BRAC UNIVERSITY
Inspiring Excellence

Component Diagram

Component Diagrams

- UML component diagrams describe software components and their dependencies to each others
- Models the physical implementation of the software (file resources)
- Models the high-level software components, and their interfaces
- We use component diagrams to model the static implementation view of a software system.
- It is a type of structural UML diagram

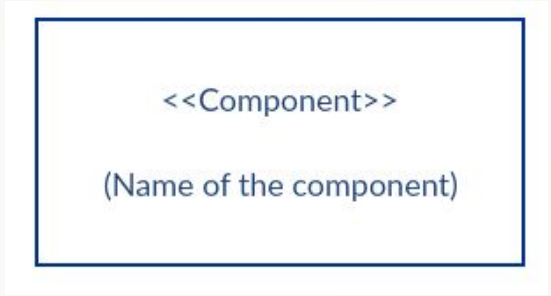
What is a Component

- Modular unit with well-defined interfaces that is replaceable within its environment
- Autonomous unit within a system
- Has one or more provided and required interfaces
- Its internals are encapsulated, hidden and inaccessible
- Its dependencies are designed such that it can be treated as independently as possible
- The components can be a software component such as a database or user interface; or a hardware component such as a circuit, microchip or device; or a business unit such as supplier, payroll or shipping.

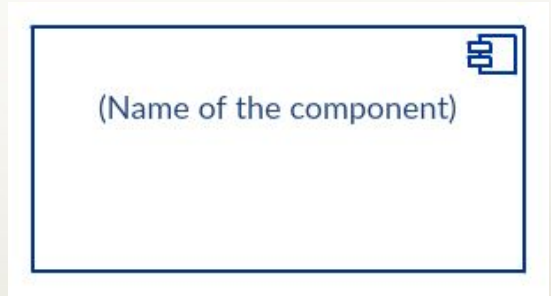
Component Notation

1) Rectangle with the component stereotype (the text <<component>>). The component stereotype is usually used above the component name to avoid confusing the shape with a class icon.Components can be labelled with a number of standard stereotypes

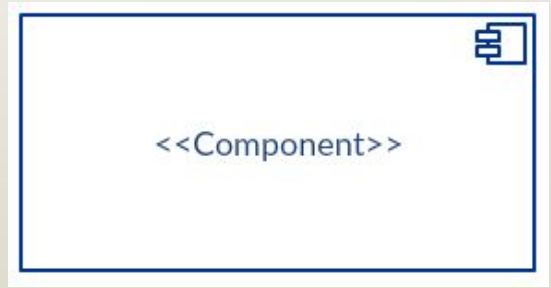
ex: <<entity>>, <<subsystem>>,<<table>>,<<library>>, etc



2) Rectangle with the component icon in the top right corner and the name of the component.



3) Rectangle with the component icon and the component stereotype.



Component Elements

1. Interfaces

An interface represents a declaration of a set of operations and obligations

2. Usage dependencies

A usage dependency is relationship which one element requires another element for its full implementation

3. Ports

Port represents an interaction point between a component and its environment

4. Connectors

Connect two components through interface.

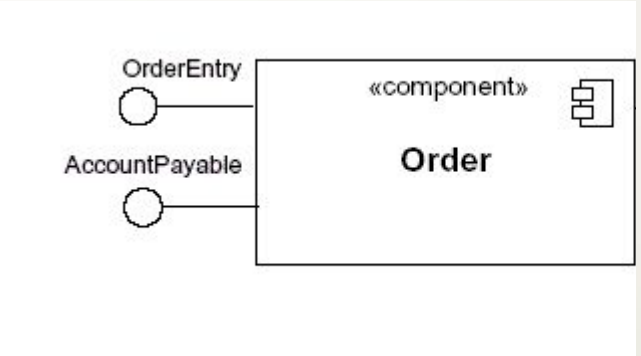
INTERFACE

- A component defines its behaviour in terms of provided and required interfaces
- An interface
  - Is the definition of a collection of one or more operations
  - Provides only the operations but not the implementation
  - Implementation is normally provided by a class/ component
  - In complex systems, the physical implementation is provided by a group of classes rather than a single class
- 2 types
  - Provided Interface
  - Required interface

INTERFACE

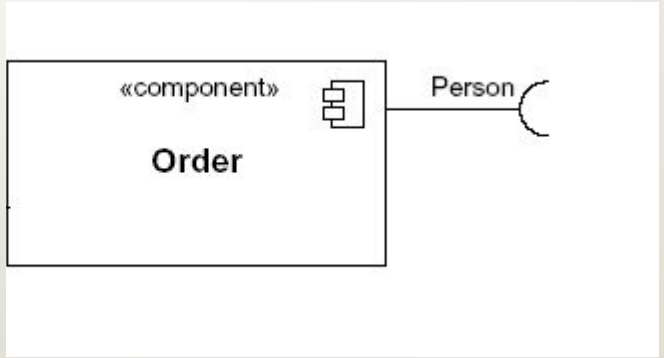
Provided Interface

- A component's provided interface describes the services provided by the component.
- Is modeled using a ball, labelled with the name, attached by a solid line to the component



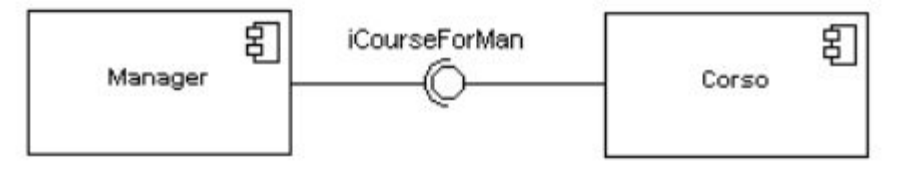
Required Interface

- A required interface declares the services a component will need.
- Is modeled using a socket, labelled with the name, attached by a solid line to the component

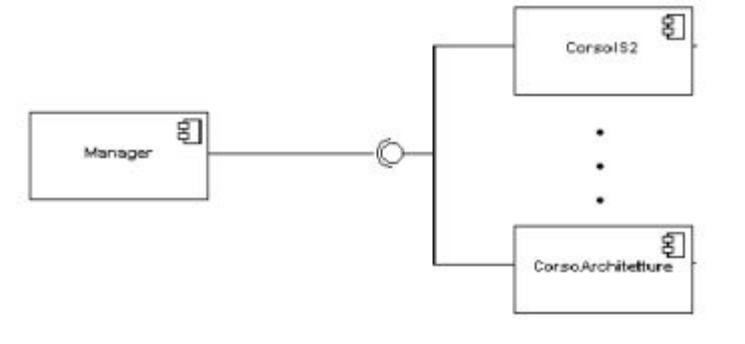


INTERFACE

Where two components/classes provide and require the same interface, these two notations may be combined

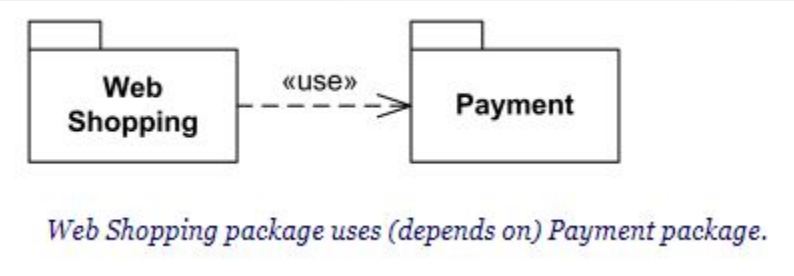


In a system context where there are multiple components that require or provide a particular interface, a notation abstraction can be used that combines by joining the interfaces



Usage Dependency

- A usage dependency is relationship which one element requires another element for its full implementation
- Is a dependency in which the client requires the presence of the supplier
- Is shown as dashed arrow with a <> keyword
- The arrowhead point from the dependent component to the one of which it is dependent

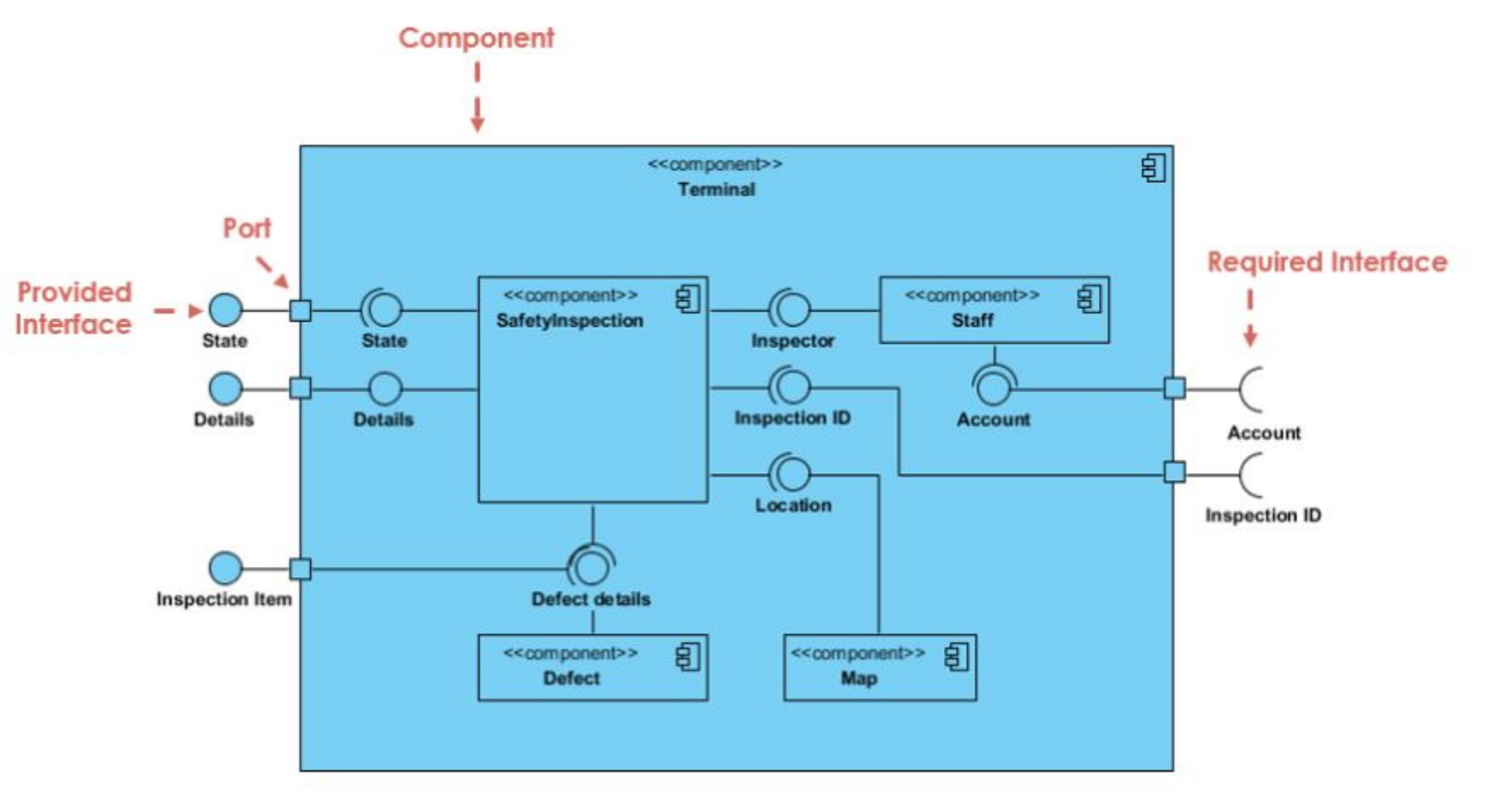


Web Shopping package uses (depends on) Payment package.

Port

Specifies a distinct interaction point

- Between that component and its environment
- Between that component and its internal parts
- port may have visibility. When a port is drawn over the boundary of a component, then it means that the port is public. It also means that all the interfaces used are made as public.
- When a port is drawn inside the component, then it is either protected or private.


Component
<<component>>
Terminal
Port
Provided Interface
State
State
<<component>>
SafetyInspection
Inspector
<<component>>
Staff
Required Interface
Details
Details
Inspection ID
Account
Account
Location
Inspection ID
Inspection Item
Defect details
<<component>>
Defect
<<component>>
Map

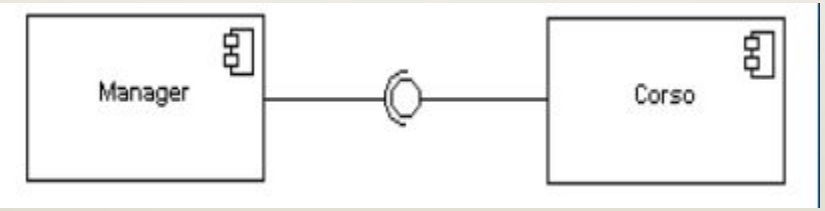
Connector

Two kinds of connectors:

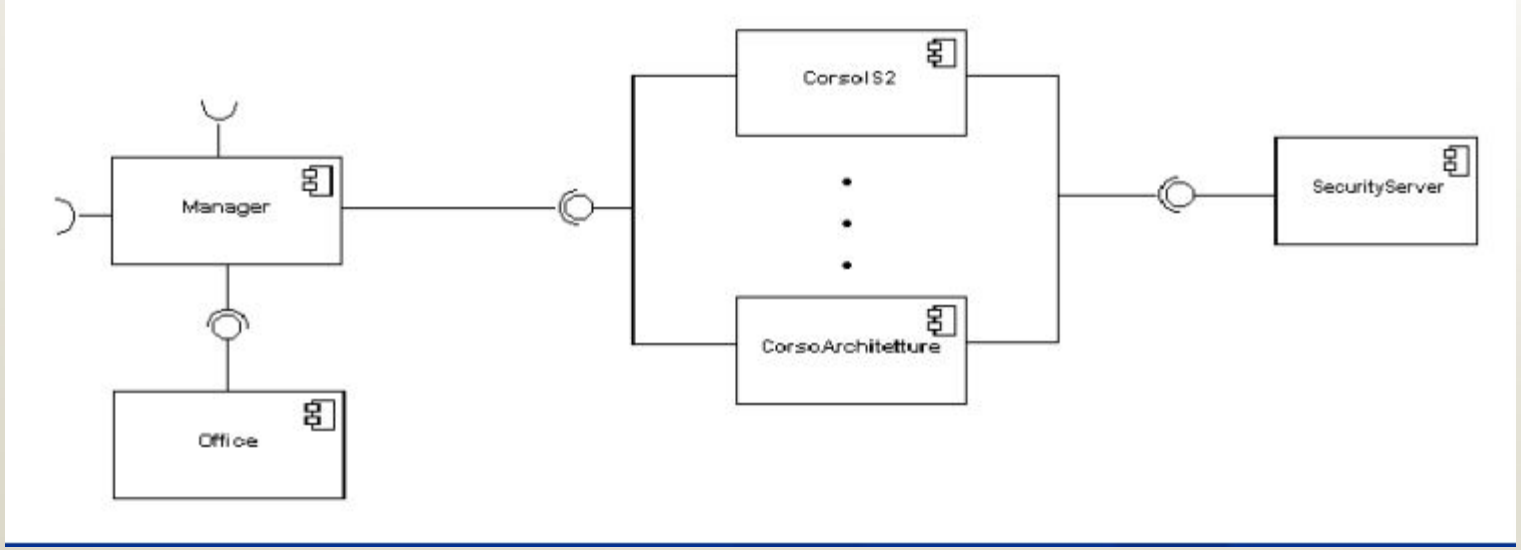
- Assembly
- Delegation

ASSEMBLY CONNECTOR

- A connector between 2 components defines that one component provides the services that another component requires
- An assembly connector is notated by a "ball-and-socket" connection

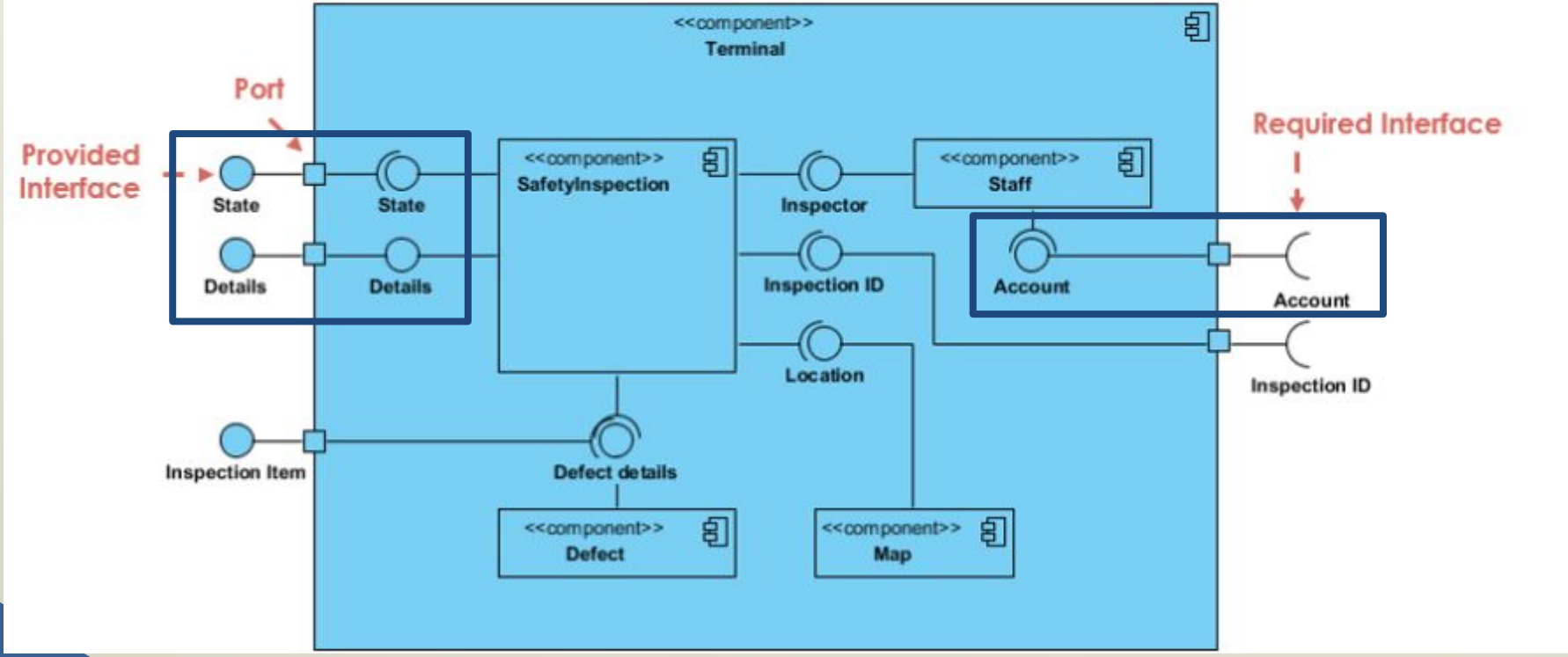


Multiple connections directed from a single required interface to provided interfaces indicates that the instance that will handle the signal will be determined at execution time



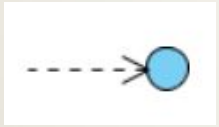
DELEGATION CONNECTOR

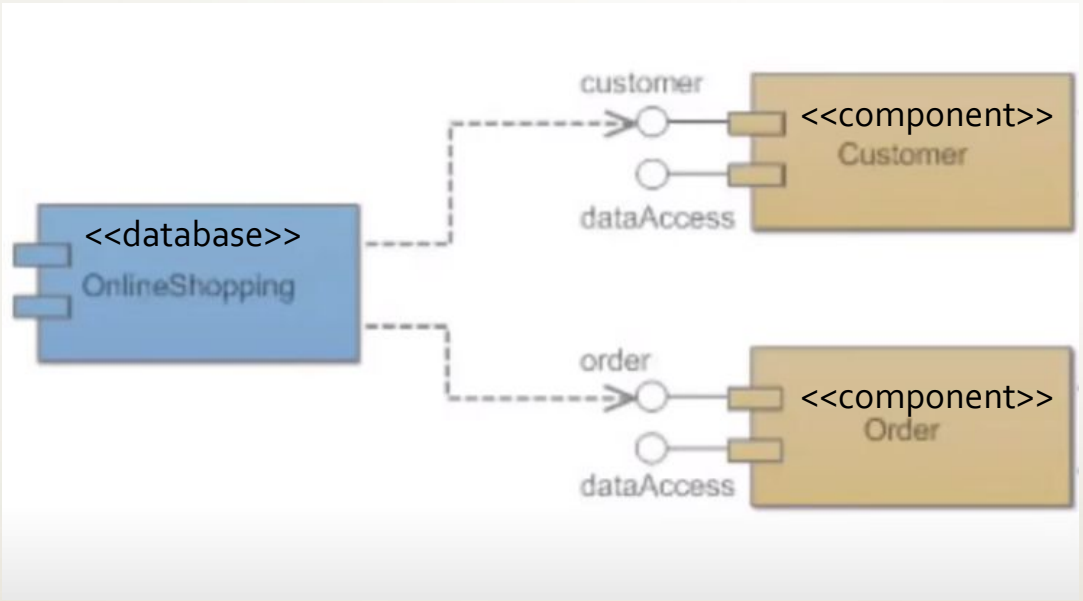
- Links the external contract of a component to the internal component
- Represents the forwarding of signals
- It must only be defined between used interfaces or ports of the same kind



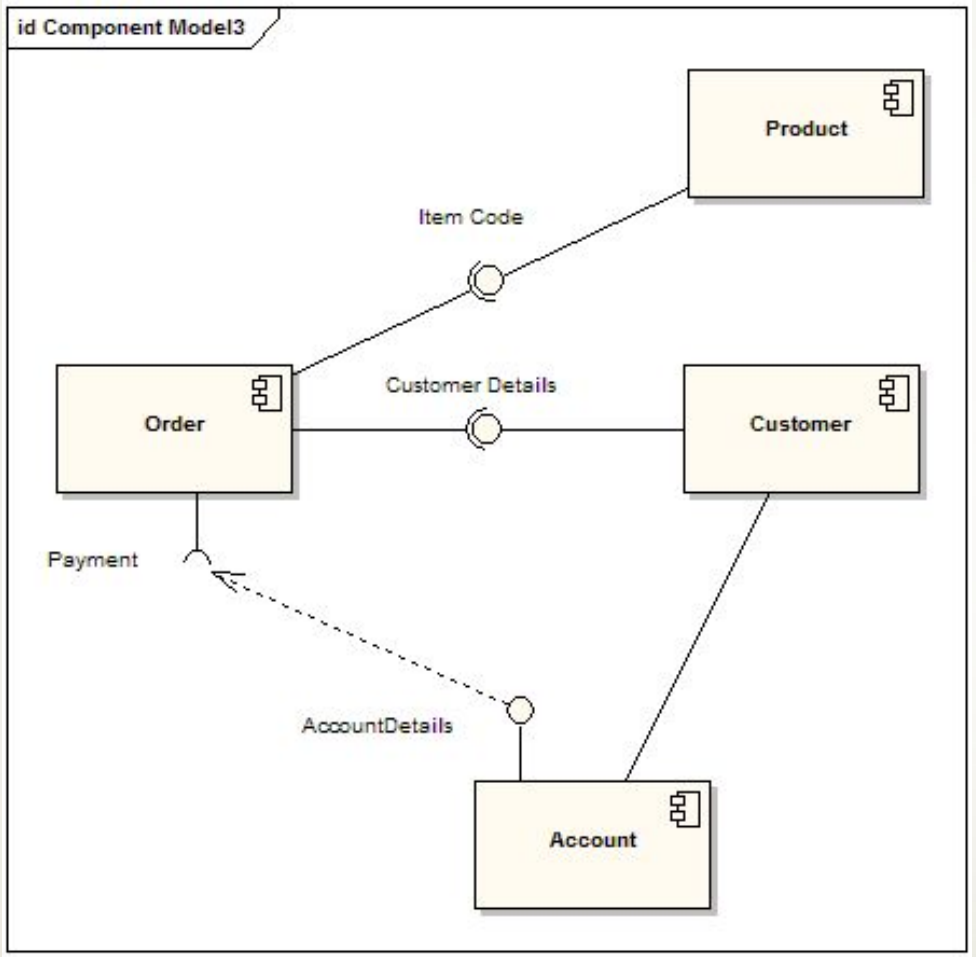
Dependency Relation

One component is dependent on data from another component





Here order and customer component depends on data from online shopping database

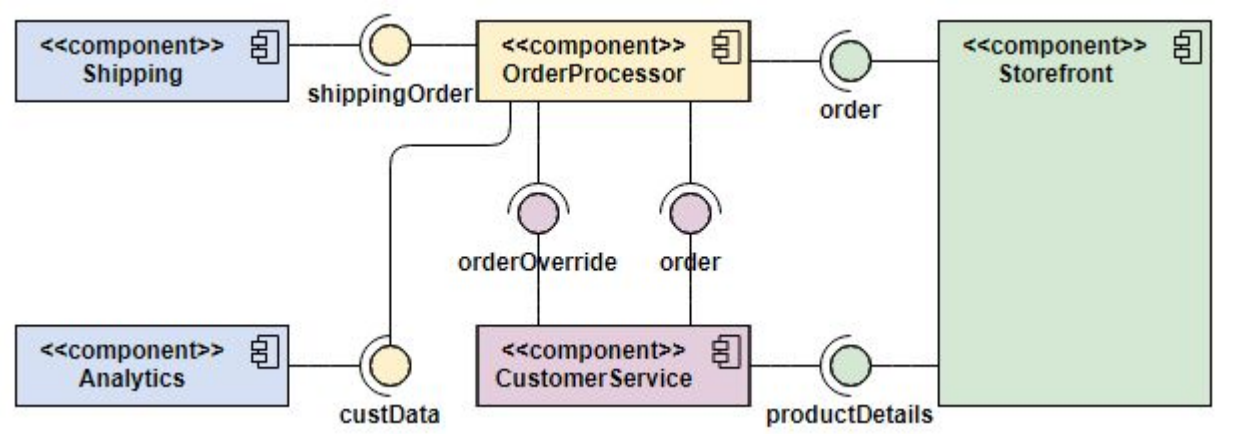


Here the payment interface depends on data from Account Component for it's full implementation. The dependency maps a customer's associated account details to the required interface "Payment"

Where to use

- To model the component of the system
- To model source code
- To model executable releases
- To model physical databases
- To model adaptable systems

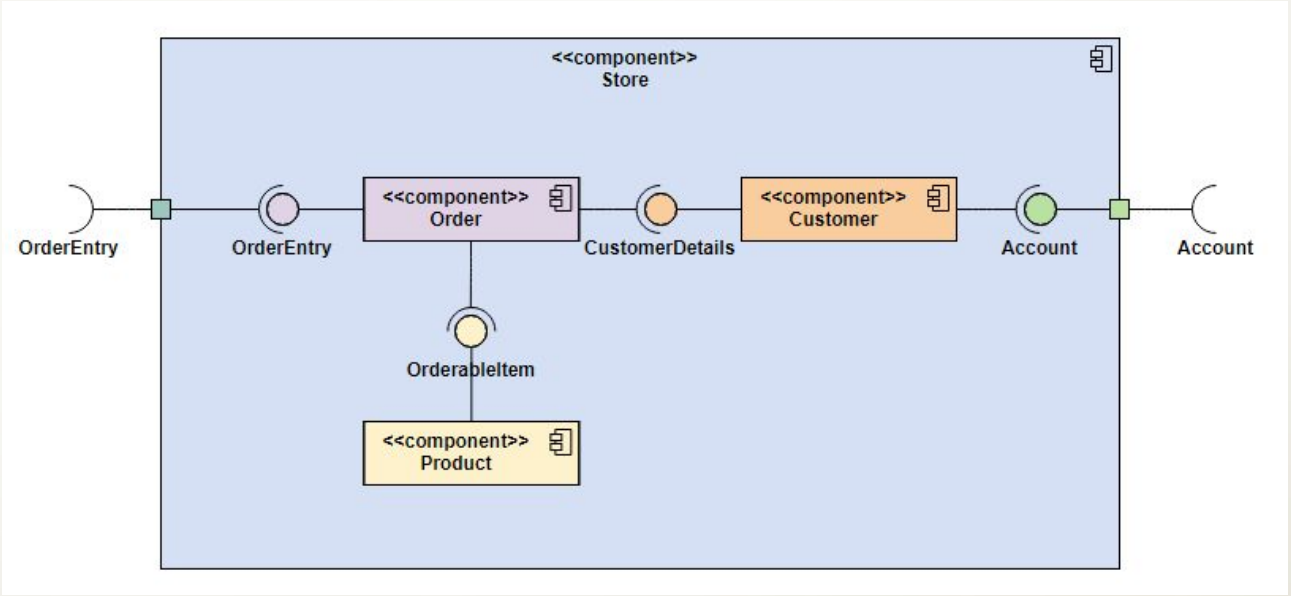
Order Processing System Component diagram example



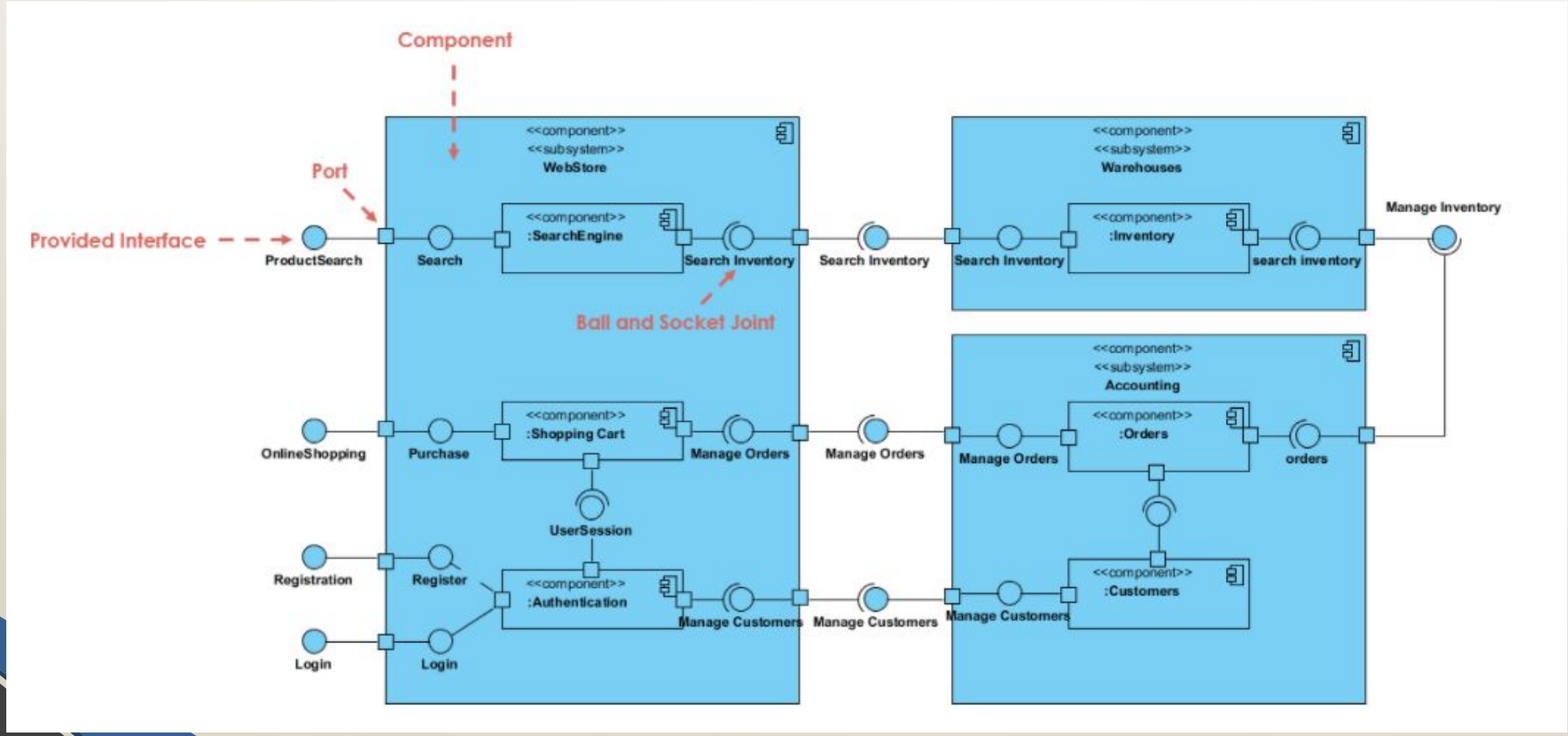
Here the Storefront component provides interfaces for Order and ProductDetails. CustomerService implements this interface as this component needs access to ProductDetails of the products available in storefront. Again CustomerService is able to order and override orders. All these operations are handled by the Orderprocessor component. Then this component provide Customer data to analytics component and shipping data to the shipping component

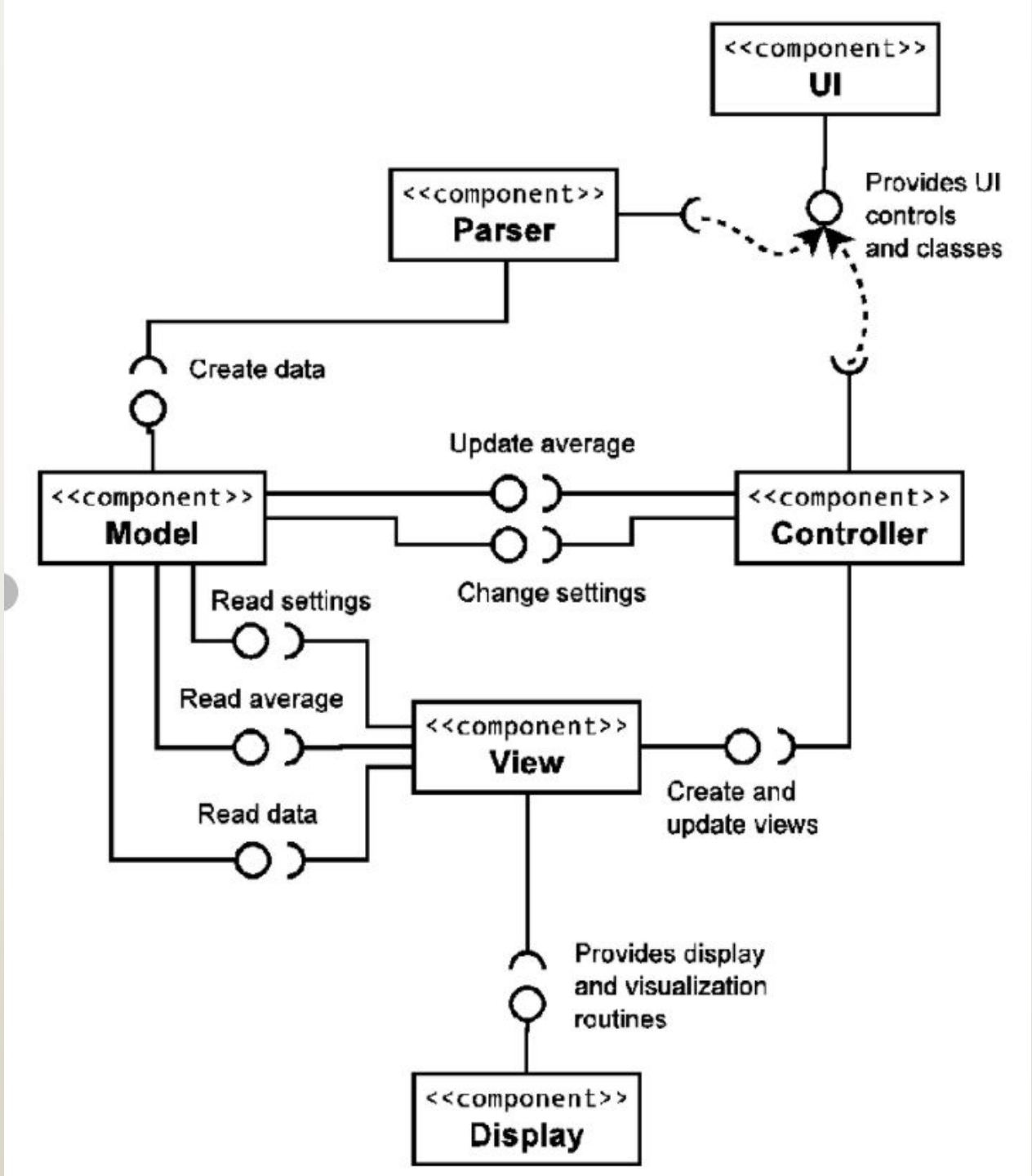
Nested Component Structure

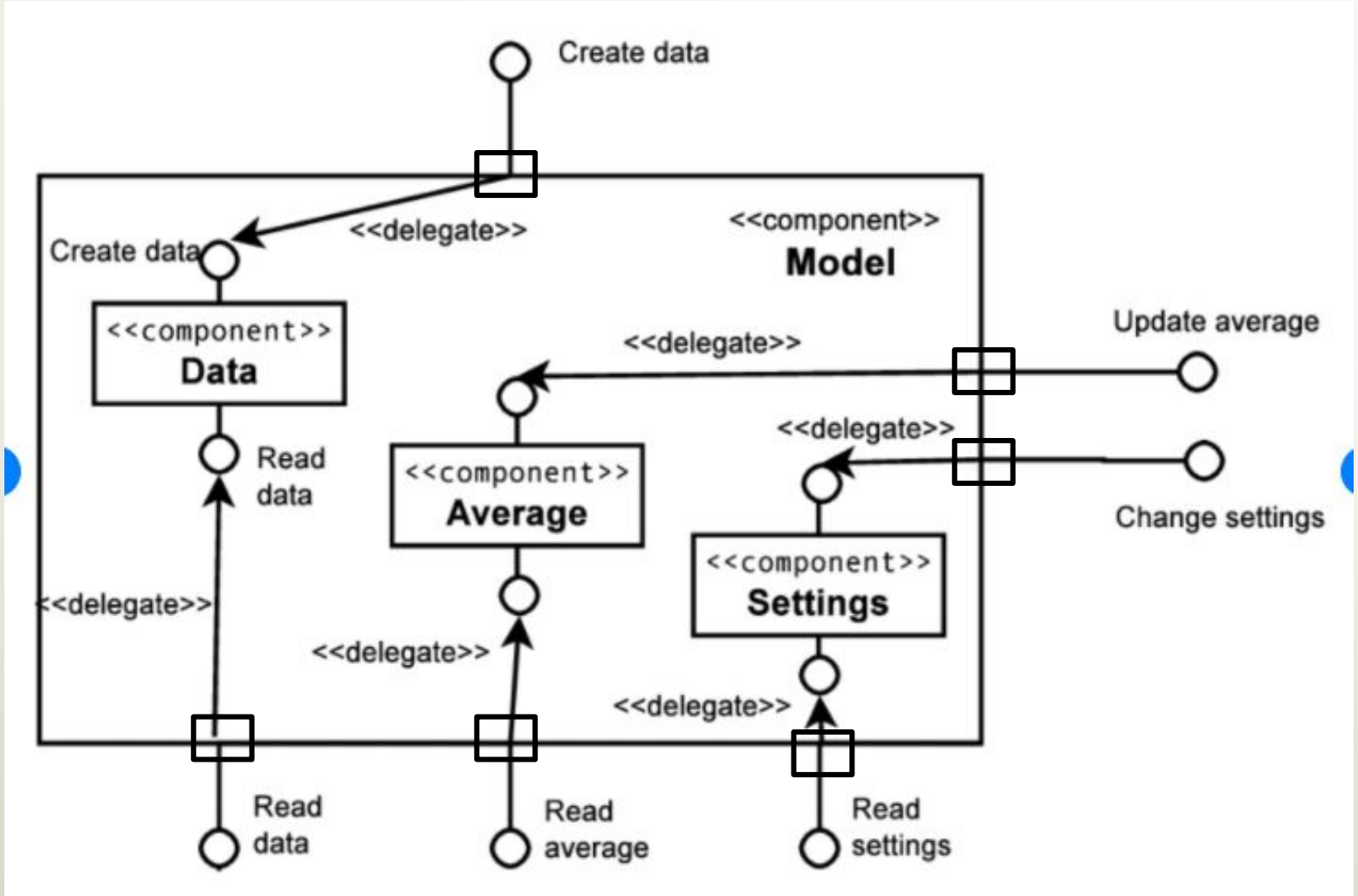
Here the Store component has some components inside it. In such cases delegation connector and port is used to pass signal from external component to the internal ones. Here we see that Customer component requires the account and and Order component provides order entry related functionalities from an external component.



Some more examples




<<component>>
UI
<<component>>
Parser
Provides UI
controls
and classes
Create data
Update average
<<component>>
Model
<<component>>
Controller
Change settings
Read settings
Read average
<<component>>
View
Create and
update views
Read data
Provides display
and visualization
routines
<<component>>
Display


Create data
<<delegate>>
Create data
<<component>>
Model
<<component>>
Data
Update average
<<delegate>>
<<delegate>>
Read
data
<<component>>
Average
Change settings
<<component>>
Settings
<<delegate>>
<<delegate>>
<<delegate>>
Read
data
Read
average
Read
settings

Life Care Hospital is a renowned hospital that uses a simple hospital management system named Life Care hospital management. The XYZ hospital management component is dependent on the access of data from four components: Hospital, Patient, Doctor, and Nurse. These four components provide an interface named readData. There is another component named Appointment which provides two interfaces setAppointment and getAppointmentDetails. The Hospital, Patient and Doctor component implements the getAppointmentDetails interface while the Receptionist component implements the setAppointment interface. There are three more components named Synchronization, Database and Security. The dbConnector interface is provided by and required by Database and Synchronization components respectively. Both Security and Synchronization Components provide three crucial interfaces; accessControl, Encryption, and Decryption which are required by all of the components except XYZ hospital management and the Database component.

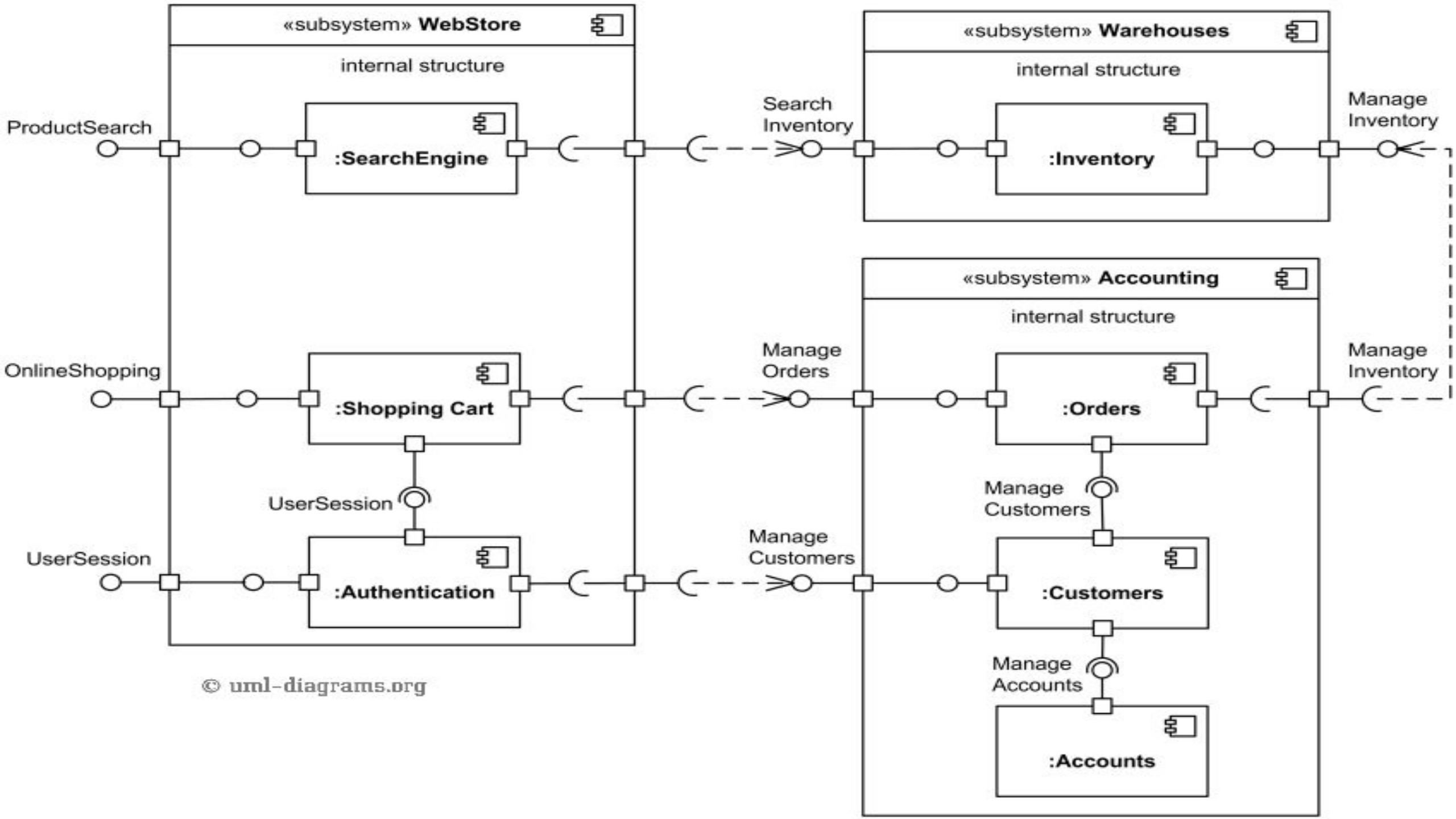
Now **Design** a **component diagram** based on the above scenario.

WebStore subsystem contains three components related to online shopping - Search Engine, Shopping Cart, and Authentication. Search Engine component allows to search or browse items by exposing provided interface Product Search and uses required interface Search Inventory provided by Inventory component. Shopping Cart component uses Manage Orders interface provided by Orders component during checkout. Authentication component allows customers to create account, login, or logout and binds customer to some account.

Accounting subsystem provides two interfaces - Manage Orders and Manage Customers. Delegation connectors link these external contracts of the subsystem to the realization of the contracts by Orders and Customers components.

Warehouses subsystem provides two interfaces Search Inventory and Manage Inventory used by other subsystems and wired through dependencies.

BRAC UNIVERSITY
Inspiring Excellence


«subsystem» WebStore
internal structure
:SearchEngine
:Shopping Cart
:Authentication
ProductSearch
OnlineShopping
UserSession
UserSession
Search Inventory
Manage Orders
Manage Customers
«subsystem» Warehouses
internal structure
:Inventory
Manage Inventory
«subsystem» Accounting
internal structure
:Orders
:Customers
:Accounts
Manage Inventory
Manage Customers
Manage Accounts
© uml-diagrams.org

An example of UML component diagram with some simplified view of provided and implemented components utilizing SafeNet Sentinel HASP Software Licensing Security Solution and Licensing API.

On the top of the diagram we have some software implemented using Sentinel HASP - License Status .Net application and License Services Java component. License Status application is intended to show license status and is manifested (implemented) by license_status.exe artifact. License Services Java component implements License Service interface and could be used by other Java applications or services.

License Status application uses License Services Net component through the License Service interface implemented by this component. The License Services Net component uses HASP .Net API provided by HASP .Net Runtime component which is part of Sentinel HASP product.

License Services Java component uses HASP Java Native Interface Proxy to communicate with HASP Java Native Interface component, both components provided by Sentinel. When product is used in Microsoft Windows, the HASP Java Native Interface could be manifested by either HASPJava.dll (32 bit OS), HASPJava_x64.dll, or HASPJava_ia64.dll (64 bit OS).

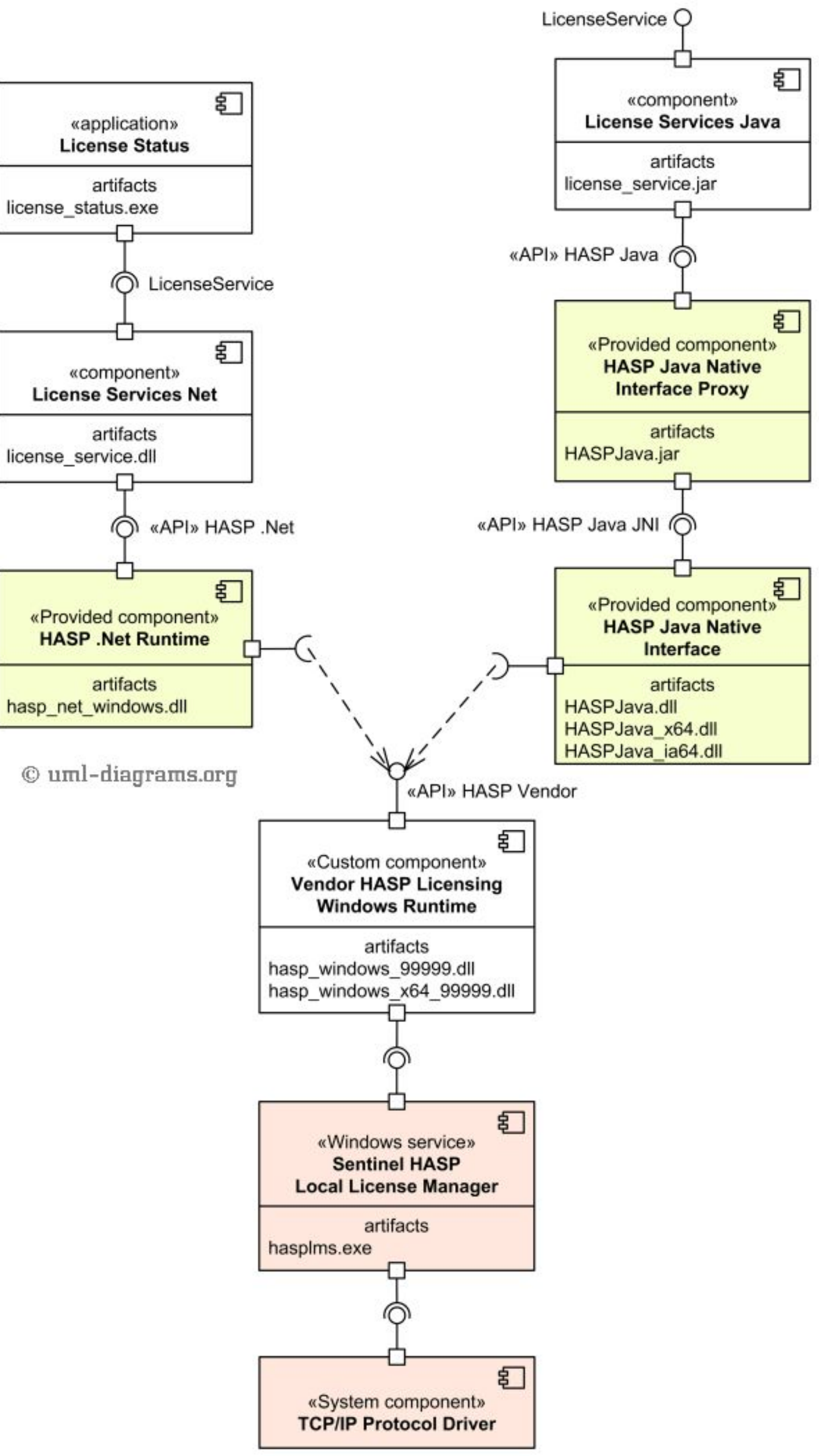
The SafeNet Sentinel LDK 6.1 Licensing API includes:

structural declarations and information on individual Sentinel Licensing API functions,

description of XML tags that can be used to define scope and output format of various API functions,

description of all API return codes.

Sentinel LDK installation includes API samples for various programming languages including C, C#, Java. For example, HASP Java API includes 4 Java classes: Hasp, HaspApiVersion, HaspTime, and HaspStatus in Aladdin package. Those classes are bundled into HASPJava.jar artifact. The HASP Java API classes load and link native methods from a platform-specific native library using the System.loadLibrary() method. As we said, for Microsoft Windows the DLL library loaded is one of the HASPJava.dll, HASPJava_x64.dll, or HASPJava_ia64.dll.


LicenseService
«component»
License Services Java
artifacts
license_service.jar
«application»
License Status
artifacts
license_status.exe
«API» HASP Java
LicenseService
«Provided component»
HASP Java Native
Interface Proxy
artifacts
HASPJava.jar
«component»
License Services Net
artifacts
license_service.dll
«API» HASP .Net
«API» HASP Java JNI
«Provided component»
HASP .Net Runtime
artifacts
hasp_net_windows.dll
«Provided component»
HASP Java Native
Interface
artifacts
HASPJava.dll
HASPJava_x64.dll
HASPJava_ia64.dll
© uml-diagrams.org
«API» HASP Vendor
«Custom component»
Vendor HASP Licensing
Windows Runtime
artifacts
hasp_windows_99999.dll
hasp_windows_x64_99999.dll
«Windows service»
Sentinel HASP
Local License Manager
artifacts
hasplms.exe
«System component»
TCP/IP Protocol Driver