Software Engineering Course

Chapter 6

Chapter 6: Other UML concepts and diagrams

2022-2023

Dr. Sofia Kouah

Plan

- Component Diagram
- Deployement Diagram
- Composite Structure Diagram

Component Diagram

Component Diagram

- The **component diagram** allows you to model the **software architecture** of a computer system of arbitrary size and complexity.
- It shows its **structuring** in terms of **software components** and **dependencies** between these components.
- This structuring is specified at a lower level of granularity than classes.

Component Diagram

- The **component diagram** also presents **a static view of the system** implementation while illustrating the implementation choices.
- It allows you to <u>specify the integration of third-party software</u> components, such as EJB, CORBA, COM+ or .Net, WSDL, etc. components.
- In addition, it allows the **identification of <u>reusable</u> components**.

Elements of the Component Diagram

The basic elements of the Component Diagram:

- A component diagram consists of the following elements: **component**, **relationship** or **dependency**, **interface**, and **port**.
- These concepts are intrinsically **interconnected.**

Component

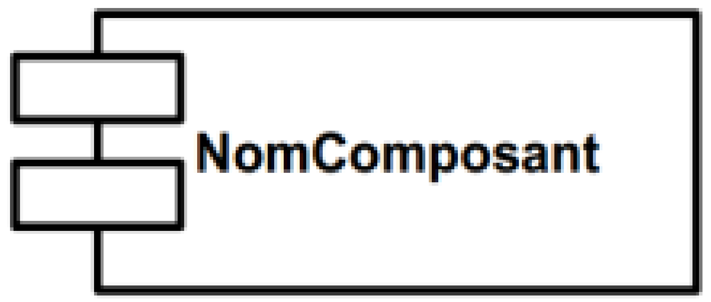
- A component is a **software unit** offering services **through one or more interfaces.**
- It represents **a modular part of a system**, which encapsulates its content and which is **replaceable in its environment**.
- A component describes the **physical implementation of a software component**, which can be <u>code</u>, a <u>script</u>, a <u>batch file</u>, a <u>data file</u>, a <u>data table</u>, an <u>HTML page</u>, etc.
- A component can be specified by:
- - A **name** that distinguishes it from other components. The full name of a component, which includes the package names and the component name, is unique.
- - A **specification of the interfaces** required and offered.
- - and a **set of connection ports**.

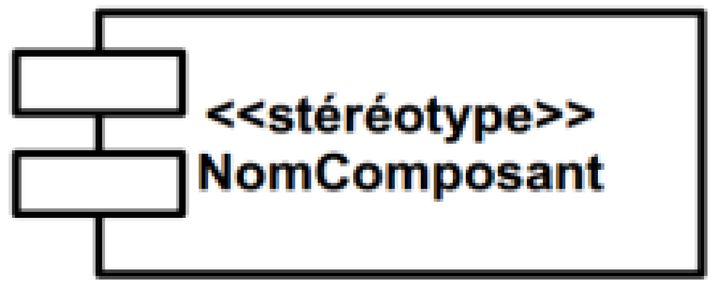
Component Vs Class

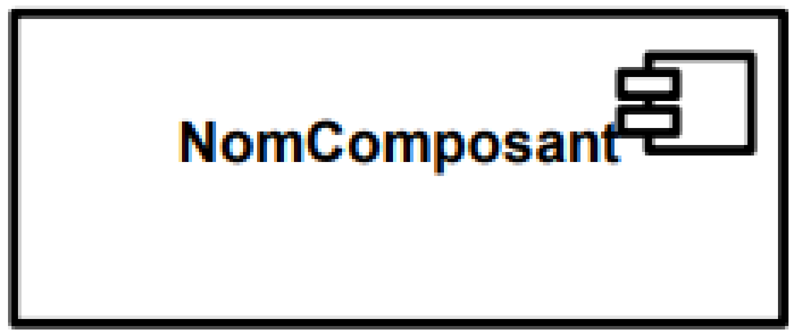
Component versus Class (respectively object)?

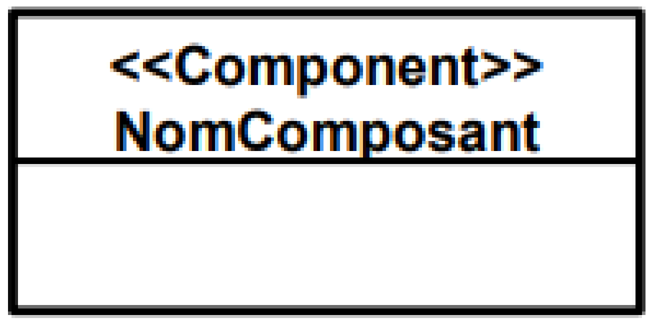
- A **class that implements one or more interfaces is a component**.
- Conversely, a **component is not necessarily a class**.
- Furthermore, since the **object** is a **class instance**, we can say that the **notion of component is close to that of the object** (i.e. class) in terms of **modularity and reuse** but at a lower level of granularity.
- We **speak of component in a software architecture context** and **of object in a code architecture context.**

Several representations of Component







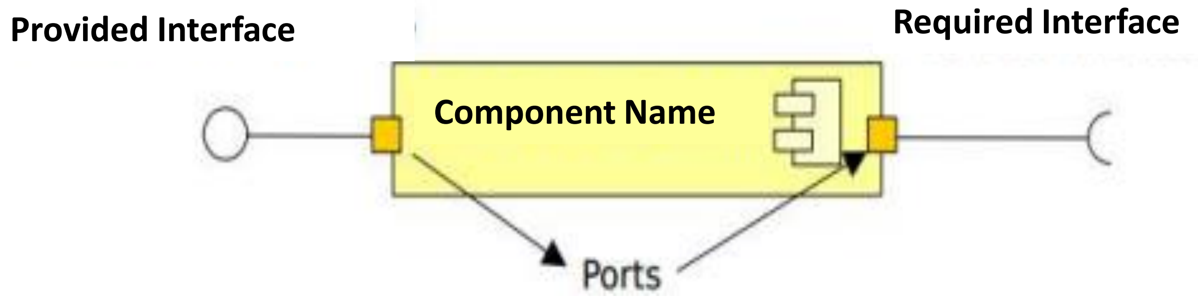


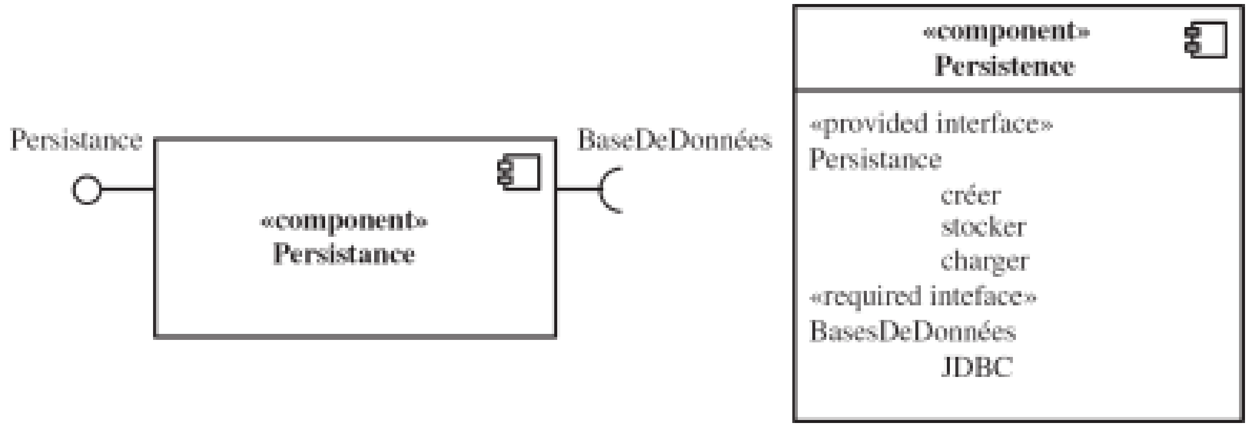
Interface

- Une interface est **une collection de spécifications d'opérations** qui définit les **services rendus** par les composants ou les classes.
- Elle constitue le **moyen d’interaction entre les composants**.
- Two types of **interfaces are distinguished**, namely: **provided interfaces and required interfaces**.
- A **provided interface** is an **interface whose component itself provides a service to other components**.
- An interface required by a component is an interface that **provides the service that that component needs to function**.
- The definition of required or provided interfaces of a component can involve the **notion of port.**

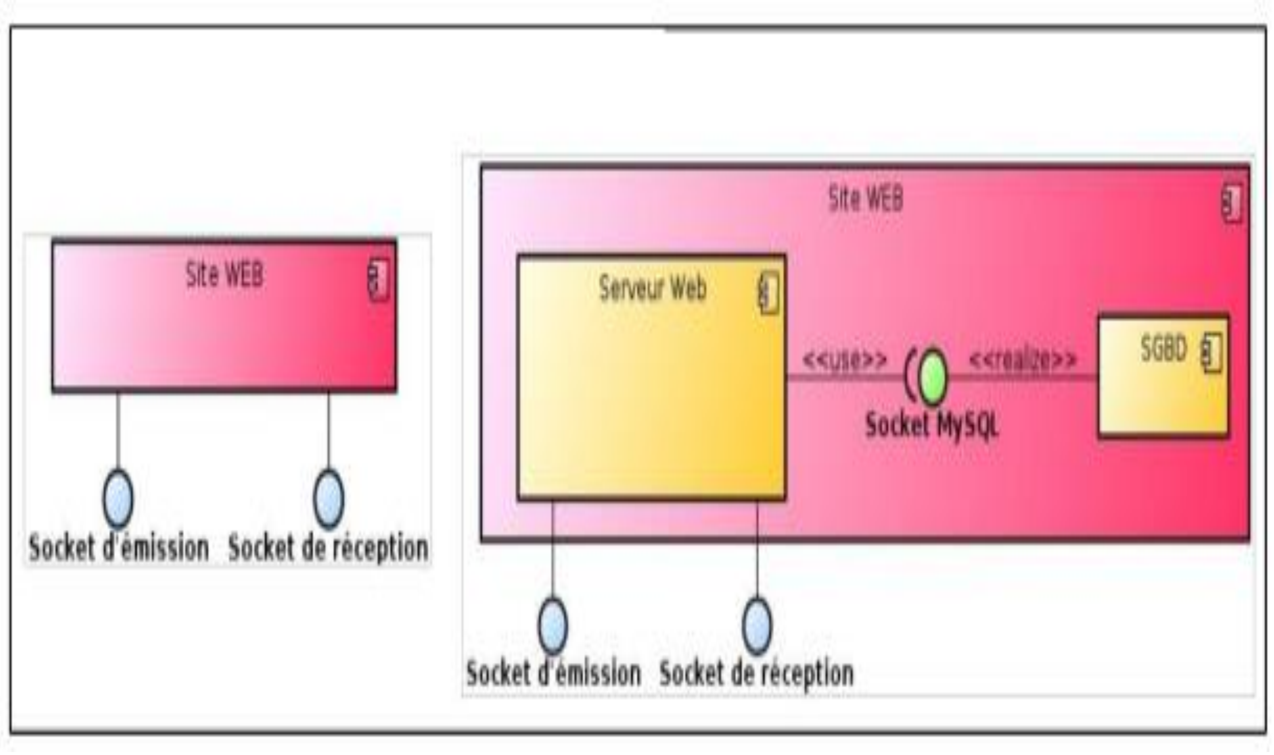
Port

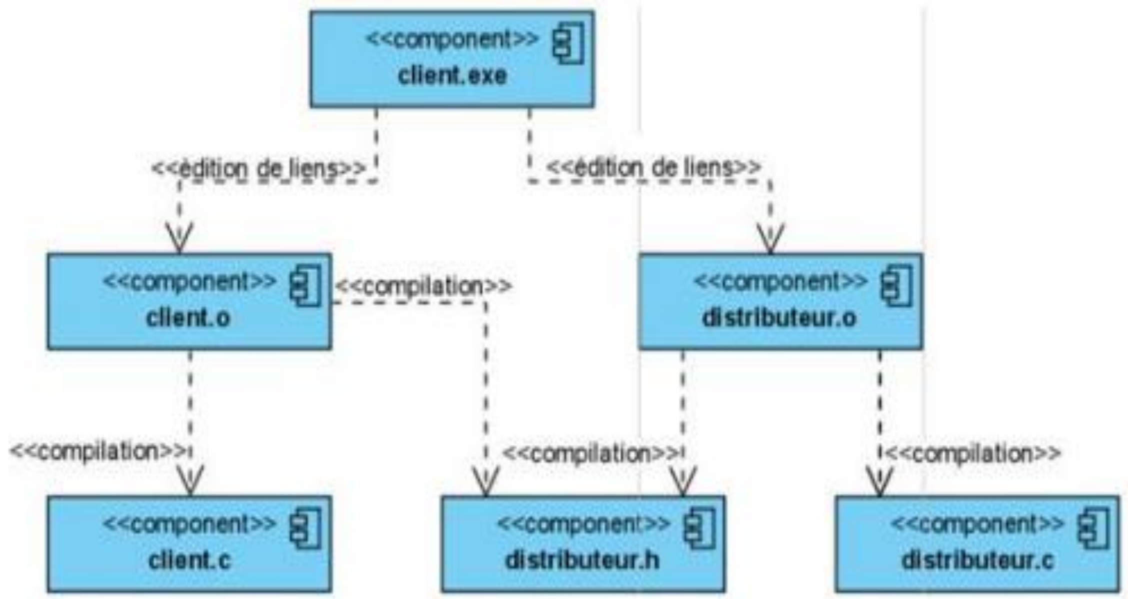
- **The port of a component** represents the connection point between the component and its environment.
- It is associated either with an offered interface or with a required interface.
- It also ensures a certain independence between the component and its external environment such that the internal structure of the component can be modified without affecting the external components.



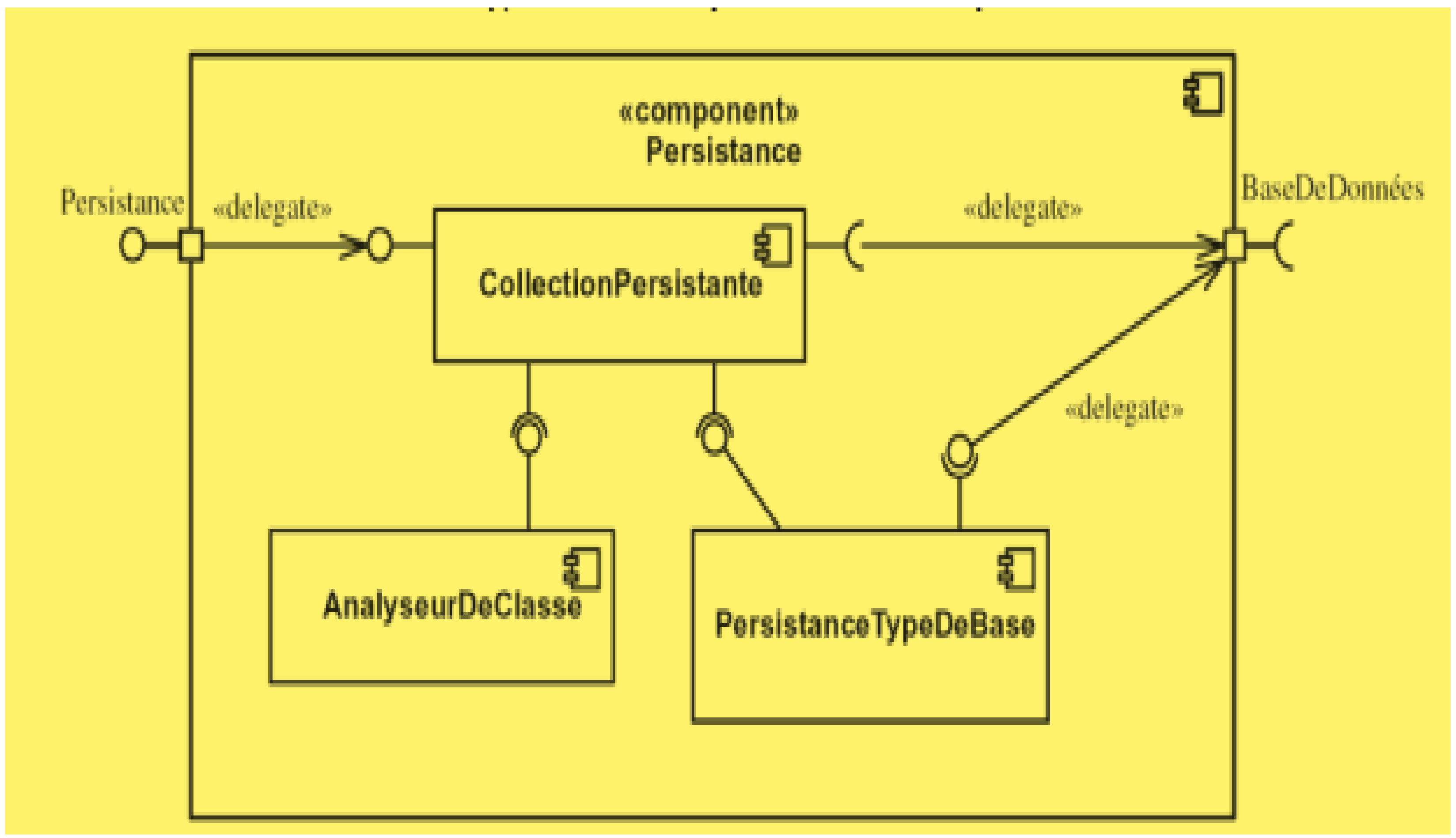


Example


Site WEB
Socket d'émission
Socket de réception
Site WEB
Serveur Web
<<use>>
<<realize>>
SGBD
Socket MySQL
Socket d'émission
Socket de réception


<<component>>
client.exe
<<édition de liens>>
<<édition de liens>>
<<component>>
client.o
<<compilation>>
<<component>>
distributeur.o
<<compilation>>
<<compilation>>
<<compilation>>
<<component>>
client.c
<<component>>
distributeur.h
<<component>>
distributeur.c

Example


«component»
Persistance
Persistance
«delegate»
CollectionPersistante
«delegate»
BaseDeDonnées
«delegate»
AnalyseurDeClasse
PersistanceTypeDeBase

Deployemnet Diagram

Deployment diagram

- The **deployment diagram** helps describe the hardware architecture of the system.
- It models the physical distribution of the hardware resources constituting the system, such as: workstations, printer, scanner, Modem, Server, etc.
- Thus, it **models the relationships and interconnections between these resources** while **specifying the deployment of software on hardware components**.
- This allows us to **describe the relationship between Hardware and Software**.

Deployment diagram

- The **deployment of an IT system** <u>precedes</u> its putting into production.
- It constitutes an important stage in its development.
- Once developed, a system will be executed on physical resources (processors, memory, etc.) in a specific and well-determined environment.
- The deployment diagram makes it possible to represent such environments (execution environment with physical resources).

Deployment diagram components

- The basic elements of the Deployment Diagram A deployment diagram consists of the following elements: **Nodes**, **associations** or **dependencies** and **artifacts**.
- **Node**: A node is a physical entity which occur during the execution phase. It models an execution resource on which artifacts can be deployed. Each node has memory, computing capacity, and other features that allow it to receive and execute software.

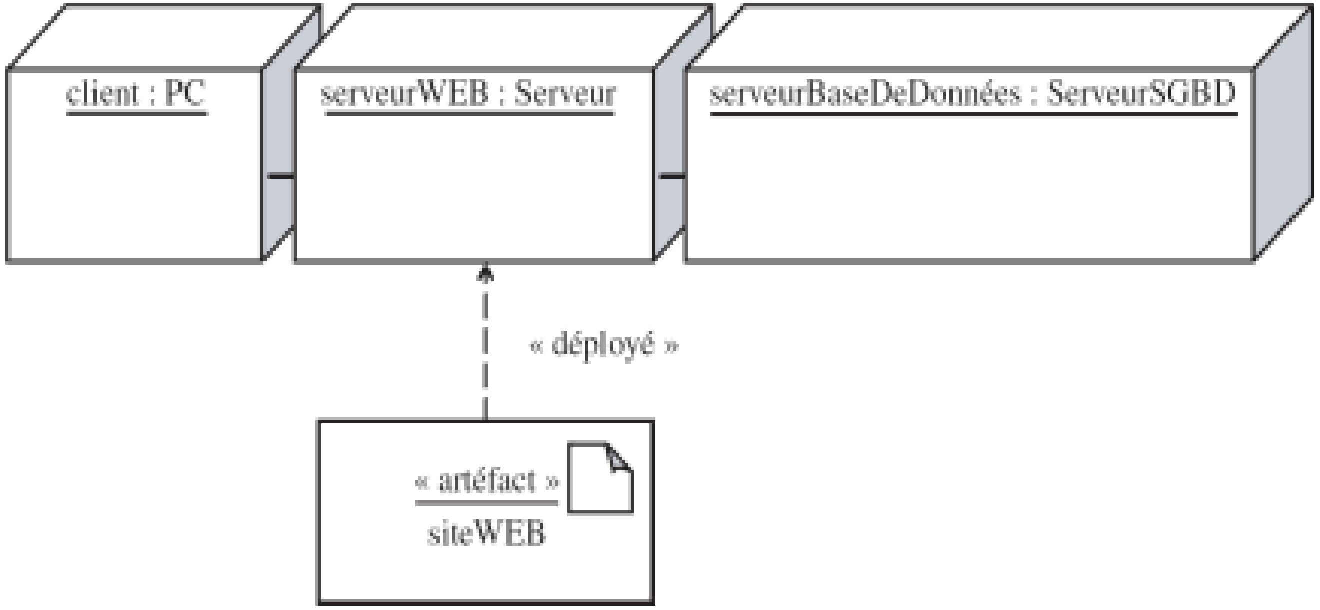
Deployment diagram

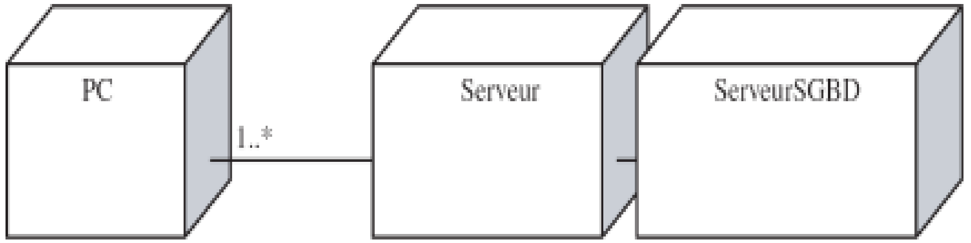
- Two types of nodes can be distinguished:
- - A **physical execution support** which can itself be composed of sub nodes.
- - An **execution environment** that is specific for certain components, such as: operating system, Java Virtual Machine, Servlet, Web Server.
- **Association**: An association between two nodes models the communication path (i.e. the connection) which **allows them to exchange information**.

The association can be **stereotyped** by the communication protocol used between the nodes, such as TCP/IP, UDP/IP, RS232, Ethernet. Thus, multiplicities can be attached to the association.

Deployment diagram

- **Artefact** : Un artéfact (Artifact en anglais) est une partie du système qui s’exécute sur un nœud. Il peut prendre plusieurs formes, entre autres : un fichier binaire exécutable, une bibliothèque partagée, un fichier source, une table de base de données, un courrier électronique, un document, un script, etc. 3.2. R



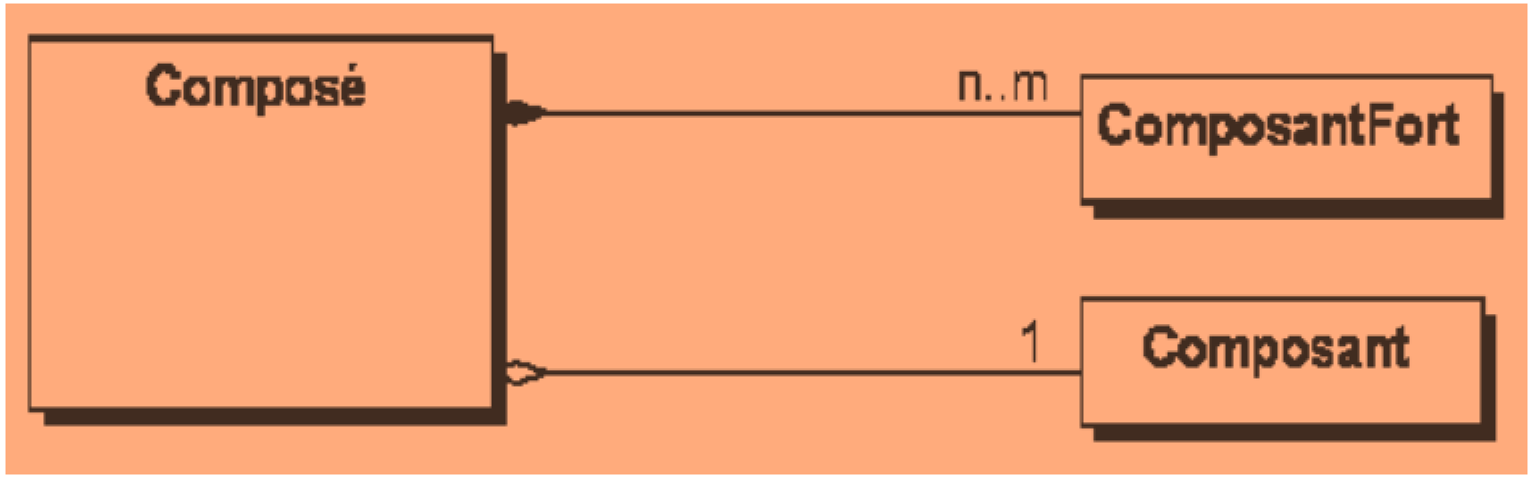


Composite Structure Diagram

Composite Structure Diagram

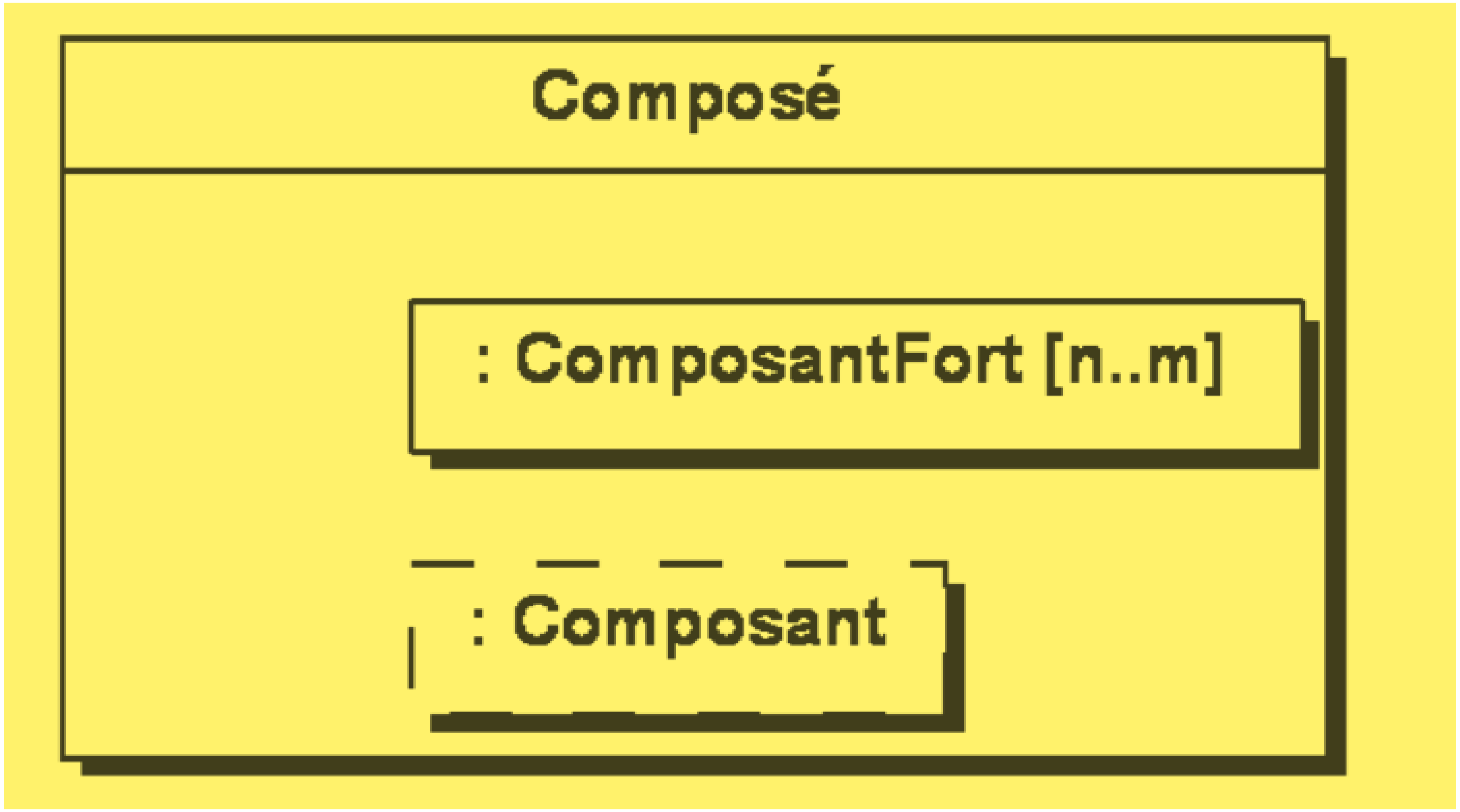
- The primary objective of the composite structure diagram is to **precisely describe a composite object** . Such a diagram is not **intended to replace** the class diagram **but to complement it.**
- In the composite structure diagram **, the compound object** is described by a **classifier** while its components are described by **parts** .
- A classifier and a part are associated with a class, the complete description of which is carried out in a class diagram.

Composite object



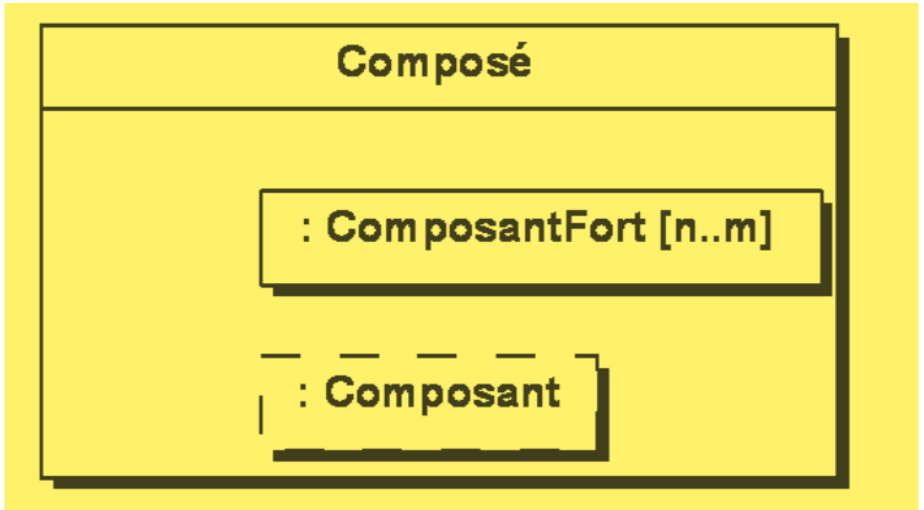
This composite object has one component from a **strong composition** and another from an **aggregation** .

Composite object

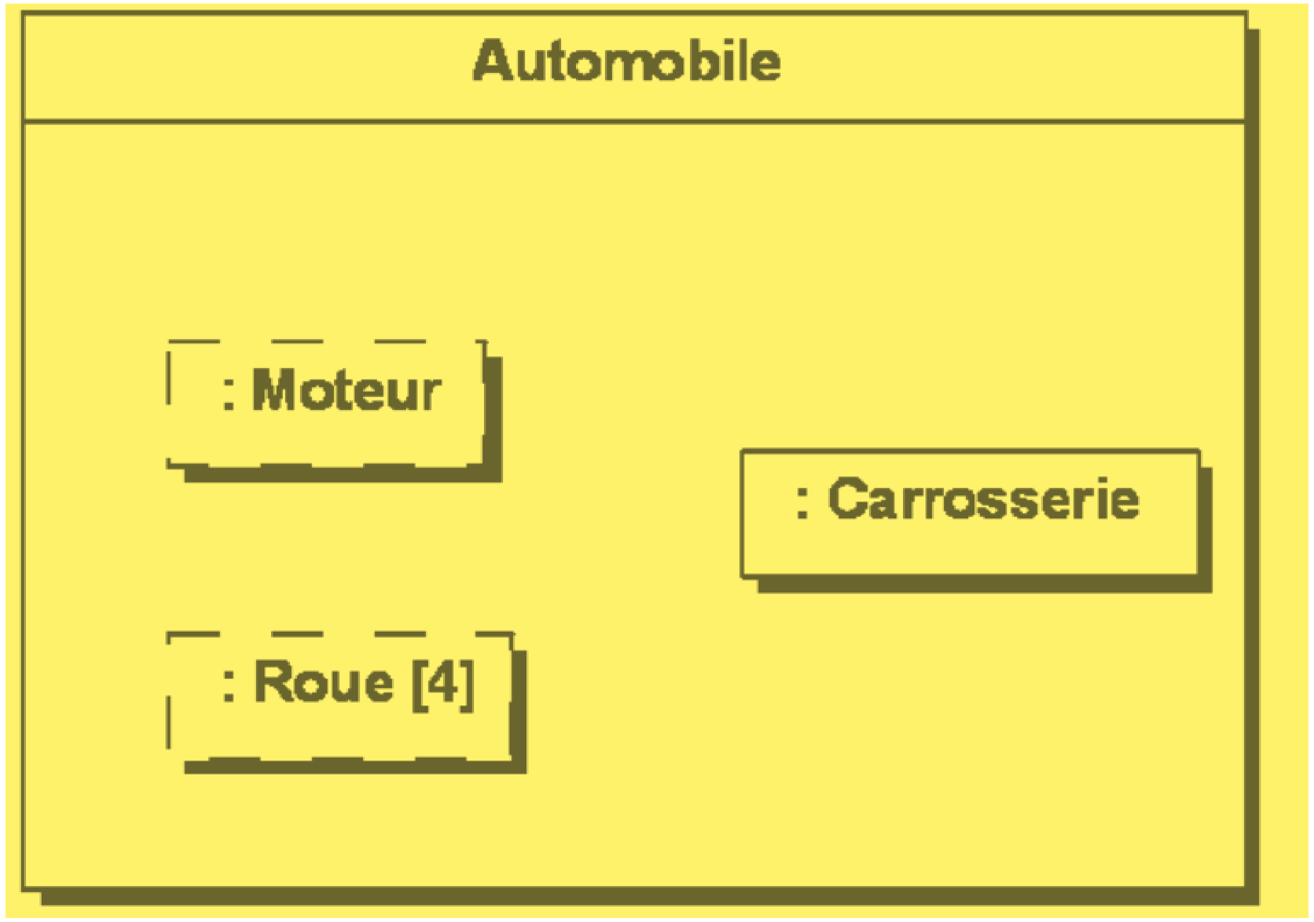


Composite object

- This figure shows the composite structure diagram corresponding to this object.
- The components are integrated within the **classifier** which describes the composed object.
- The **parts** have a **type** which is the class of the component .
- The cardinality is indicated in square brackets. By default, it is one.
- A component resulting from an **aggregation is represented by a dotted line** , a component resulting from a **strong composition is represented by a continuous line.**



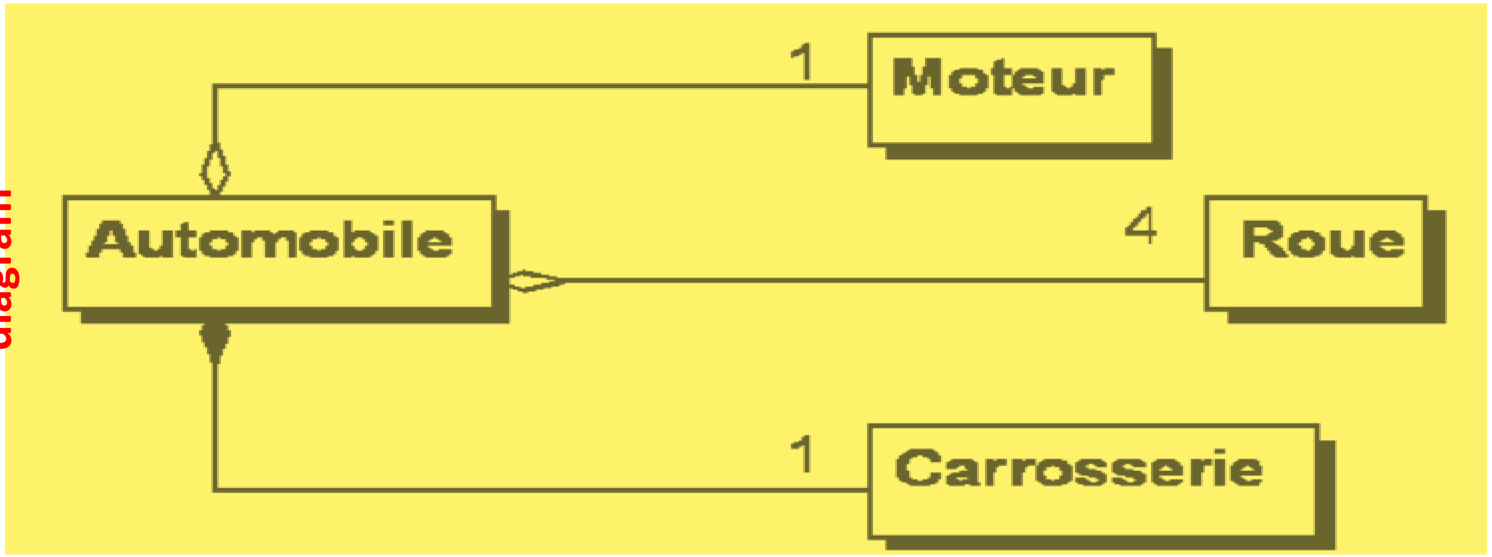
Example



Example (continued)

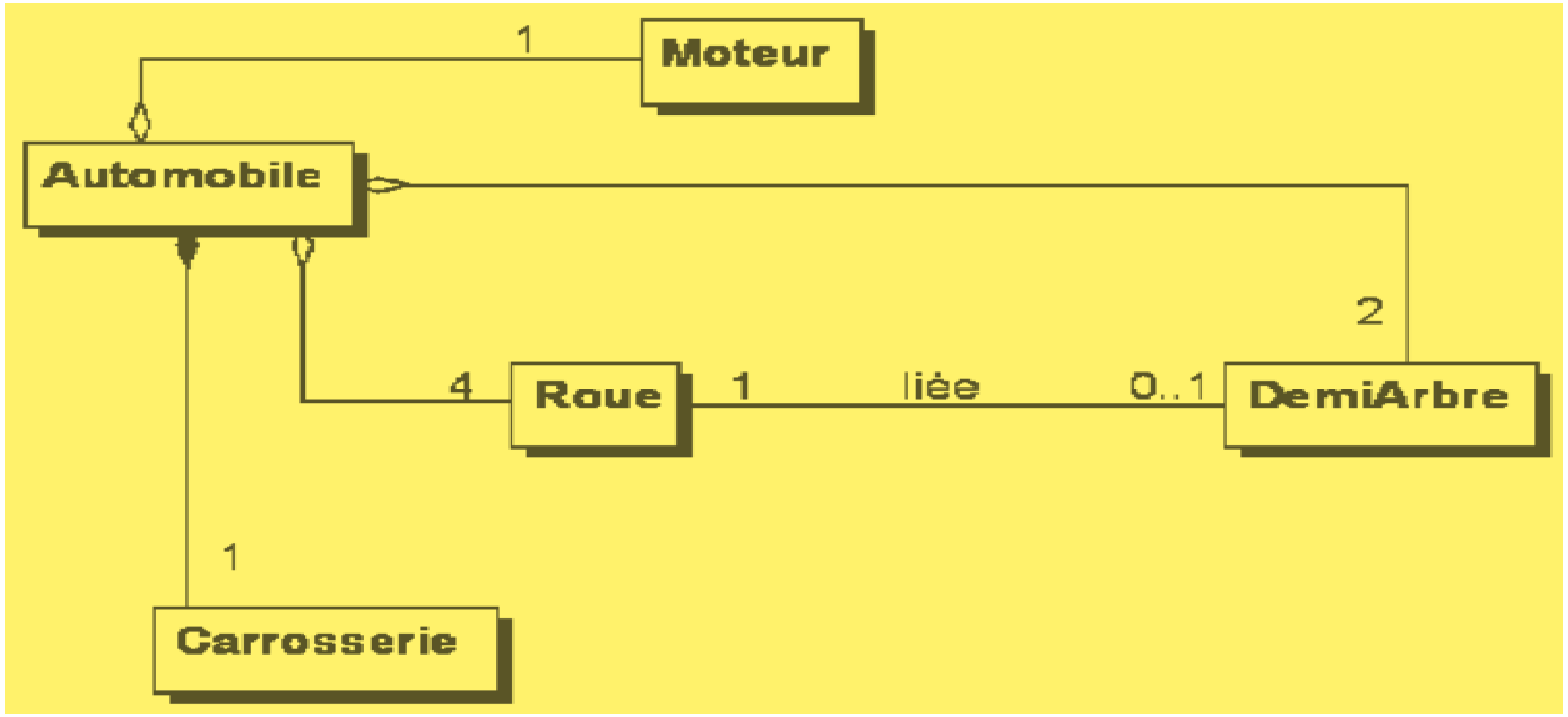
- This figure shows an example of a composite structure diagram describing an **automobile** as a composite object .
- The corresponding class diagram is shown below.
- In the composite structure diagram, Engine, Body and Wheel **are not classes** but **parts** .

Corresponding class diagram

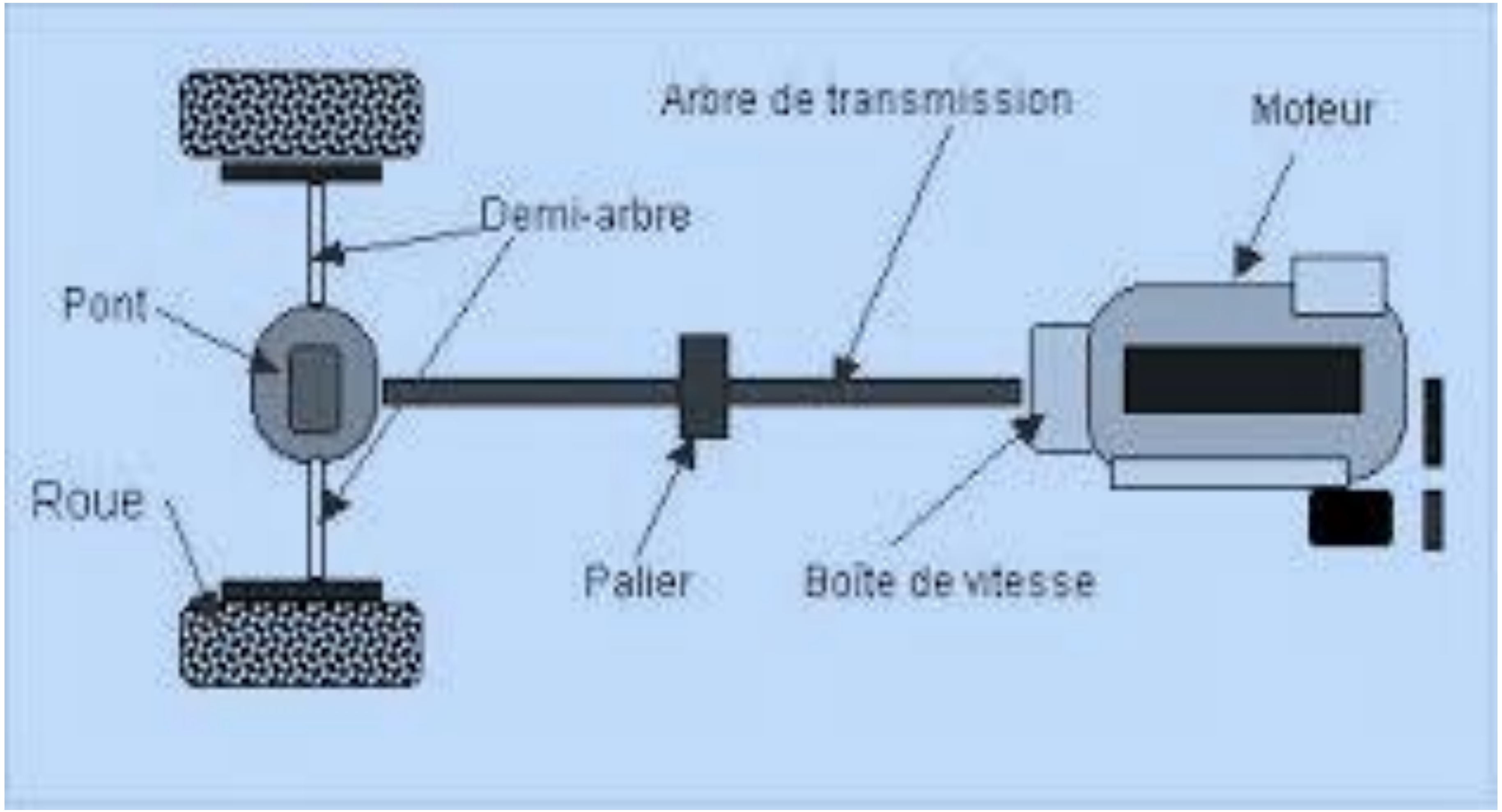


Detailed example

- The following class diagram again illustrates an automobile as a composite object.
- It introduces **the *linked association* between the wheels and the half shafts (DemiArbre)** which ensure **the transmission between the engine and the front wheels** , which are the drive wheels.



Detailed example



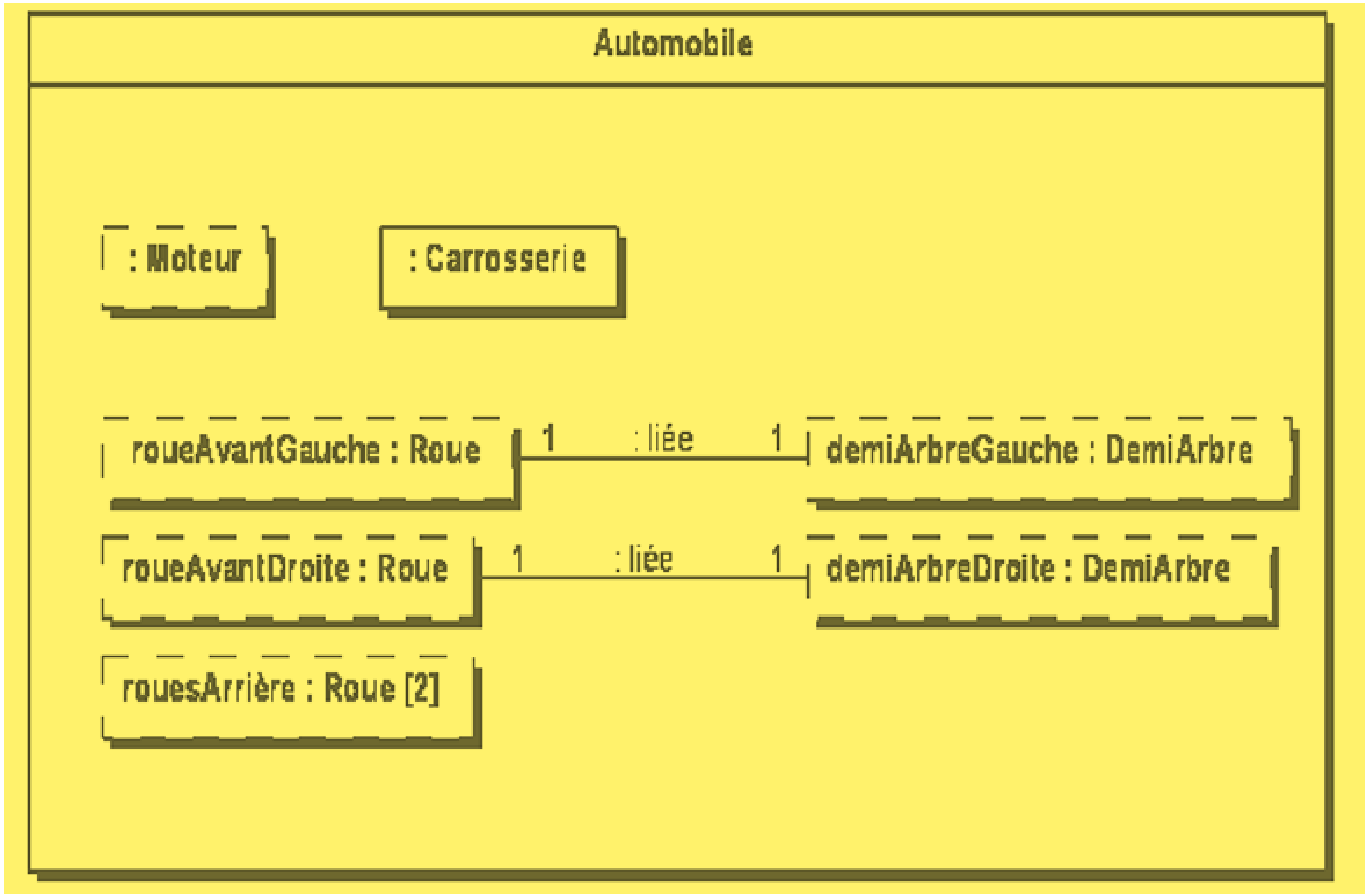
Detailed example (Continued)

- The cardinality of the *linked association* is 0..1 at the end of the half tree.
- Indeed, the cardinality is one for the front wheels and **zero for the rear wheels** . This last information **cannot be described in the class diagram , unless two** *Wheel* subclasses are introduced : *FrontWheel* and *RearWheel* .
- However, introducing two subclasses to specify a cardinality has the disadvantage of making the class diagram heavier. This possibility is not desirable.

Interest in the composite structure diagram

- The composite structure diagram allows you to **specify the role of a part** . **The role describes the use of the part within the compound object.**
- The figure below introduces **three parts for the wheels** corresponding to the left front wheel, the right front wheel and the two rear wheels. The cardinality of the parts is adapted accordingly. The name of the role is indicated before that of the type in the part.
- This notation is not the same as that used in the object diagram. This is because the notation for specifying an instance **uses the underlined style.**

Composite structure diagram with roles and connectors



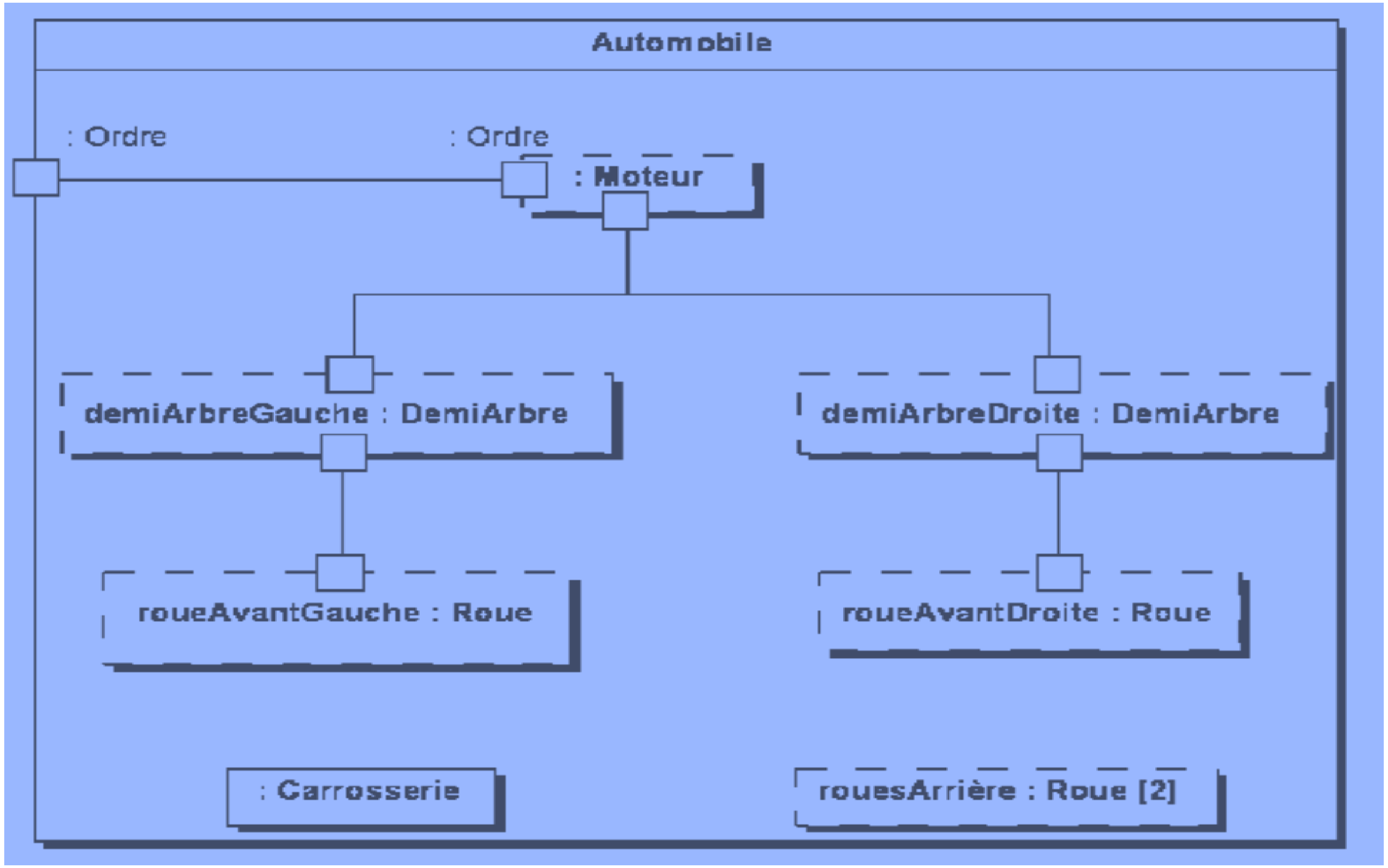
Composite Structure Diagram with Roles and Connectors (continued)

- In this figure, two parts are also introduced, corresponding to each half shaft. Between each part corresponding to a front wheel and each part corresponding to a half shaft, a connector represents the *linked association* .
- A connector connects two parts. It is typed by an inter-object association, just as a part is typed by a class.

Composite Structure Diagram with Roles and Connectors (continued)

- Connectors can also connect parts together through **ports** . A port is a point of interaction.
- It has an interface which constitutes its type and defines all possible interactions. The interactions carried out by a port are carried out with the other ports linked to it by a connector.
- A port can also be introduced at the classifier level. Such a port then aims to serve as a **gateway between the internal parts of the classifier and external objects.** to this one.
- From an encapsulation point of view, such a port is usually **public** . It is then known outside the classifier.

Composite structure diagram introducing ports



Composite structure diagram introducing ports

- The previous figure illustrates the same decomposition into parts of the *Automobile object* as in the last example.
- Connectors have been added between the **motor and half trees.** Each connector between parts is connected through a port which is in the form of a white square. A port has also been added at the classifier level. It is typed by the *Order interface* . It is connected to a motor port also typed by this interface.

Composite structure diagram introducing ports

- In terms of interactions, this figure illustrates:

  - that the *Automobile class* can interact with the outside to receive orders intended for its engine and which are transmitted to it;

  - that the engine communicates with the half shafts (transmission of movement);

  - that each half shaft communicates with the wheels (transmission of movement).

Composite structure diagram introducing ports

- Connectors are not typed for reasons of simplification.
- Likewise, the interface of the half-shaft ports, the wheels and the motor port connected to the half-shafts has not been indicated. This could be the Transmission interface.