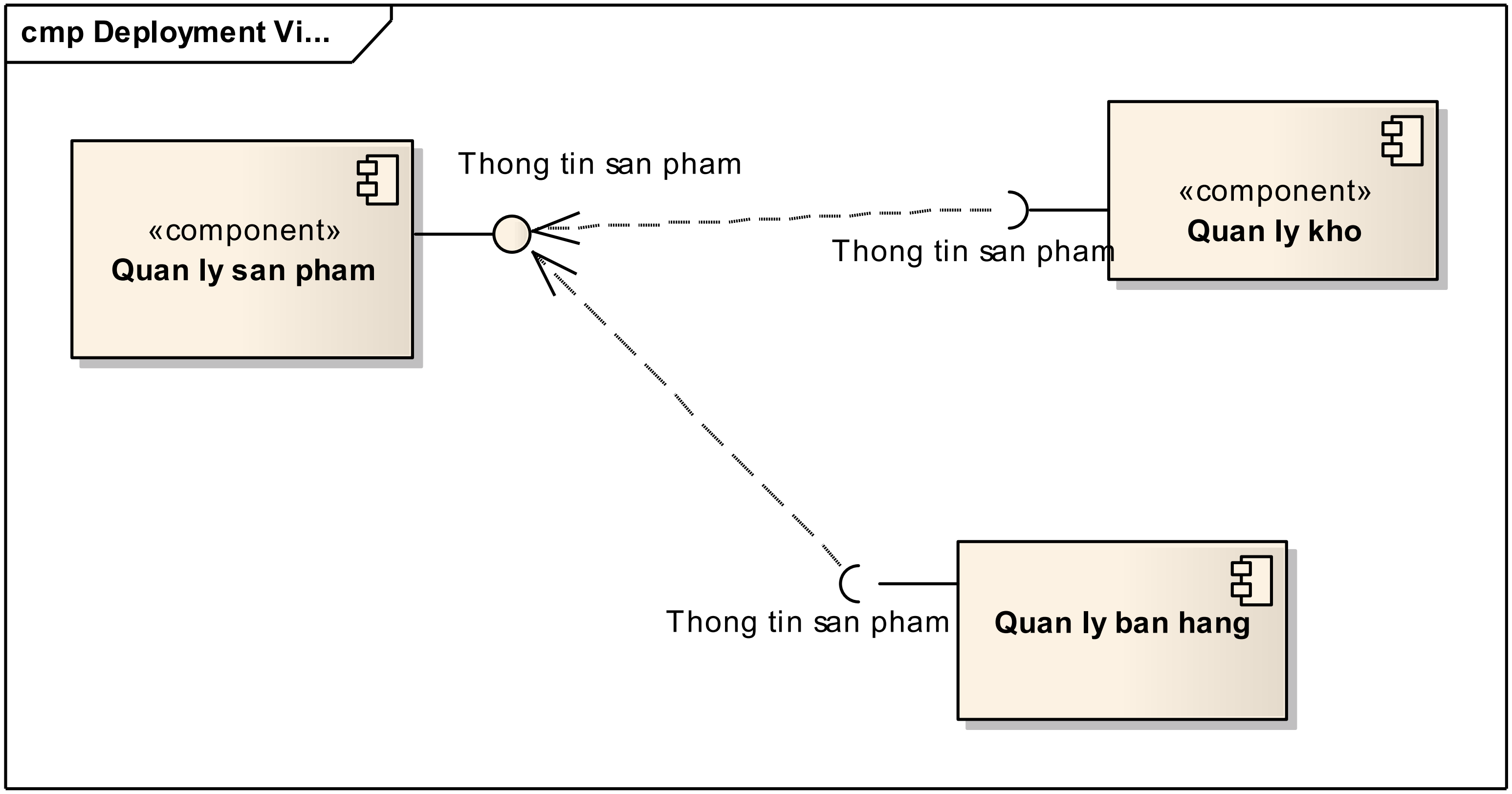
Black-Box and White-Box Views

- **Black-Box: outside view only**
- **White-Box: focuses on the internal structure of the components**



Ví dụ 1

Black-box component view



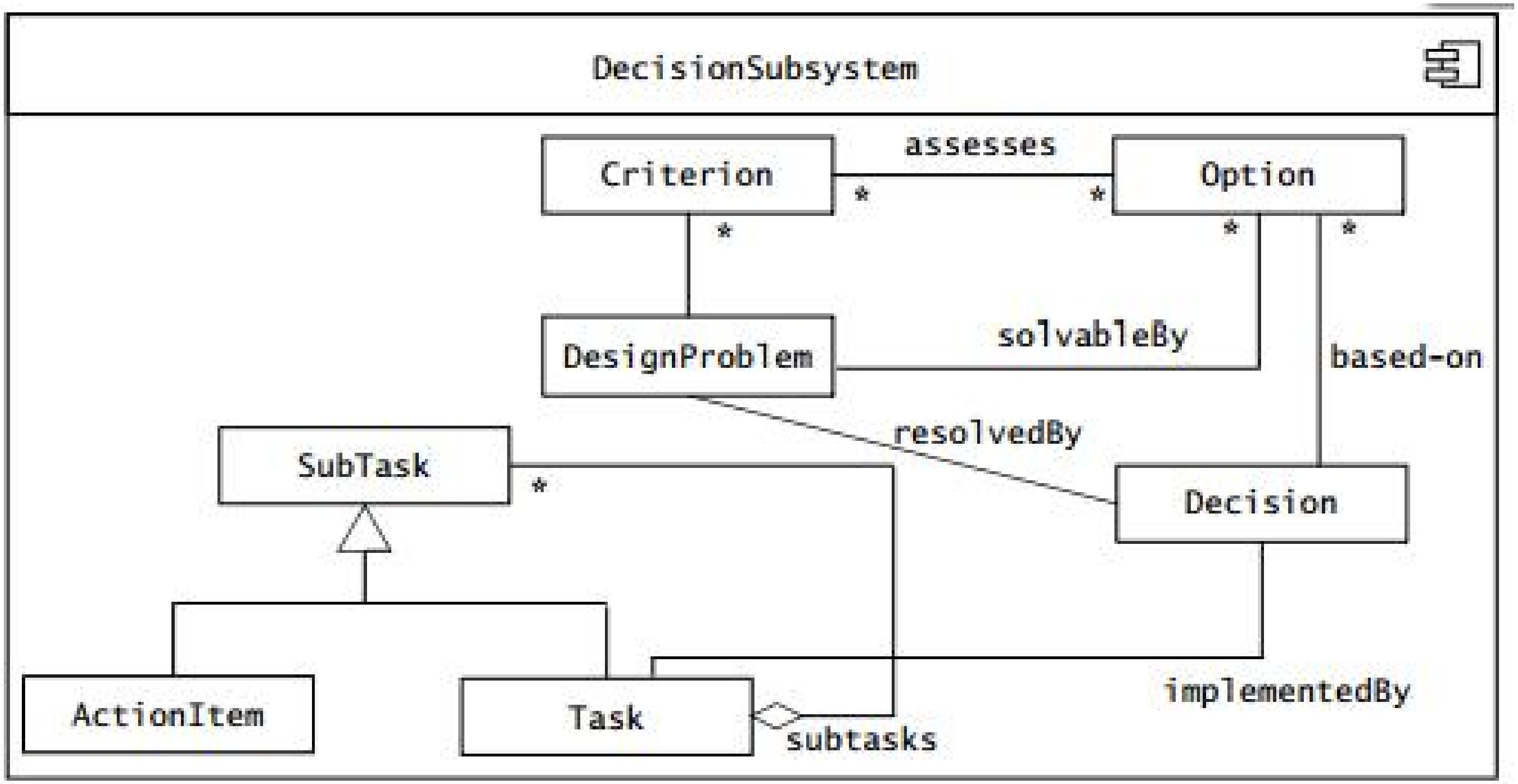
Contents

1. **Component diagram**
   a. Introduction
   b. Component: defining and notation
   c. Types of Interfaces
   d. Connecting components
   e. Classes within a component
   f. Ports and internal structures
   g. Types of Connectors
   h. Black-box and White-box views
2. **Architectural styles**
3. **Decomposing system**

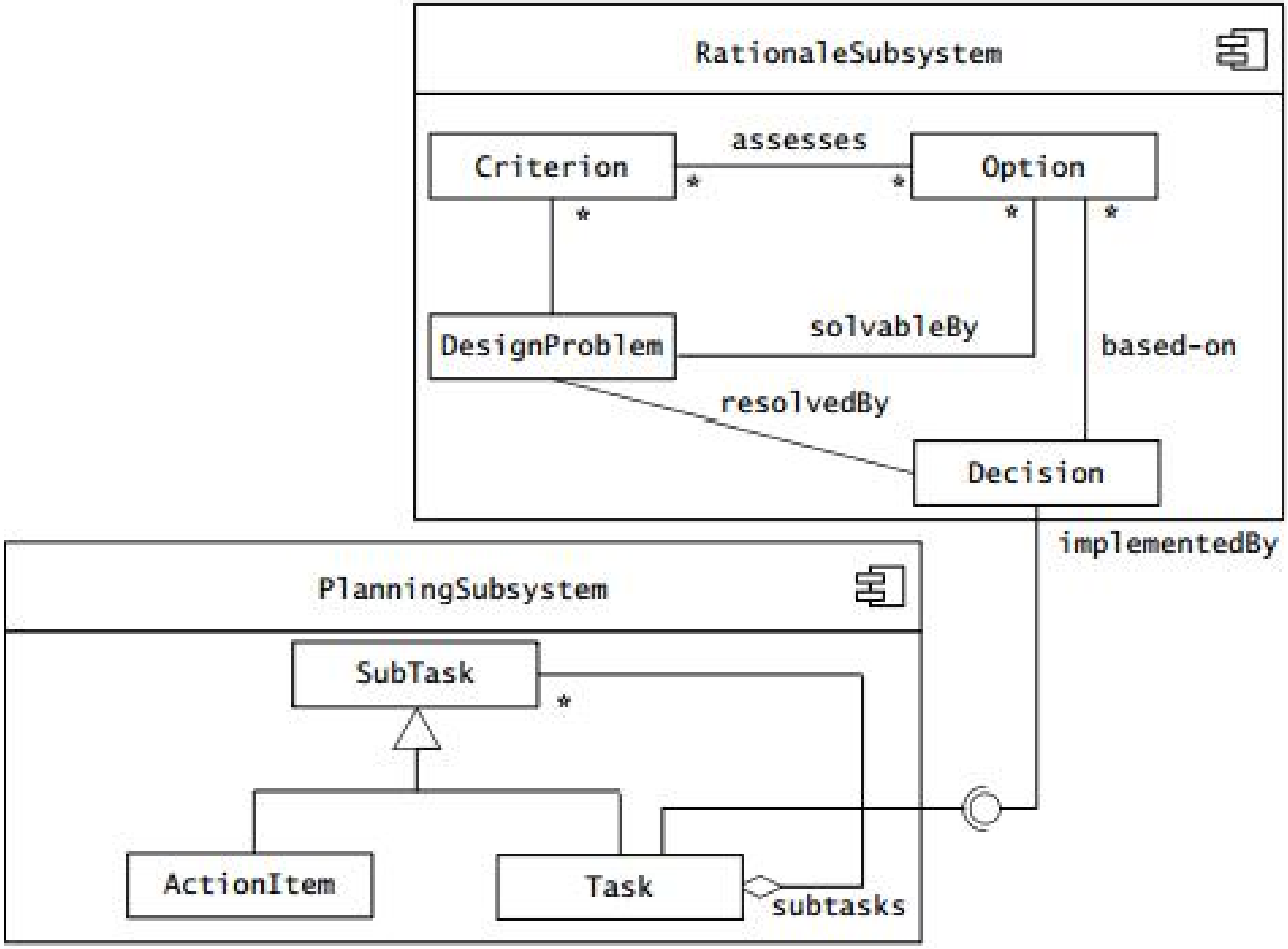
How to decompose the system

- **Cohesion**
  - Is the number of dependencies within a system
  - If a subsystem contains many objects that are related to each other and perform similar tasks, its cohesion is high
- **Coupling**
  - is the number of dependencies between two subsystems.
  - If two subsystems are
    loosely coupled, they are relatively independent

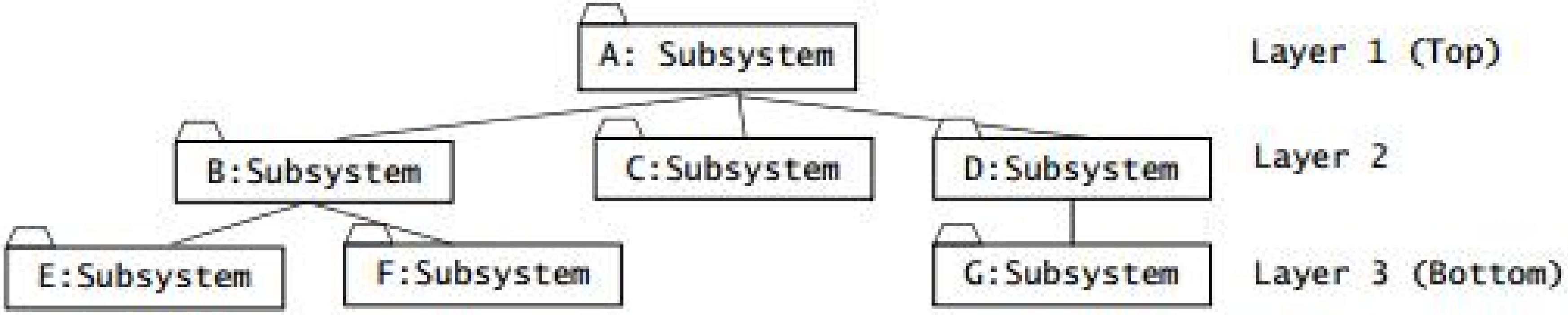
Example


DecisionSubsystem
Criterion
assesses
*
*
Option
*
*
*
DesignProblem
solvableBy
based-on
resolvedBy
SubTask
*
Decision
ActionItem
Task
subtasks
implementedBy

Example


RationaleSubsystem
Criterion
assesses
Option
*
*
*
*
*
DesignProblem
solvableBy
based-on
resolvedBy
Decision
implementedBy
PlanningSubsystem
SubTask
*
ActionItem
Task
subtasks

Layers and Paritions



Architectural styles

❖ Repository

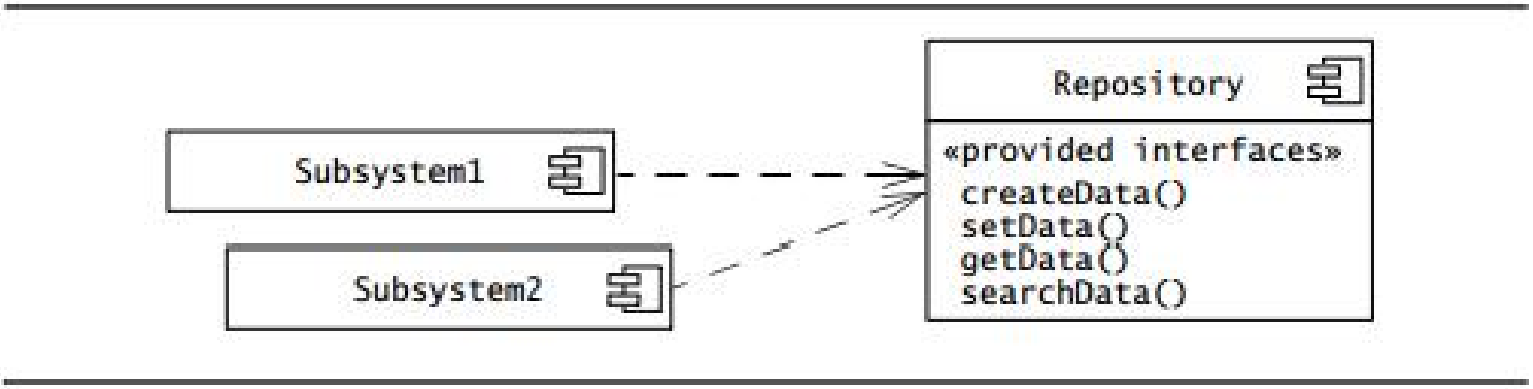
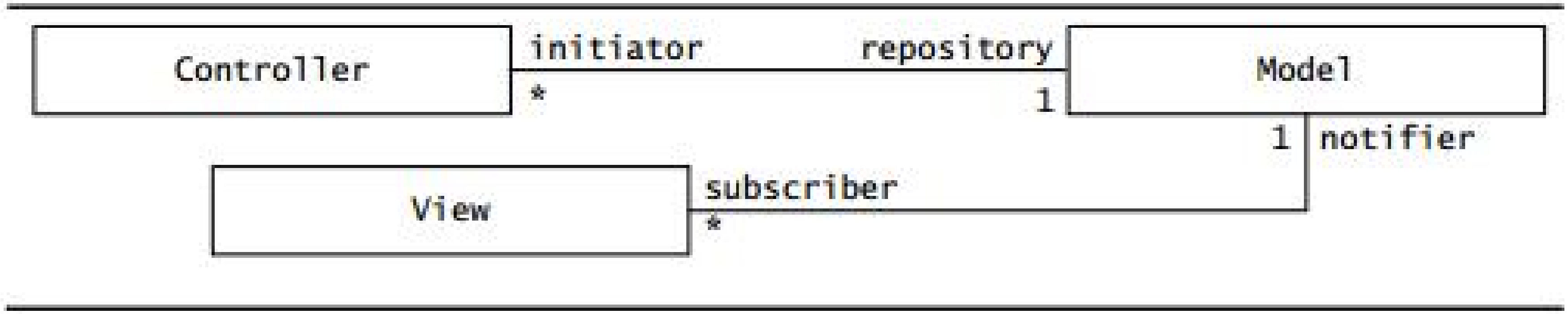


Figure 6-13 Repository architectural style (UML component diagram). Every Subsystem depends only on a central data structure called the Repository. The Repository has no knowledge of the other Subsystems.

Architectural styles

Model/View/Controller

- The Controller gathers input from the user and sends messages to the Model
- The Model maintains the central data structure
- The Views display the Model and are notified (via a subscribe/notify protocol) whenever the Model is changed.



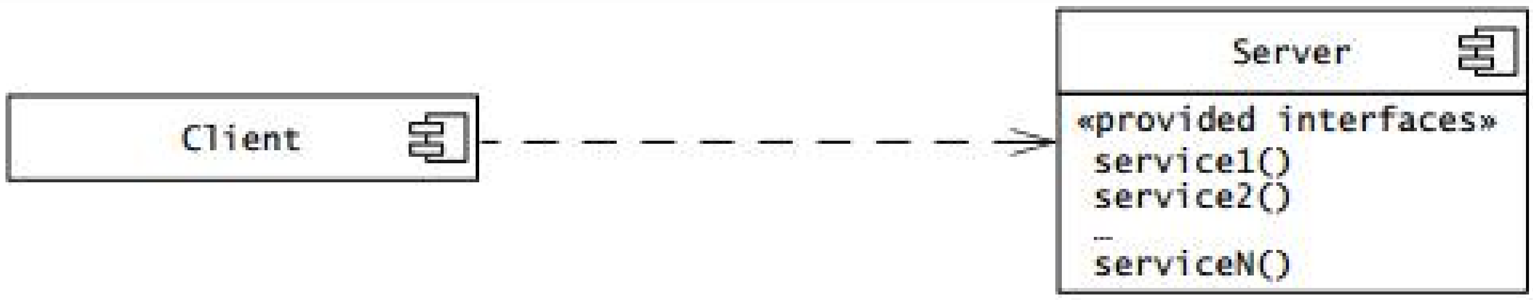
Architectural styles

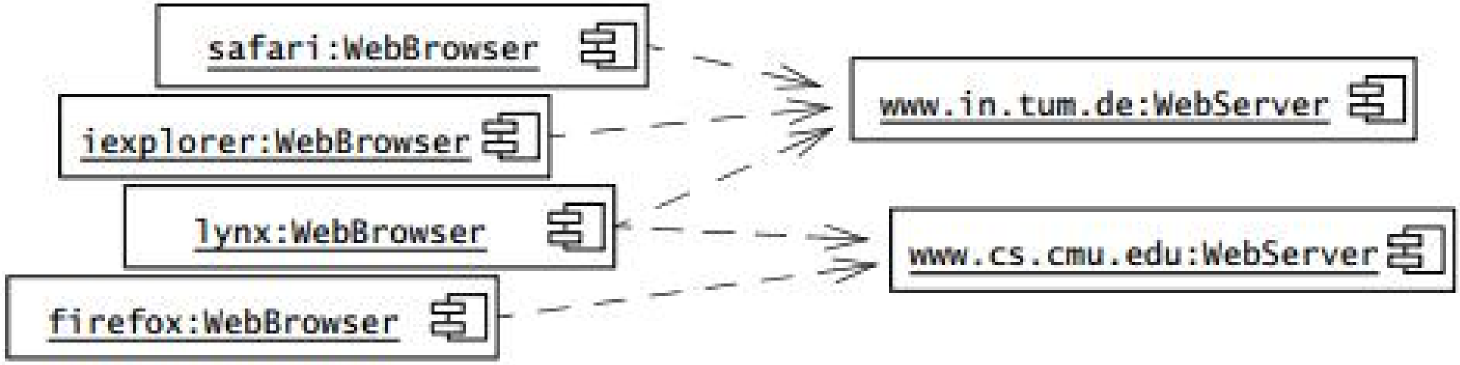
Model/View/Controller

- Subsystems are classified into three different types:
  - **Model** subsystems maintain domain knowledge,
  - **View** subsystems display it to the user
  - **Controller** subsystems manage the sequence of interactions with the user
- The model subsystems are developed such that they do not depend on any view or controller subsystem.

Architectural styles

❖ Client/server





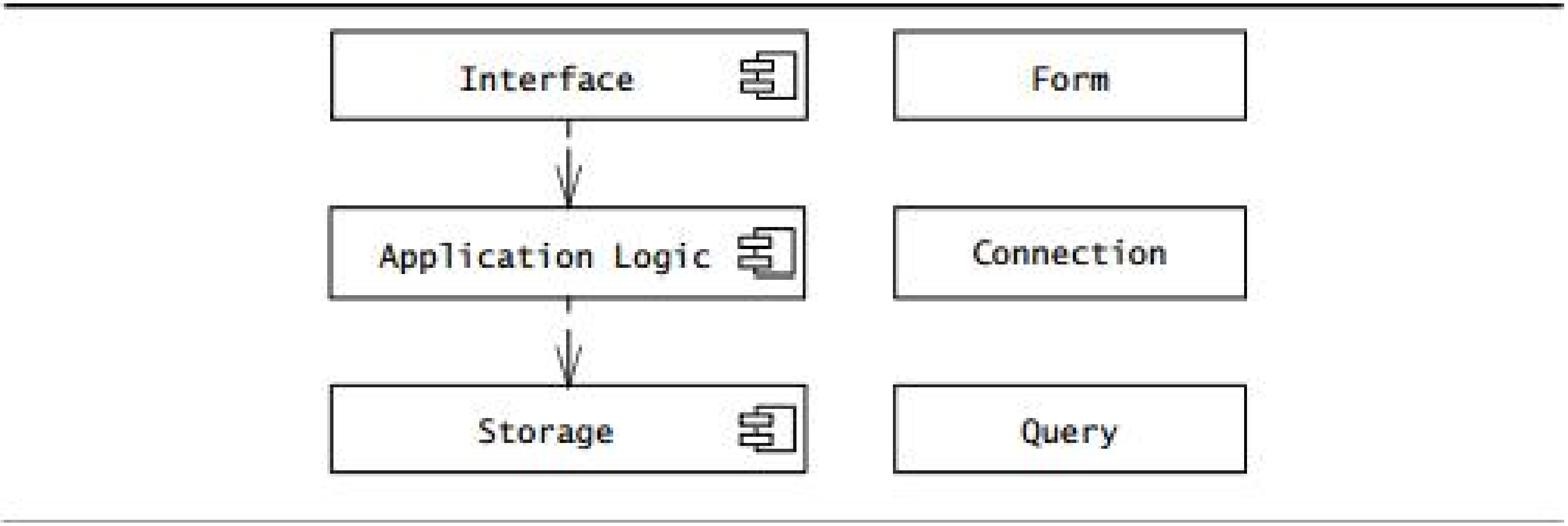
Architectural styles

Three-tier

- The three-tier architectural style organizes subsystems into three layers (Figure 6-22):
  - The *interface layer* includes all boundary objects that deal with the user, including windows, forms, web pages, and so on.
  - The *application logic layer* includes all control and entity objects, realizing the processing, rule checking, and notification required by the application.
  - The *storage layer* realizes the storage, retrieval, and query of persistent objects

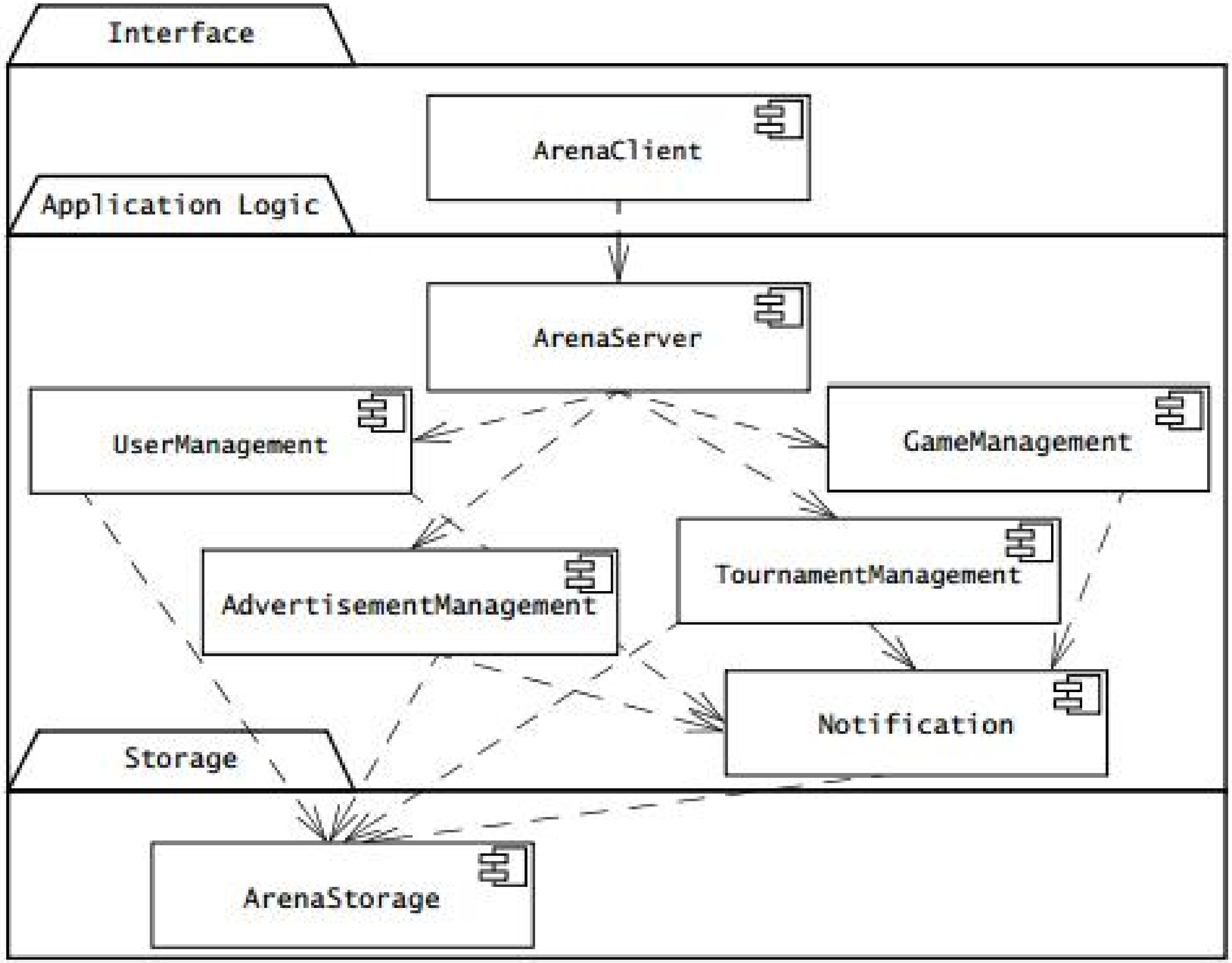
Architectural styles

Three-tier



Architectural styles

Three-tier



Contents

1. **Component diagram**
   a. Introduction
   b. Component: defining and notation
   c. Types of Interfaces
   d. Connecting components
   e. Classes within a component
   f. Ports and internal structures
   g. Types of Connectors
   h. Black-box and White-box views
2. **Architectural styles**
3. **Decomposing system**

Identifying subsystems

- **How to group objects into subsystems**
  - Assign objects identified in one use case into the same subsystem.
  - Create a dedicated subsystem for objects used for moving data among subsystems.
  - Minimize the number of associations crossing subsystem boundaries.
  - All objects in the same subsystem should be functionally related.

Bài tập

- **Phân rã hệ thống đã thiết kế thành các thành phần (components). Sử dụng công cụ vẽ lược đồ component cho hệ thống**
- **Lưu ý:**
  - Xác định rõ các interface của các component
  - Xác định rõ các class trong từng component
  - Xác định rõ mối quan hệ giữa các component