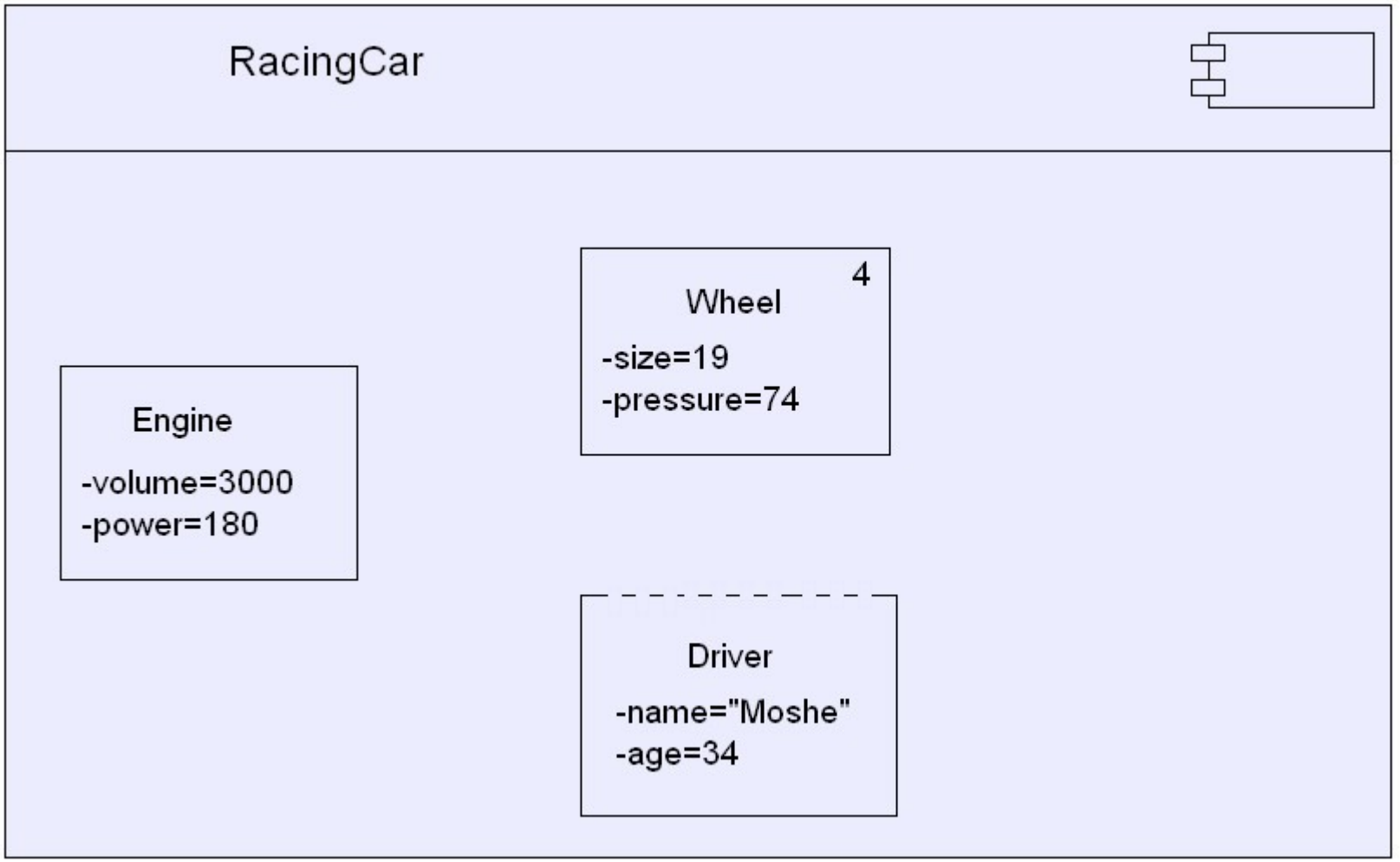
UML Composite Structure Diagrams

Introduction

- Similarly to UML Component Diagram, the UML Composite Structure Diagram focuses on showing the internal structure of a specific classifier.
- Many of the notations known from UML Component Diagram can be used within a UML Composite Diagram as well.
- While the purpose of the Component Diagram is showing the internal structure, the purpose of the Composite Structure Diagram is showing how a specific funcionality is implemented.

Introduction



Connectors

- A connector represents a communication link between two objects.
- A connector can be an object that represents an association or a value we hold in a variable. A value held in a variable is another way to represent a connection between two objects.

Connectors

- The notation for connector is a solid line. For each connector we can provide a name and a type in the following format:

  `name:classname`

  `name`

  The name of the connector

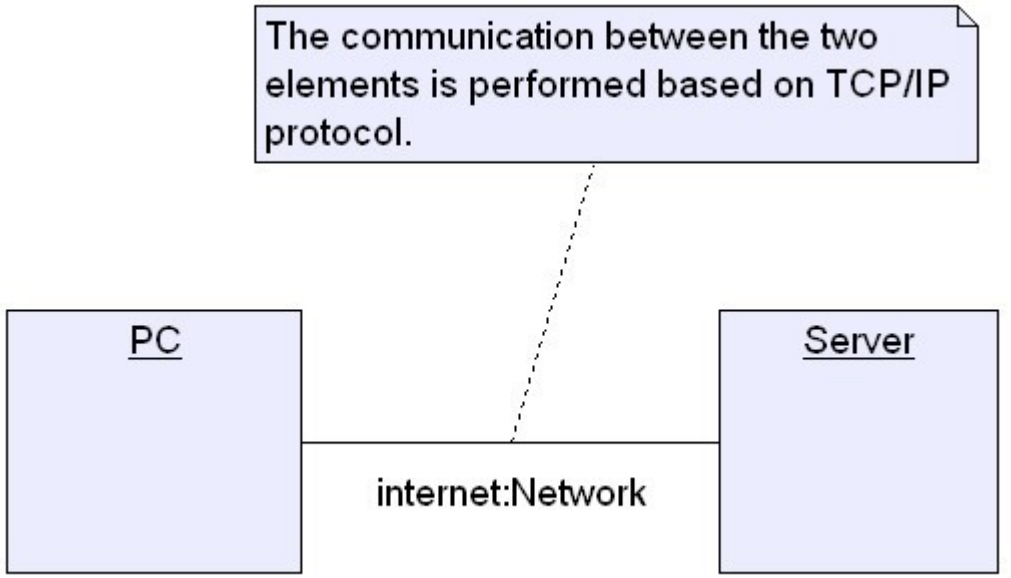
  `classname`

  The name of the class that was instantiated to represent an association. Relevant when the connector represents a link between two objects... a link that is represented by object of a specific class... an object that represents the link as an association.

Connectors

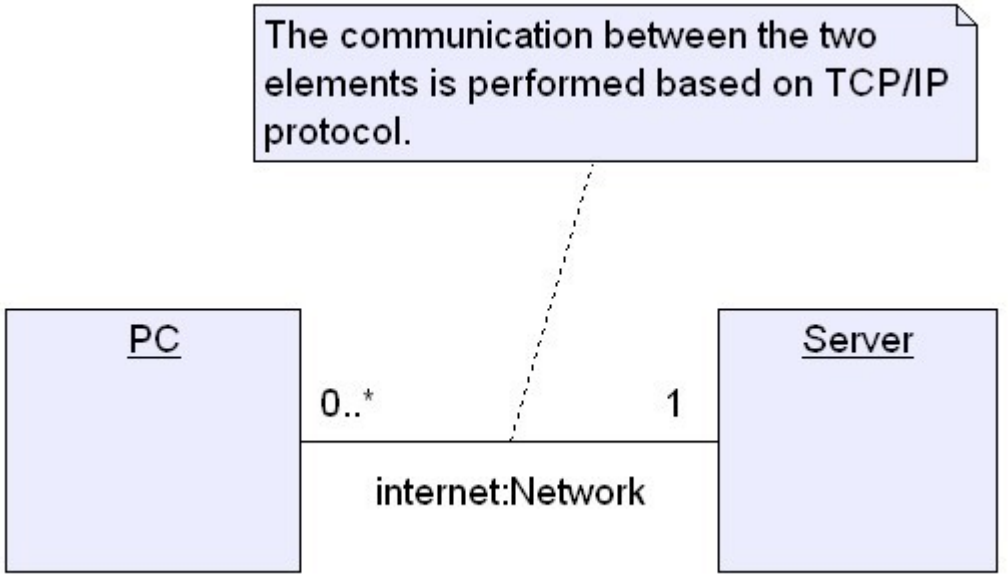
More Information

- Using a simple note it is possible to provide more information about the connector, which means: more information about the communication between the two elements (performed via the connector).



Connector's Multiplicity

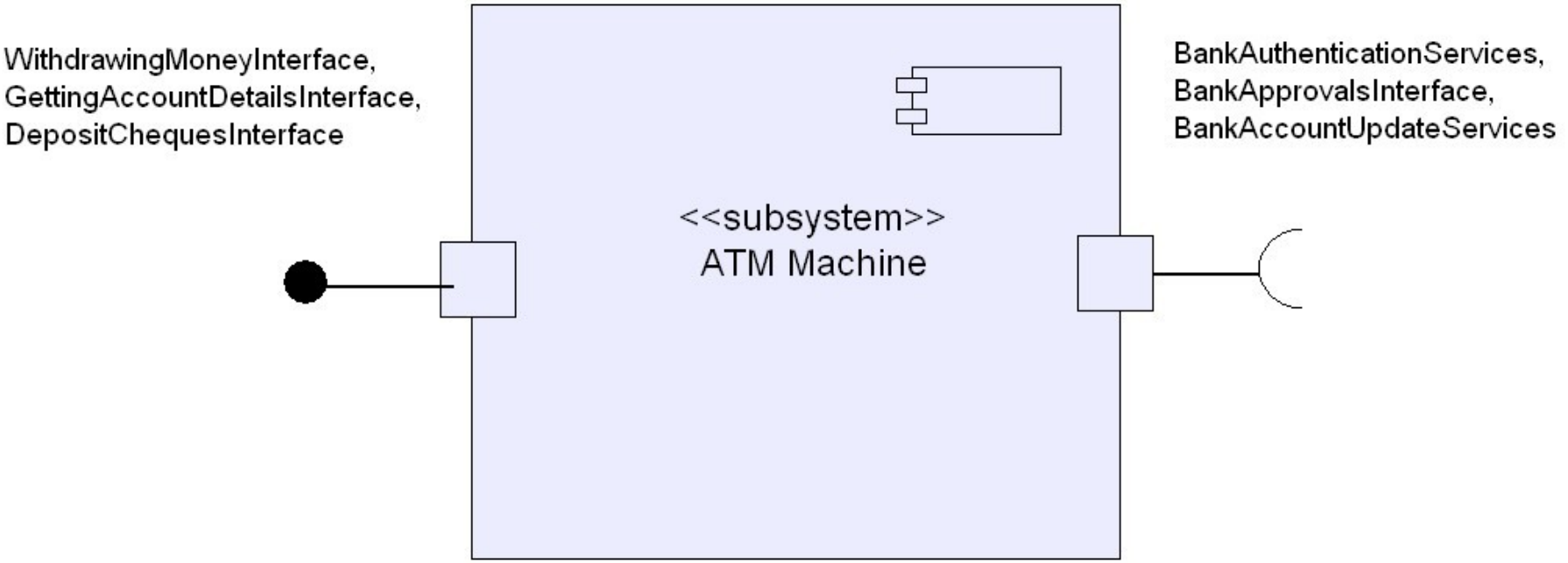
- Using the formal multiplicity syntax it is possible to specify the multiplicity of each connector end.



The Ports

- A port is a way to offer functionality from a composite structure without exposing details about that composite structure internal implementation.
- The port notation is a small square drawn above the edge line of the boarder.
- The port name and its multiplicity are written near to it.

The Ports



Required and Provided Interfaces

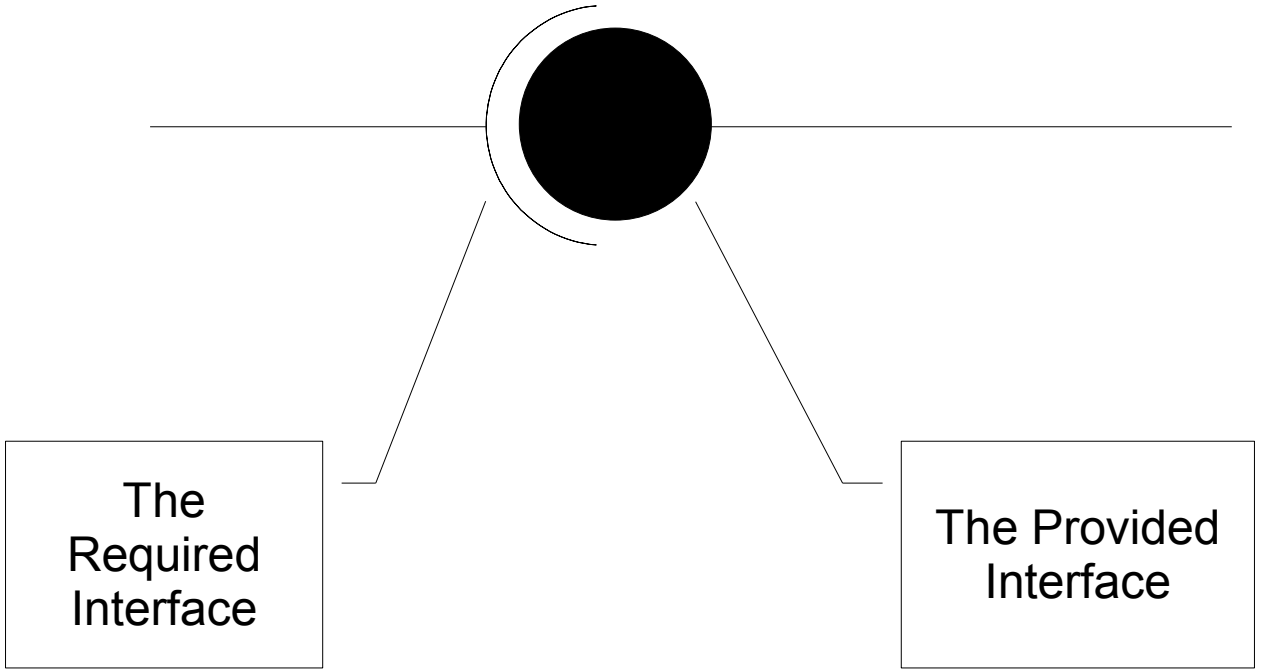
- The ports are associated with required/provided interfaces from/to the environment.

  Example:

  An ATM machine might provide an interface to withdraw money... at the same time it might require an interface to the bank system through which the withdraw could be approved and updated on the bank system data base.

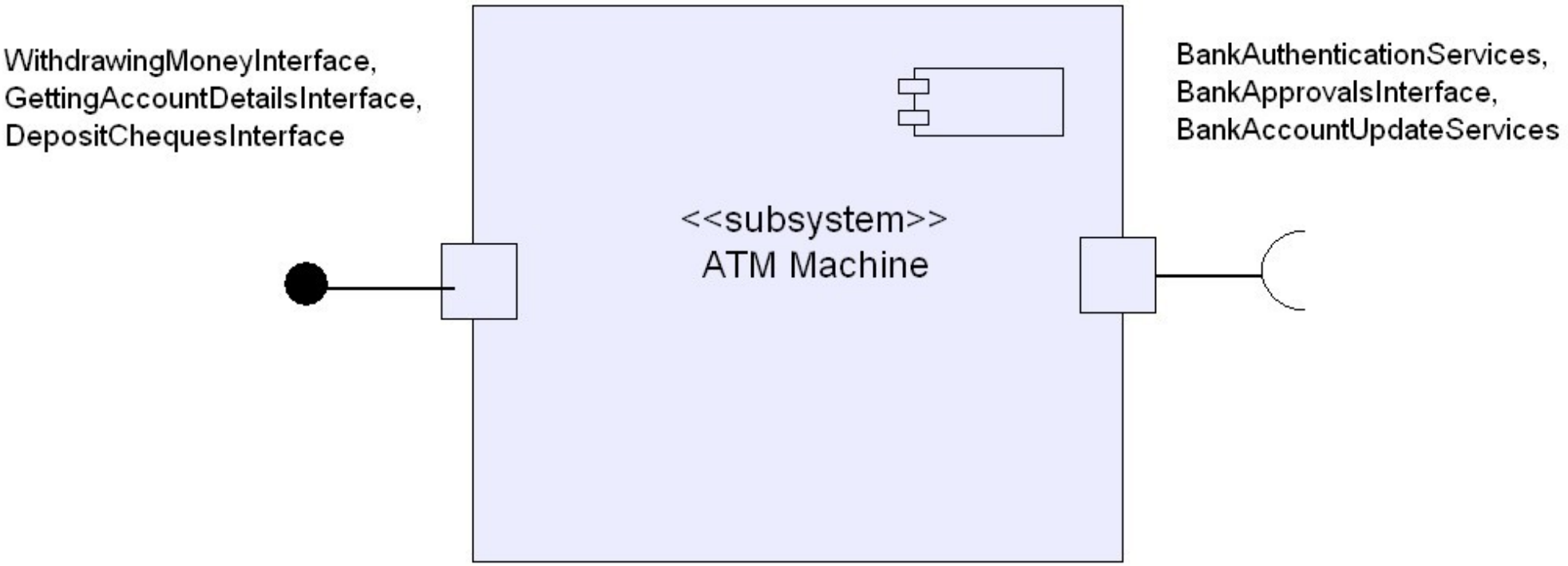
Required and Provided Interfaces

- The required/provided interfaces are usually depicted using the following notation:



Required and Provided Interfaces

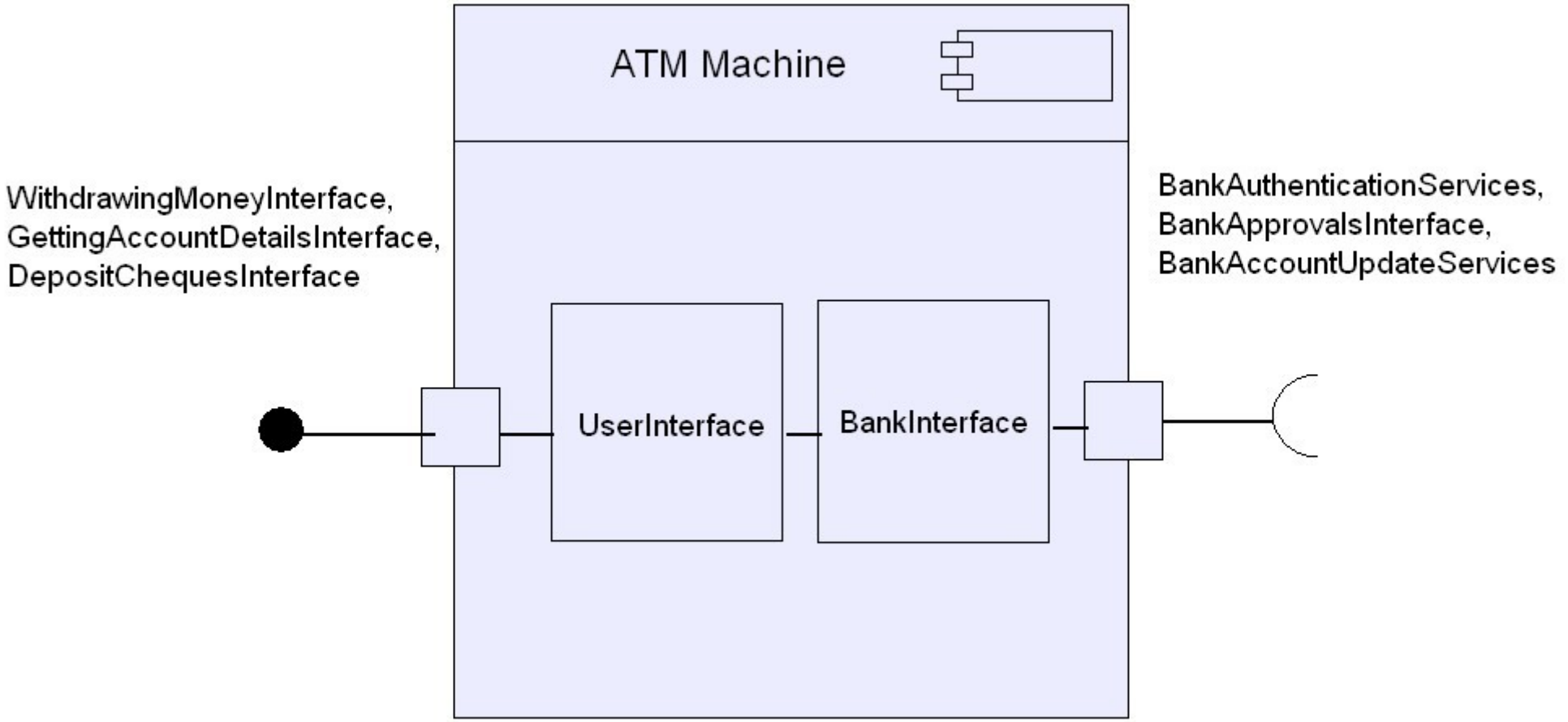
- If there are multiple required/provided interfaces we can write them comma separated near the relevant socket/ball.



Ports Implementation

- Each port is wired to its internal implementation using connectors.
- The internal implementation can be some code owned by the classifier (this is usually the case when dealing with small classes). In such case the port will be named “Behavioral Port”.
- If the implementation of the port is done by another inner element then we will link the port to its internal implementation.

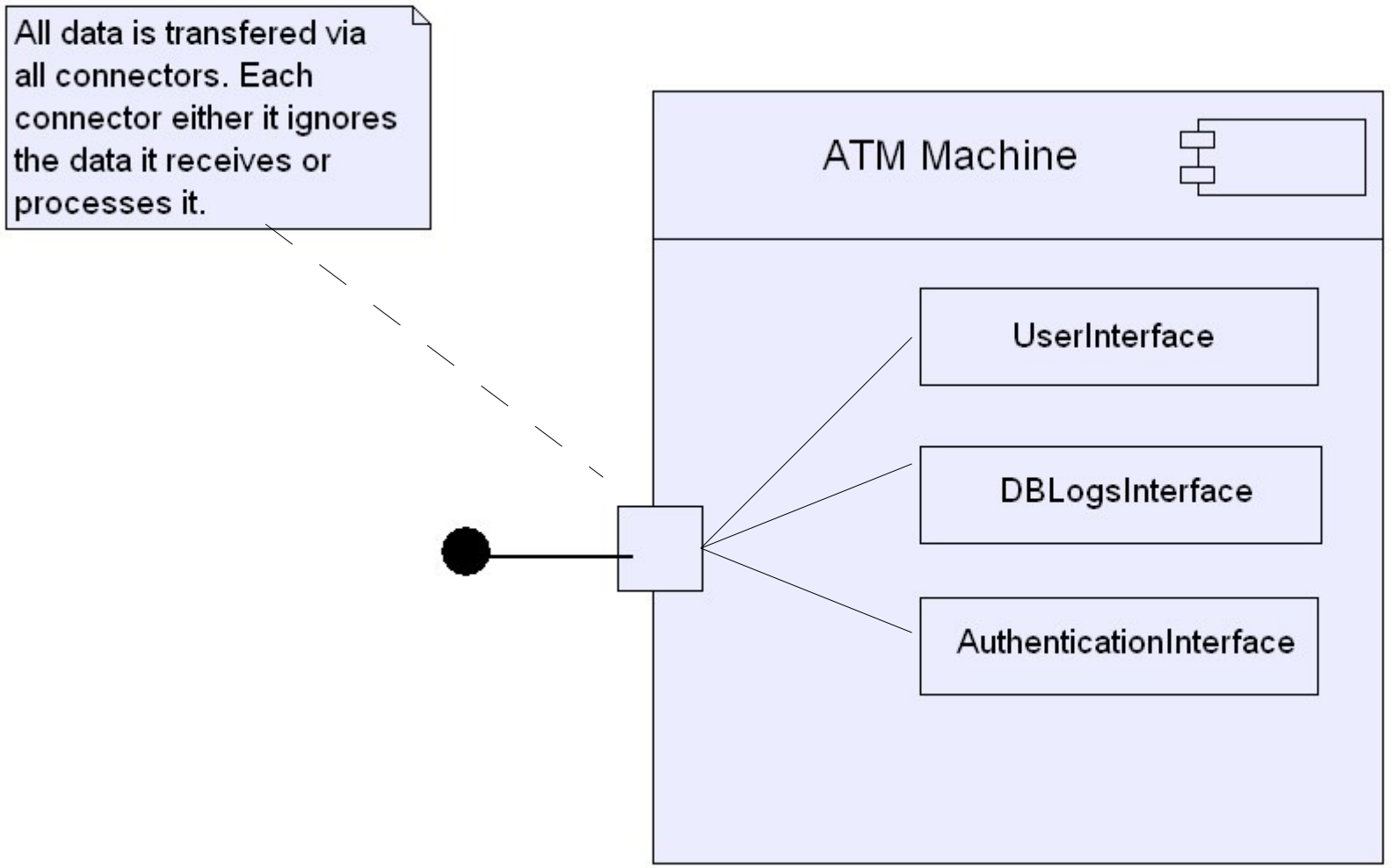
Ports Implementation



One Port & Multiple Connectors

- UML specification allows having one port connected with multiple connectors.
- When this is the case it is important to add a note with explanation about how does the data that arrives from the environment continues its flow to the connectors... is it duplicated for each connector.. is there a mechanism that selects which data to forward through which connector.. etc.

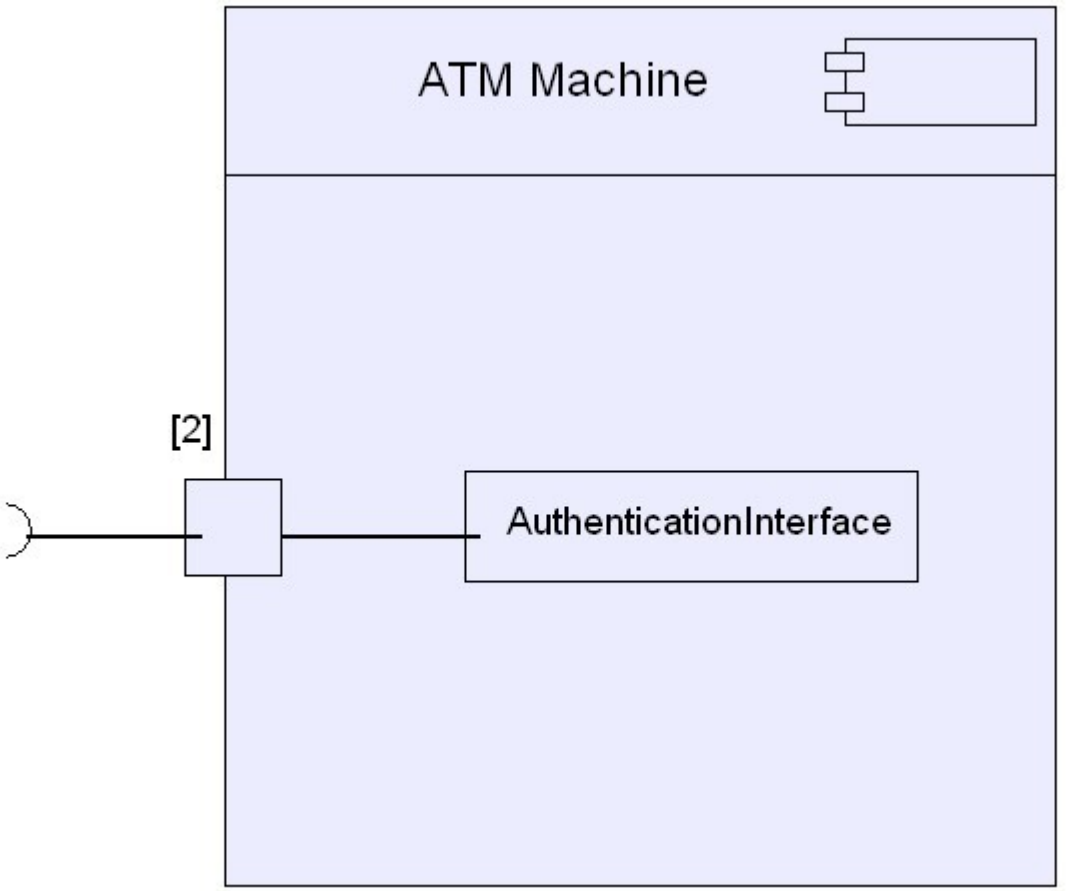
One Port & Multiple Connectors



Multiple Ports

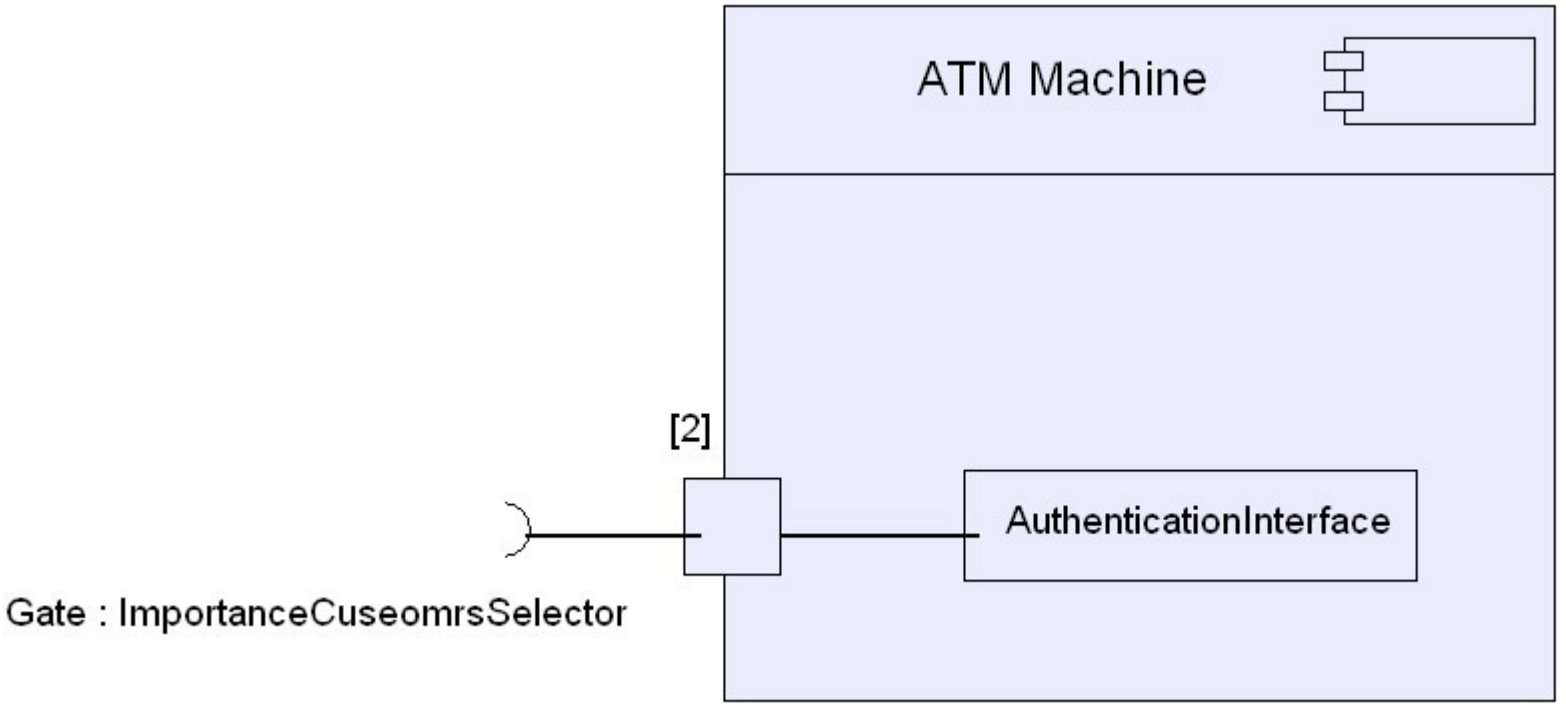
- Having more than one port for providing/receiving the same interface/s is feasible. In such case we will specify the number of ports near the port notation.
- Having multiple ports can happen when we have two instances of two different classes that both of them implement the interface that was declared for that port (e.g. AuthenticationInterface was implemented twice... first in a class that authenticates regular customers... and second is a class that authenticates premium customers.

Multiple Ports



Ports' Types

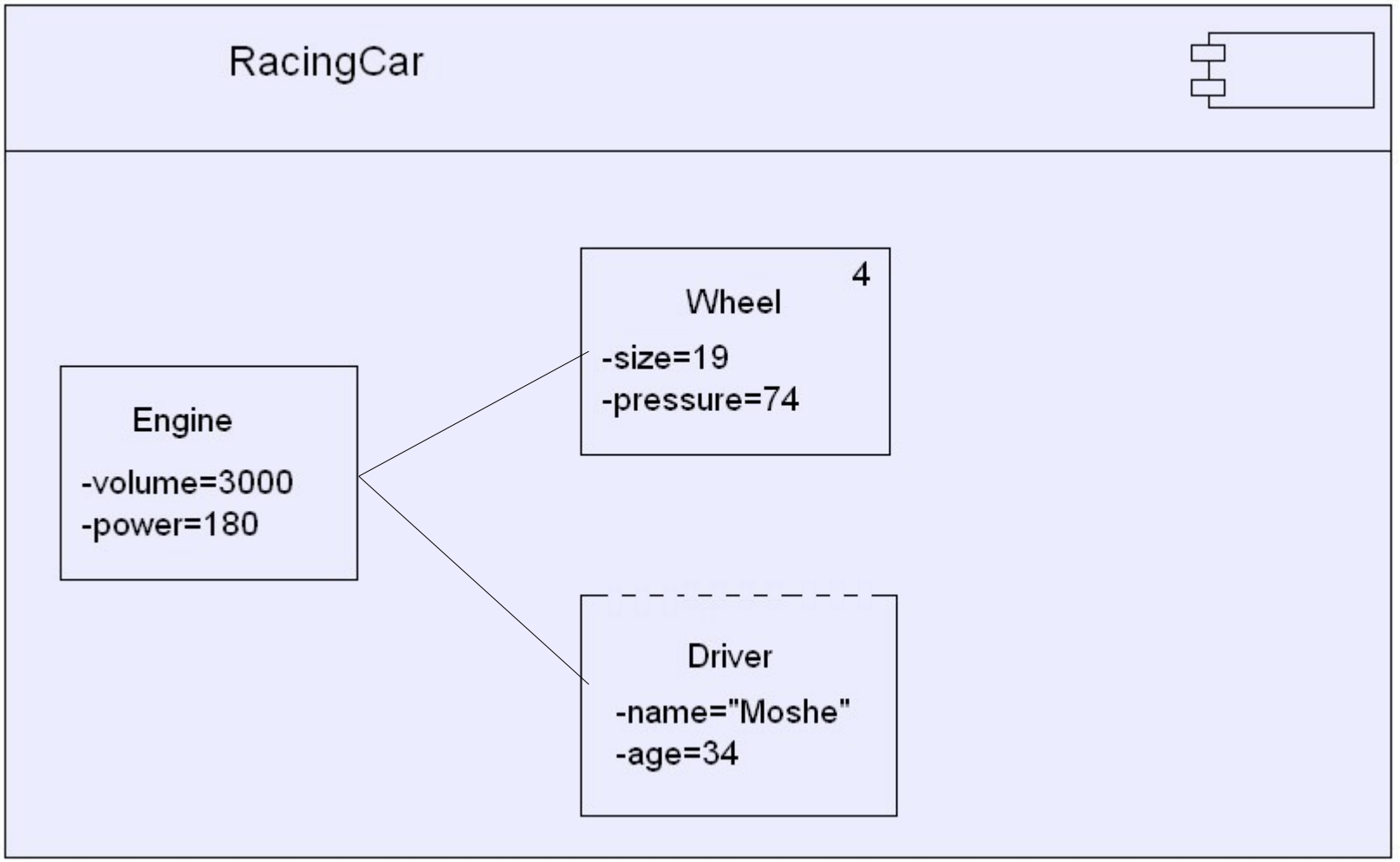
- When the port is instantiated we can mention the class type from which it was instantiated.



Structured Classes & Properties

- It is possible to specify the initial values for each object.

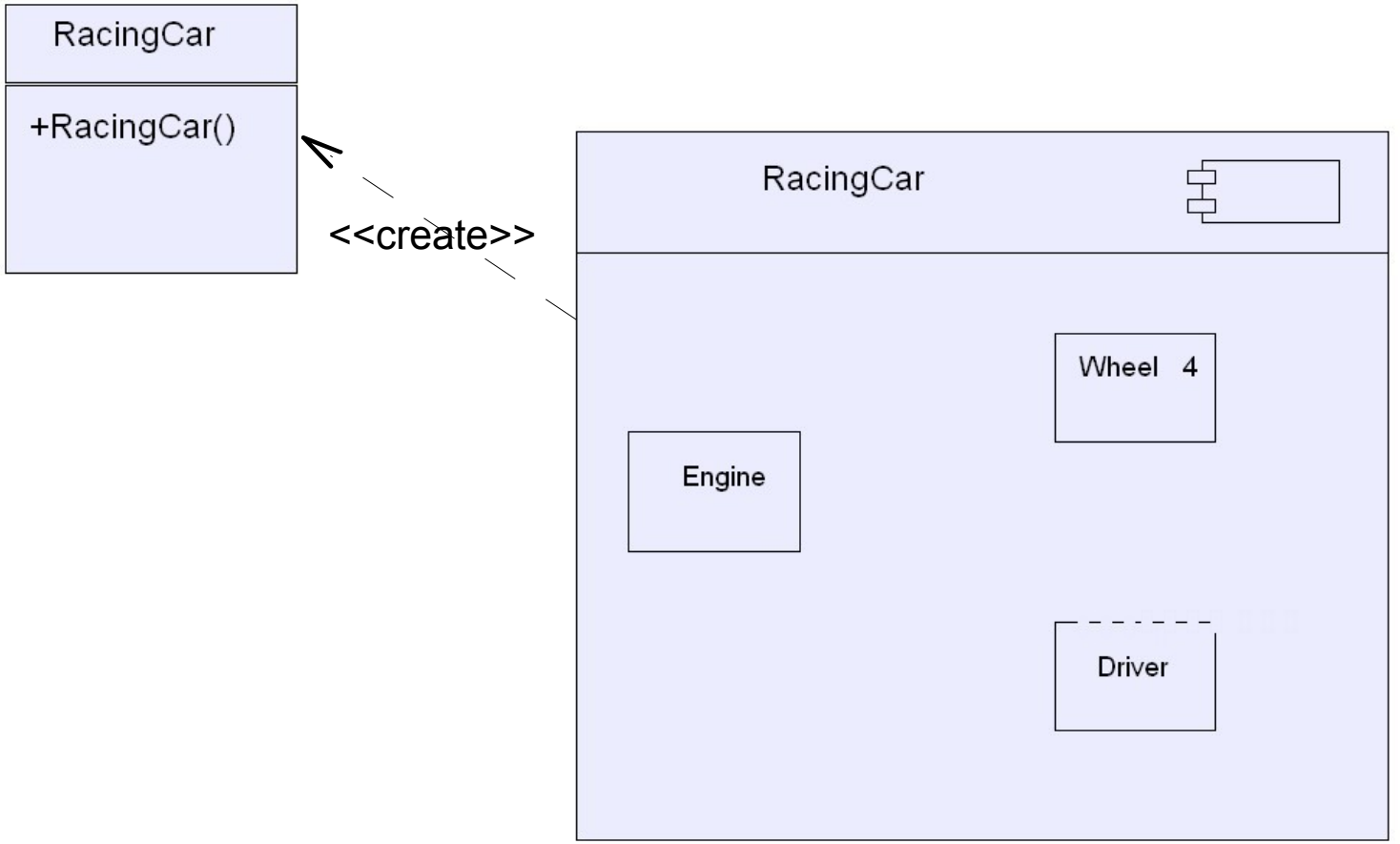
Structured Classes & Properties



Structured Classes & Properties

- It is possible to show that the object was created using a specific constructor of the classifier by using a dependency line labeled with the `<<create>>` keyword and use it to connect the object with that specific constructor.

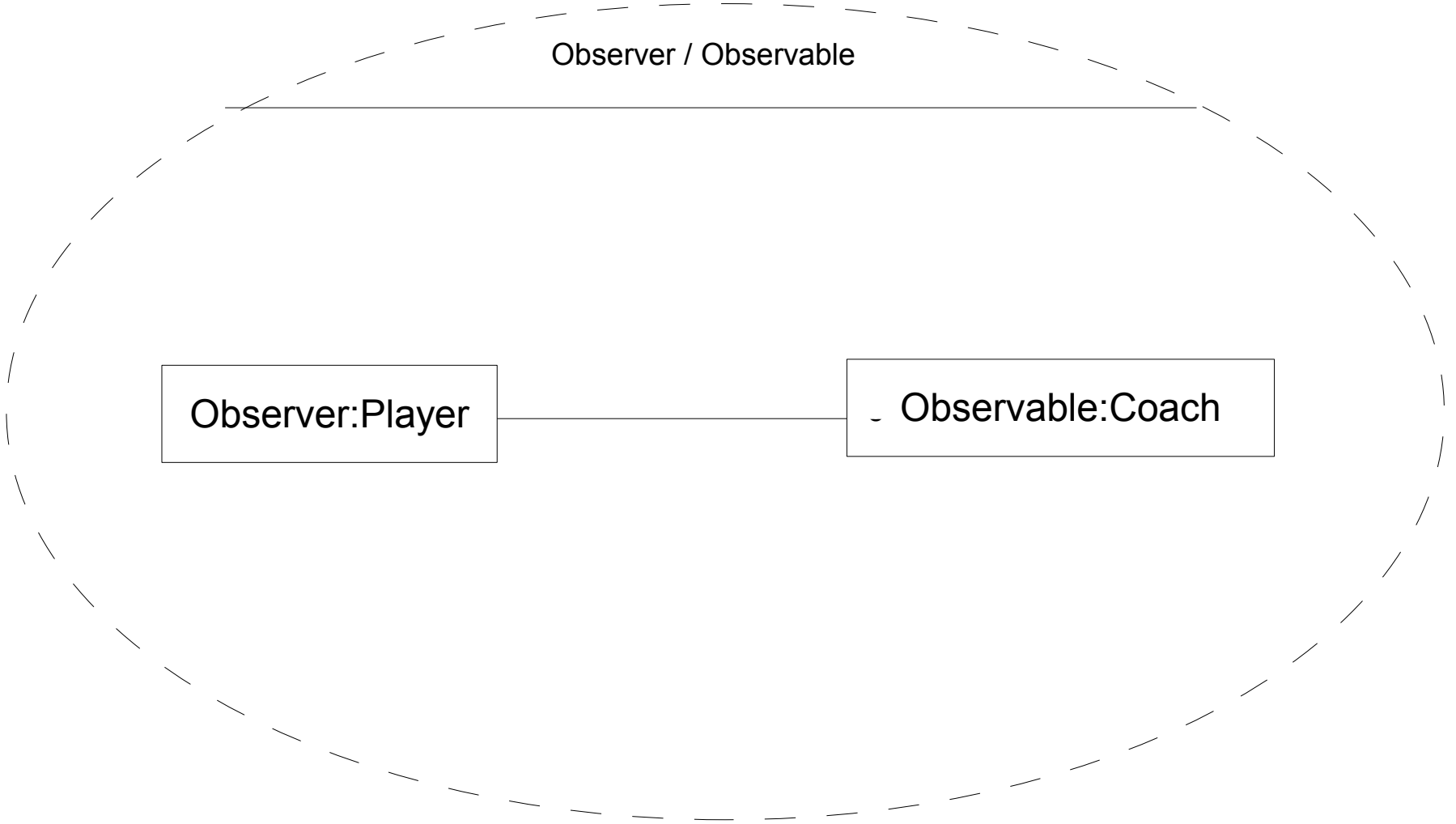
Structured Classes & Properties



Collaboration

- Organizing the elements in order to realize a behavior is called Collaboration. Such collection of instances is wired together using connectors to show the communication flow.
- Within the collaboration we name each involved instance with a name based on its role (not its class type).
- A collaboration is depicted using a dashed eclipse. The name we give that collaboration is written within that eclipse.

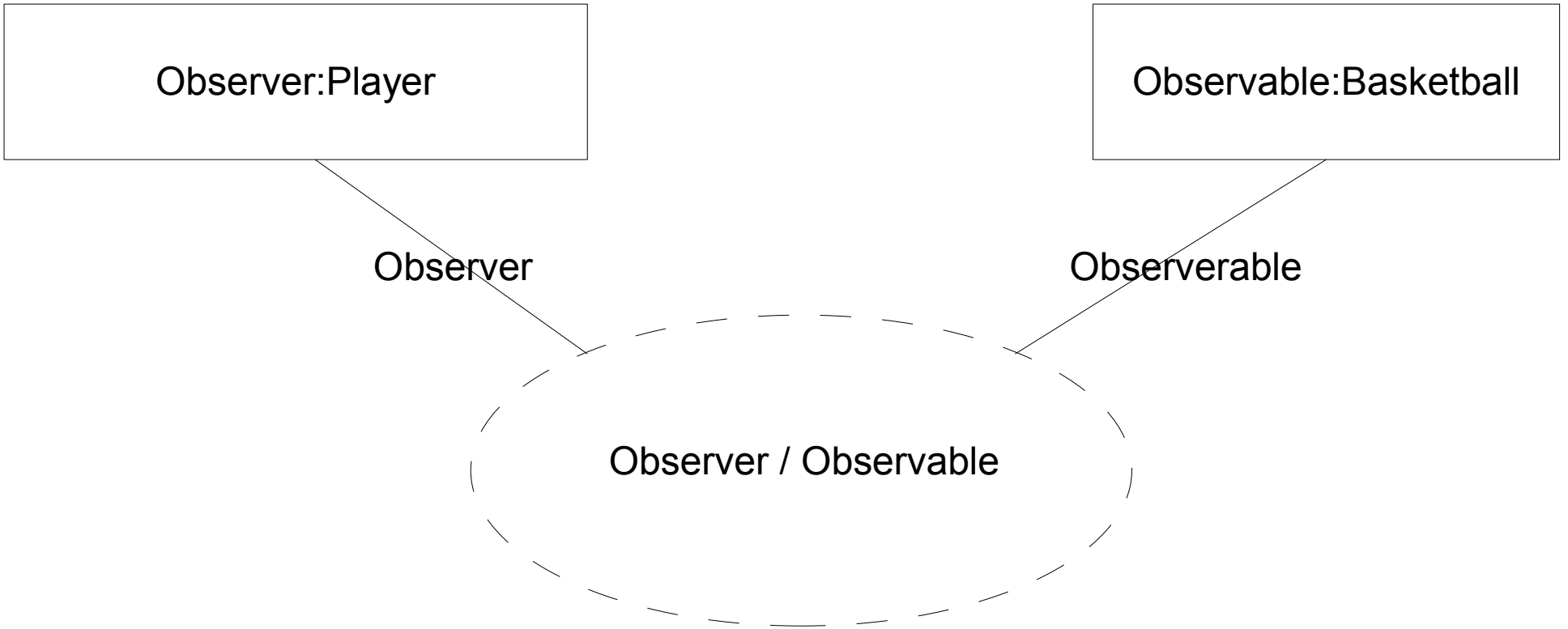
Collaboration



Collaboration

- Alternatively, we can draw a smaller collaboration eclipse and depict the two instances outside of the eclipse and tie them with the eclipse using communication links. When doing so, the role name will be written above each one of the communication links (instead of writing it within the instance rectangle).
- Choosing this alternative approach we can also have more space to include a list of all members (fields & methods) for each one of the objects.

Collaboration



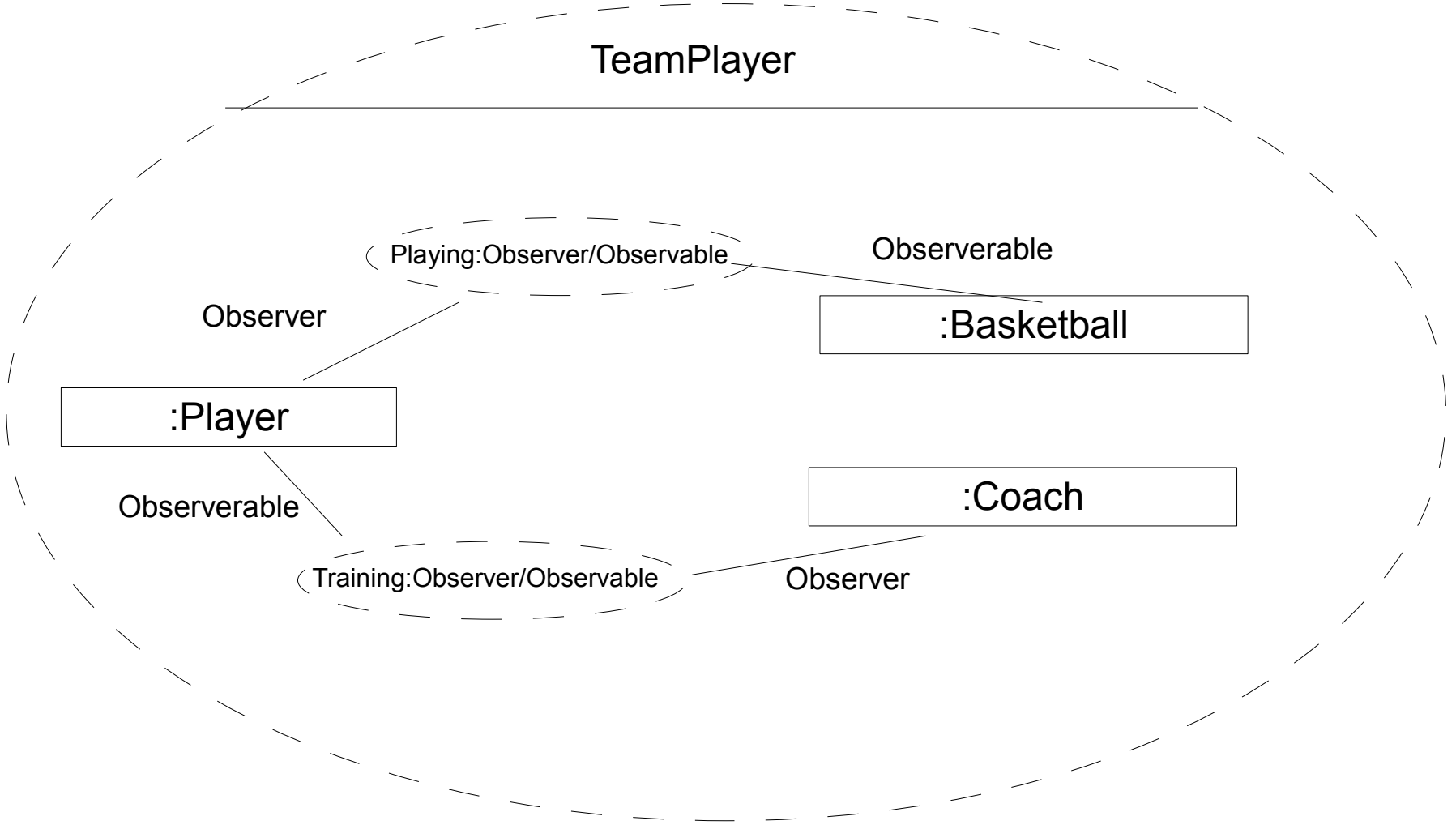
Collaboration Occurrence

- A collaboration occurrence is kind of an instance of collaboration. This way, if a given collaboration already exists and we find that same collaboration made up as a sub system of other objects we can consider it as a collaboration occurrence.
- When using collaboration occurrence we can assign role names to internal elements of the classifier. This is very helpful given that there may be multiple occurrences of a particular collaboration.

Collaboration Occurrence

- Showing more than one collaboration occurrence in the same diagram can be done by drawing the collaboration as a small dashed eclipse and drawing dashed lines from that eclipse to each element that takes part fulfilling a specific role in the collaboration.

Collaboration Occurrence



UML Composite Structure Diagrams

Introduction

- Similarly to UML Component Diagram, the UML Composite Structure Diagram focuses on showing the internal structure of a specific classifier.
- Many of the notations known from UML Component Diagram can be used within a UML Composite Diagram as well.
- While the purpose of the Component Diagram is showing the internal structure, the purpose of the Composite Structure Diagram is showing how a specific funcionality is implemented.

The component and the composite structure seem to be very similar. What is the difference between these two diagram types?

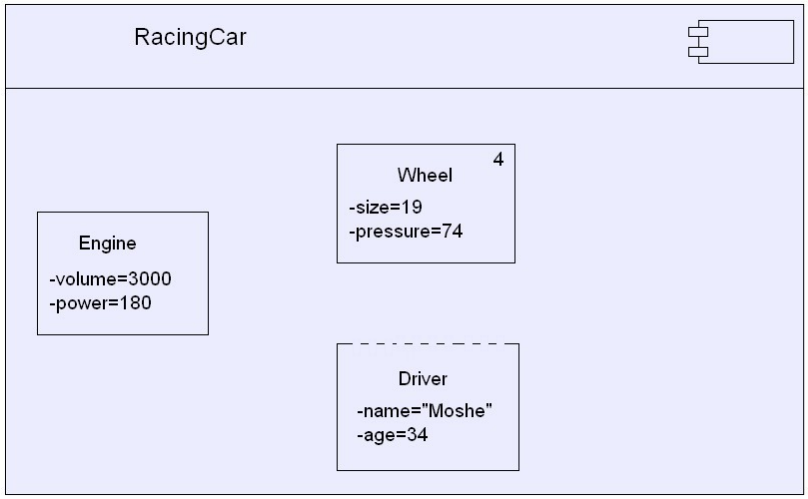
Component Diagram

"Set of constructs that can be used to define software systems of arbitrary size and complexity" (UML Specification). Using the Component Diagram fits those cases in which we analyze the system from a component-based development perspective.

Composite Structure Diagram

"A composite structure diagram depicts the internal structure of a classifier." (UML Specification). We will use the composite structure diagram when we focus on a specific composition of interconnected elements, representing run-time instances.

Introduction



Connectors

- A connector represents a communication link between two objects.
- A connector can be an object that represents an association or a value we hold in a variable. A value held in a variable is another way to represent a connection between two objects.

Connectors

- The notation for connector is a solid line. For each connector we can provide a name and a type in the following format:

  `name:classname`

  `name`

  The name of the connector

  `classname`

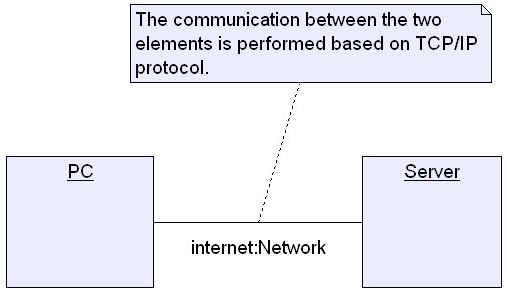
  The name of the class that was instantiated to represent an association. Relevant when the connector represents a link between two objects... a link that is represented by object of a specific class... an object that represents the link as an association.

Connectors

The connection between the two objects exists between their two classes as association. Therefore, the connector is an object. In this case, it is an object of type Network and the object name is internet.

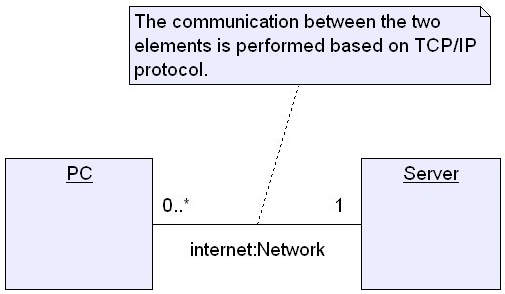
More Information

- Using a simple note it is possible to provide more information about the connector, which means: more information about the communication between the two elements (performed via the connector).



Connector's Multiplicity

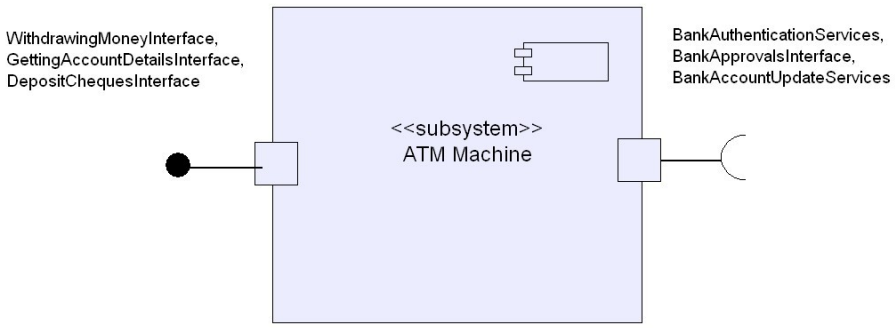
- Using the formal multiplicity syntax it is possible to specify the multiplicity of each connector end.



The Ports

- A port is a way to offer functionality from a composite structure without exposing details about that composite structure internal implementation.
- The port notation is a small square drawn above the edge line of the boarder.
- The port name and its multiplicity are written near to it.

The Ports


WithdrawingMoneyInterface,
GettingAccountDetailsInterface,
DepositChequesInterface
<<subsystem>>
ATM Machine
BankAuthenticationServices,
BankApprovalsInterface,
BankAccountUpdateServices

Required and Provided Interfaces

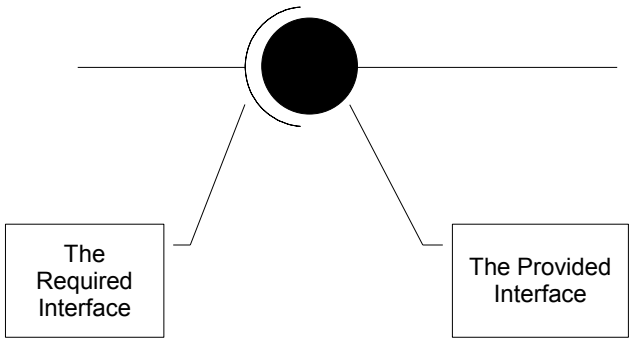
- The ports are associated with required/provided interfaces from/to the environment.

  Example:

  An ATM machine might provide an interface to withdraw money... at the same time it might require an interface to the bank system through which the withdraw could be approved and updated on the bank system data base.

Required and Provided Interfaces

- The required/provided interfaces are usually depicted using the following notation:



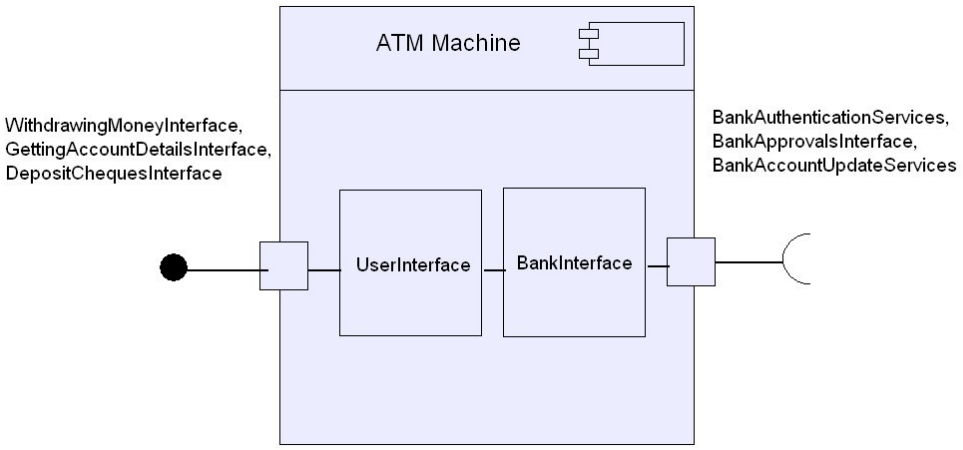
Required and Provided Interfaces

- If there are multiple required/provided interfaces we can write them comma separated near the relevant socket/ball.

Ports Implementation

- Each port is wired to its internal implementation using connectors.
- The internal implementation can be some code owned by the classifier (this is usually the case when dealing with small classes). In such case the port will be named “Behavioral Port”.
- If the implementation of the port is done by another inner element then we will link the port to its internal implementation.

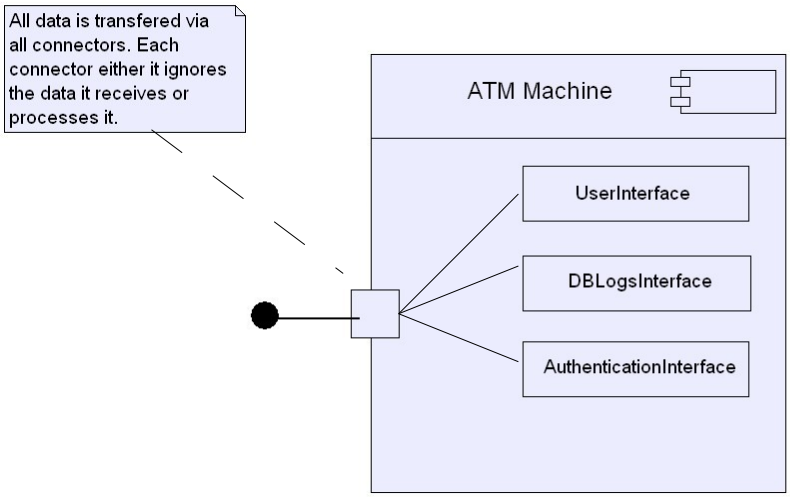
Ports Implementation



One Port & Multiple Connectors

- UML specification allows having one port connected with multiple connectors.
- When this is the case it is important to add a note with explanation about how does the data that arrives from the environment continues its flow to the connectors... is it duplicated for each connector.. is there a mechanism that selects which data to forward through which connector.. etc.

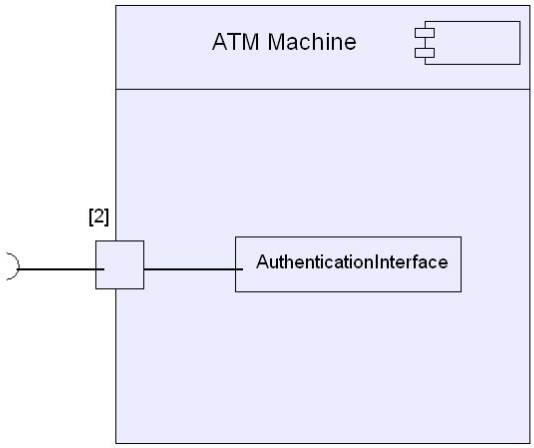
One Port & Multiple Connectors



Multiple Ports

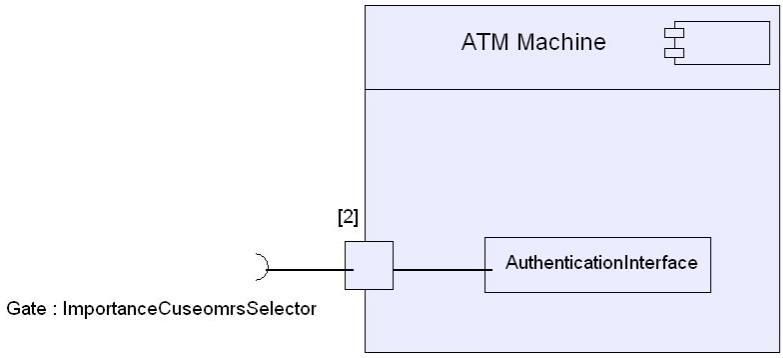
- Having more than one port for providing/receiving the same interface/s is feasible. In such case we will specify the number of ports near the port notation.
- Having multiple ports can happen when we have two instances of two different classes that both of them implement the interface that was declared for that port (e.g. AuthenticationInterface was implemented twice... first in a class that authenticates regular customers... and second is a class that authenticates premium customers.

Multiple Ports



Ports' Types

- When the port is instantiated we can mention the class type from which it was instantiated.

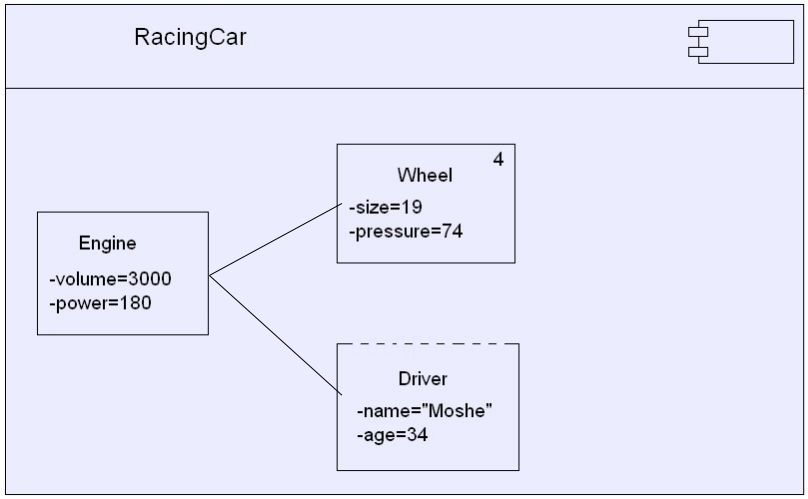


Structured Classes & Properties

- It is possible to specify the initial values for each object.

The multiplicity can be reflected by drawing the number in the rectangle upper right corner or after the property name (in brackets).

Structured Classes & Properties

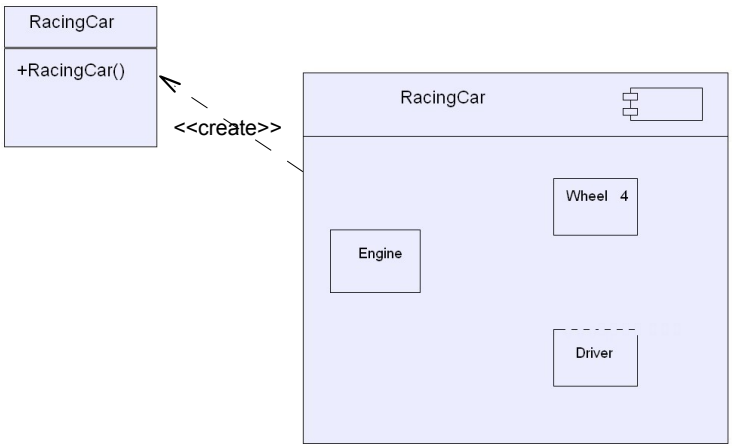


The dashed/solid rectangle option replaces the special notation used in class diagram to represent composition, aggregation & association

Structured Classes & Properties

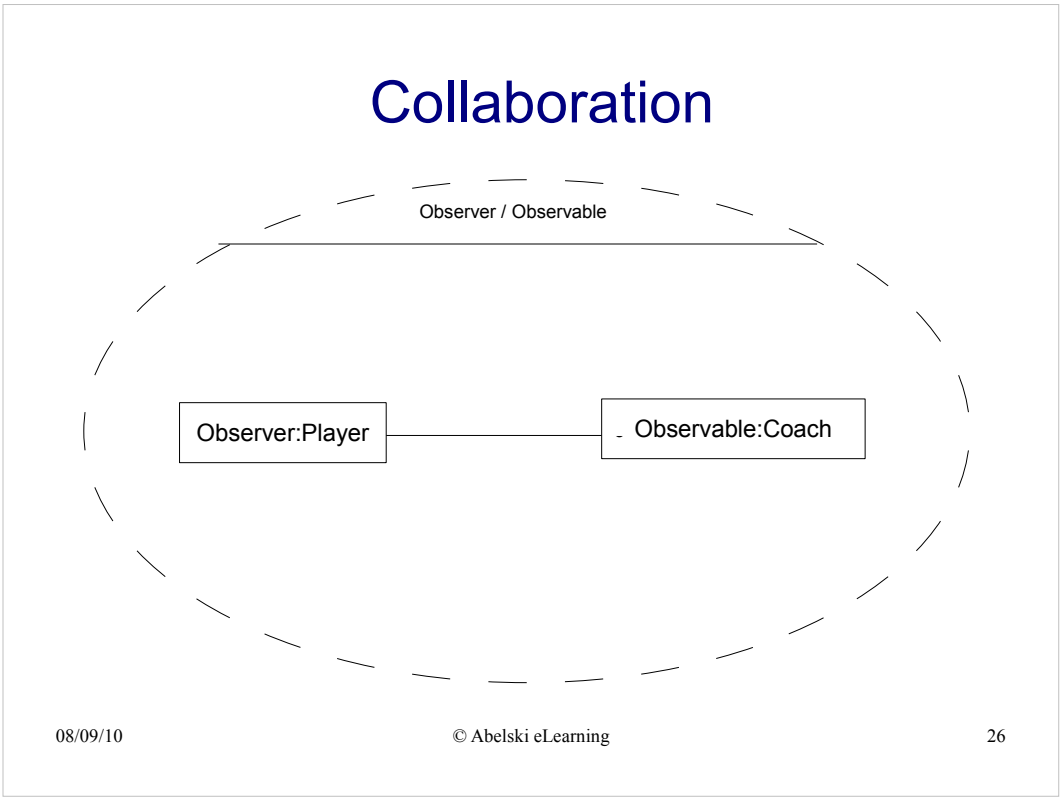
- It is possible to show that the object was created using a specific constructor of the classifier by using a dependency line labeled with the `<<create>>` keyword and use it to connect the object with that specific constructor.

Structured Classes & Properties



Collaboration

- Organizing the elements in order to realize a behavior is called Collaboration. Such collection of instances is wired together using connectors to show the communication flow.
- Within the collaboration we name each involved instance with a name based on its role (not its class type).
- A collaboration is depicted using a dashed eclipse. The name we give that collaboration is written within that eclipse.



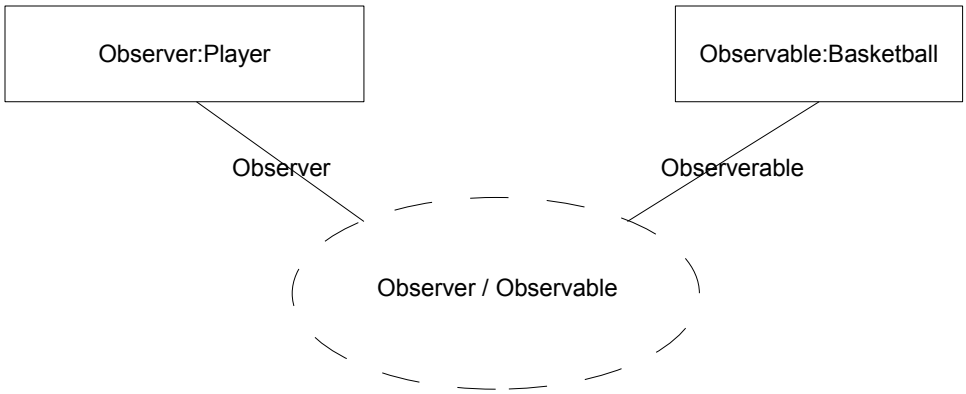
UML 2.0 specification allows creating collaboration diagrams with any of the available behavioral diagrams.

Collaboration

- Alternatively, we can draw a smaller collaboration eclipse and depict the two instances outside of the eclipse and tie them with the eclipse using communication links. When doing so, the role name will be written above each one of the communication links (instead of writing it within the instance rectangle).
- Choosing this alternative approach we can also have more space to include a list of all members (fields & methods) for each one of the objects.

UML 2.0 specification allows creating collaboration diagrams with any of the available behavioral diagrams.

Collaboration



Collaboration Occurrence

- A collaboration occurrence is kind of an instance of collaboration. This way, if a given collaboration already exists and we find that same collaboration made up as a sub system of other objects we can consider it as a collaboration occurrence.
- When using collaboration occurrence we can assign role names to internal elements of the classifier. This is very helpful given that there may be multiple occurrences of a particular collaboration.

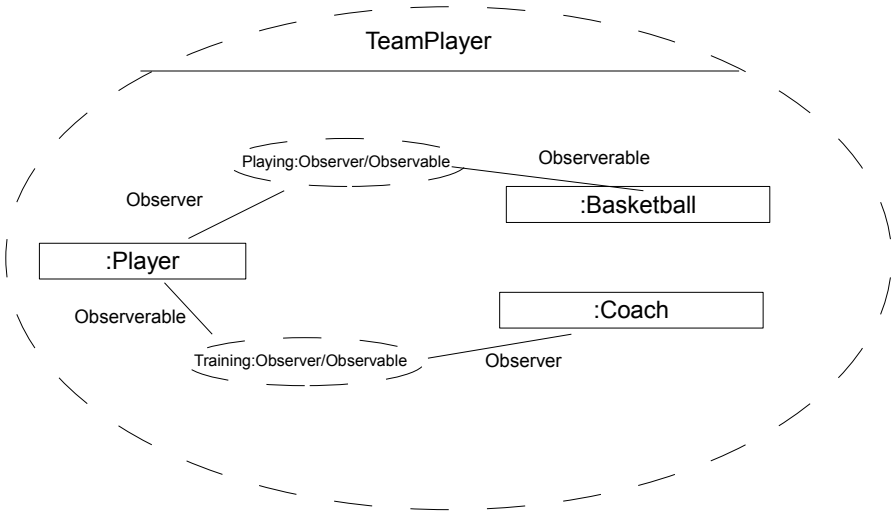
UML 2.0 specification allows creating collaboration diagrams with any of the available behavioral diagrams.

Collaboration Occurrence

- Showing more than one collaboration occurrence in the same diagram can be done by drawing the collaboration as a small dashed eclipse and drawing dashed lines from that eclipse to each element that takes part fulfilling a specific role in the collaboration.

UML 2.0 specification allows creating collaboration diagrams with any of the available behavioral diagrams.

Collaboration Occurrence



UML 2.0 specification allows creating collaboration diagrams with any of the available behavioral diagrams.