

CA314 – Object Oriented Analysis & Design - 9

File name: CA314_Section_09_Ver01

Author: L Tuohey

No. of pages: 15

Table of Contents

9. Implementation Diagrams; Packages & Subsystems (see ref 1, Chap. 13, 14).....	3
9.1 Implementation Diagrams.....	3
9.1.1 Overview: Component & Deployment Models	3
9.1.2 Components (Ref. 1 & UML notation guide v1.4).....	3
9.1.3 Components (OMG UML, Version 2.2).....	6
9.1.3.1 Overview.....	6
9.1.3.2 Semantics	6
9.1.3.3 Notation (brief sample indication only).....	7
9.1.4 DEPLOYMENT MODEL	10
9.1.5 Further examples of Component & Deployment Models.....	11
9.2 Packages & Subsystems.....	12
9.2.1 Packages (14.1 of Ref. 1).....	12
9.2.1.1 Reasons for.....	12
9.2.1.2 Diagrams for packages.....	12
9.2.1.3 Name Space Control	13
9.2.1.4 <<import>> or <<access>>	13
9.2.2 Subsystems (14.2 of Ref. 1).....	14
9.3 Recap of UML Models	15

9. Implementation Diagrams; Packages & Subsystems (see ref 1, Chap. 13, 14)

9.1 Implementation Diagrams

9.1.1 Overview: Component & Deployment Models

Component Model	Deployment Model
<i>classifiers</i>	<i>instances</i>
Development View - Dependencies between parts of code	Physical & Process Views - Structure of run-time system, including hardware

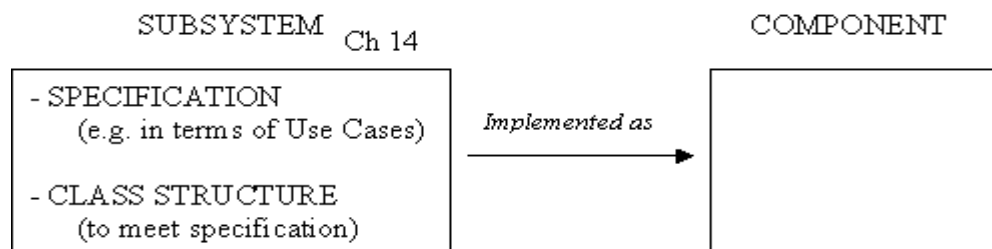
What is meant by a component?

Commonly, the term “component” is used fairly loosely to mean a “plug-in” part of a system. For the purposes of (this) section 9, a more specific concept is in mind. There have been some changes in this area between UML versions so that we indicate the older notation and terminology (Reference 1 & UML notation guide version 1.4) and the newer (OMG Unified Modeling Language™ (OMG UML), Superstructure, Version 2.2). However, it will be seen that the basic ideas have not really changed.

9.1.2 Components (Ref. 1 & UML notation guide v1.4)

Definition: A component is a "distributable piece of implementation of a system, including software code (source, binary, executable) but also including business documents etc. ..."

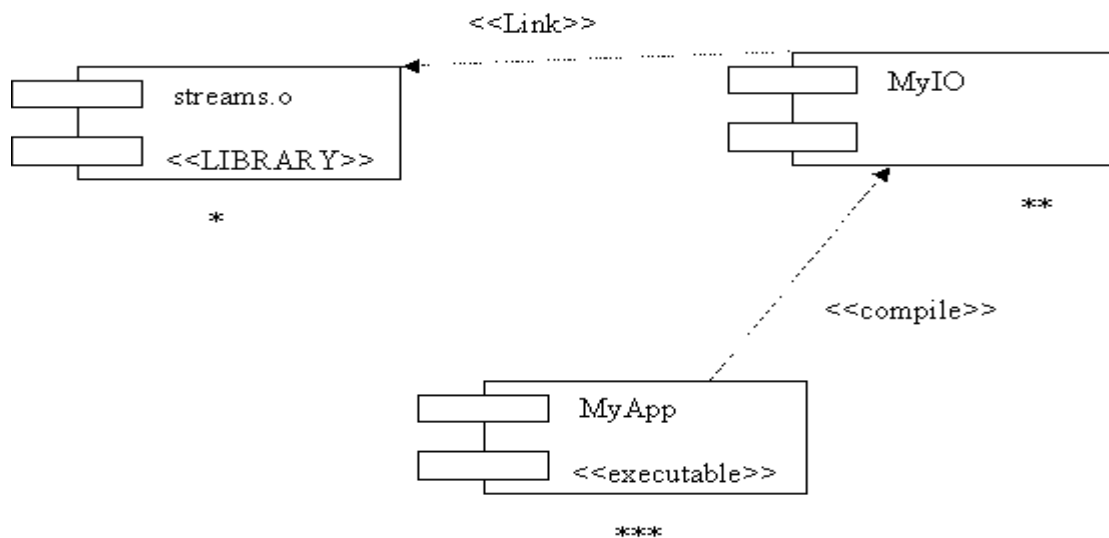
In this sense, we may think of a "component" as an implementation of a "subsystem" (**Chapter 14 of Ref. 1**), thus,



In UML (1.4), components in this sense are depicted by



Example: Compile-time dependencies for a C++ program - Fig 13.1 of Ref 1:

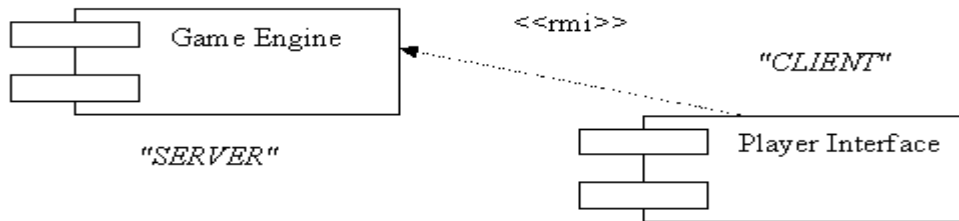


- * is a standard language I/O library
- ** MyIO is supposed to be a specially written I/O component
- *** MyApp is the application program

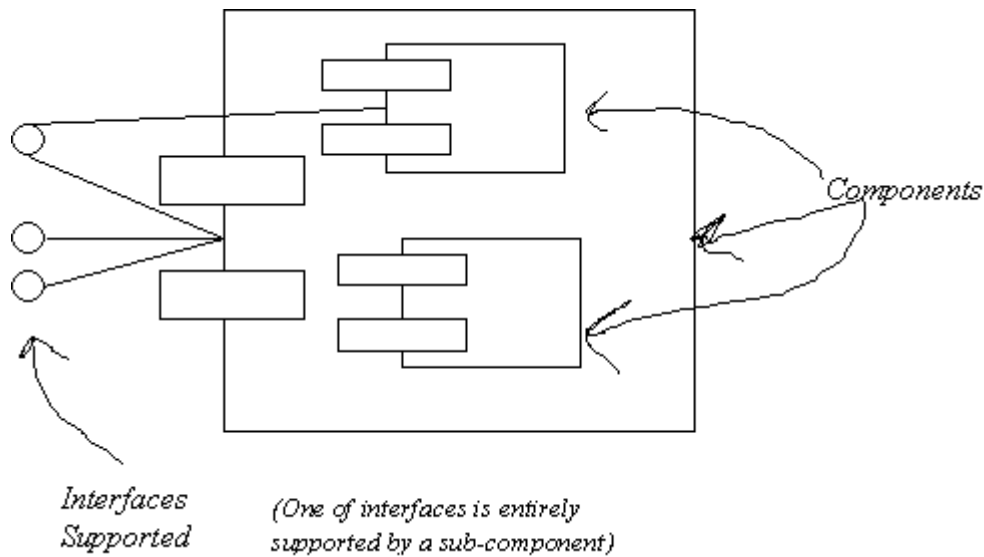
There are various common kinds of components, including,

Kind of Component	Dependency [names are suggestions only]
<u>Source code</u> (e.g. file containing code of a class)	<u><<compile>></u> Any components which have to be available when it is compiled
<u>Binary object code</u> (e.g. a class library)	<u><<link>></u> Any object code with which it must be linked to form an executable
<u>Executable application</u> (e.g. a bought-in spreadsheet, data base manager, etc)	<u><<rmi>></u> Any other executable programs it needs to interact with it at run-time

Example (Figure 13.2 of Ref. 1) showing run-time dependency:



General UML notation for components (including interfaces):



Note on "Classifiers and Instances"(Panel 13.1 of Ref. 1):

CLASSIFIER	INSTANCE
Class	Object
Use Case	Scenario
Actor	Actor
Association	Link
Component	Component
Subsystem	Subsystem

E.G. a component type might be an executable file while corresponding instances would be instances of the file that are running at the same time.

Note: Recall the *notation* to distinguish classifier and instance in UML, namely that the same icon is used for both but that underlining represents an instance.

9.1.3 Components (OMG UML, Version 2.2)

9.1.3.1 Overview

The following sections are taken from the “UML Superstructure Specification, v2.2” and are essentially intended to give a flavour of UML components. The specification is freely available on-line for anyone who wants to find out more.

9.1.3.2 Semantics

A component is a self contained unit that encapsulates the state and behavior of a number of classifiers. A component specifies a formal contract of the services that it provides to its clients and those that it requires from other components or services in the system in terms of its **provided** and **required interfaces**.

A component is a substitutable unit that can be replaced at design time or run-time by a component that offers equivalent functionality based on compatibility of its interfaces. As long as the environment obeys the constraints expressed by the provided and required interfaces of a component, it will be able to interact with this environment. Similarly, a system can be extended by adding new component types that add new functionality.

The required and provided interfaces of a component allow for the specification of structural features such as attributes and association ends, as well as behavioral features such as operations and events. A component may implement a provided interface directly, or, its realizing classifiers may do so, or they may be inherited. The required and provided interfaces may optionally be organized through **ports**, these enable the definition of named sets of provided and required interfaces that are typically (but not always) addressed at run-time.

A component has an **external view** (or “black-box” view) by means of its publicly visible properties and operations. Optionally, a behavior such as a protocol state machine may be attached to an interface, port, and to the component itself, to define the external view more precisely by making dynamic constraints in the sequence of operation calls explicit. Other behaviors may also be associated with interfaces or connectors to define the ‘contract’ between participants in a collaboration (e.g., in terms of use case, activity, or interaction specifications).

A component also has an *internal view* (or “white-box” view) by means of its private properties and realizing classifiers. This view shows how the external behavior is realized internally.

A number of UML standard stereotypes exist that apply to component. For example, «subsystem» to model large-scale components, and «specification» and «realization» to model components with distinct specification and realization definitions, where one specification may have multiple realizations.

9.1.3.3 Notation (brief sample indication only)

A component is shown as a Classifier rectangle with the keyword «component». Optionally, in the right hand corner a component icon can be displayed. This is a classifier rectangle with two smaller rectangles protruding from its left hand side.

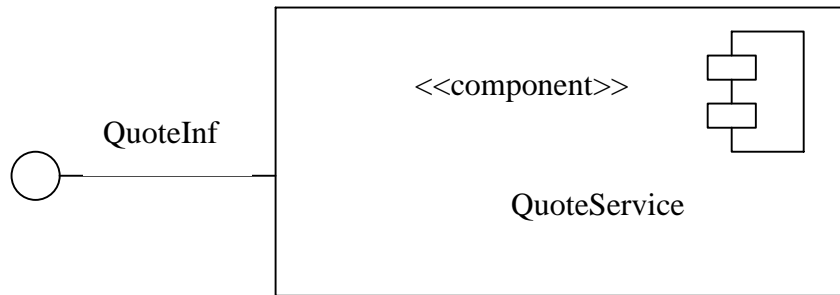


Figure 8.5 (UML spec 2.2) - A Component with one provided interface

An external view of a Component is by means of Interface symbols sticking out of the Component box (external, or black-box view):

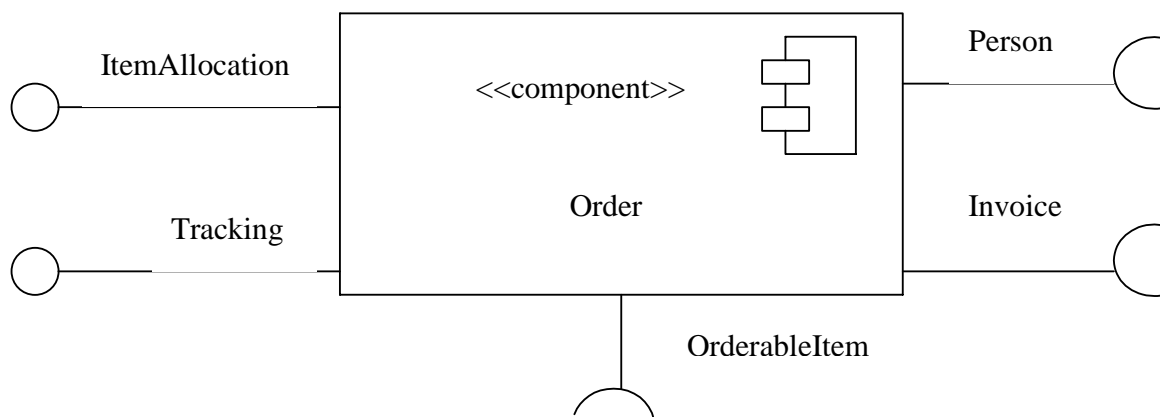


Figure 8.6 (UML spec 2.2) - A Component with 2 provided & 3 required interfaces

Alternatively, the interfaces and/or individual operations and attributes can be listed in the compartments of a component box (for scalability, tools may offer way of listing and abbreviating component properties and behavior):

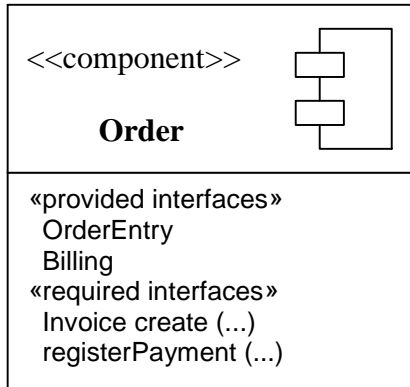


Figure 8.7 (UML spec 2.2) - Black box notation showing a listing of the properties of a component

For displaying the full signature of an interface of a component, the interfaces can also be displayed as typical classifier rectangles that can be expanded to show details of operations and events.

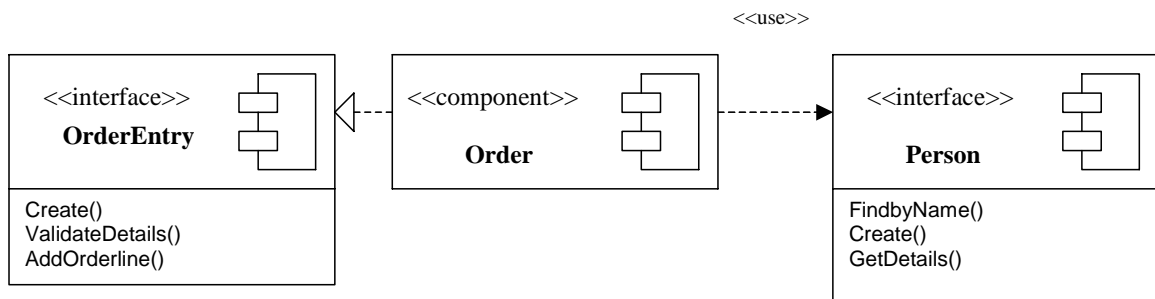


Figure 8.8 (UML spec 2.2)- Explicit representation of provided & required interfaces, allowing interface details such as operation to be displayed (if desired)

An internal, or white box view of a Component is where the realizing classifiers are listed in an additional compartment. Compartments may also be used to display a listing of any parts and connectors, or any implementing artifacts.

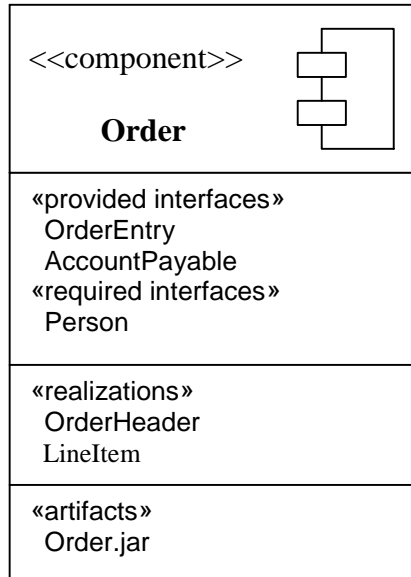


Figure 8.9 (UML spec 2.2) - A white-box representation of a component

9.1.4 DEPLOYMENT MODEL

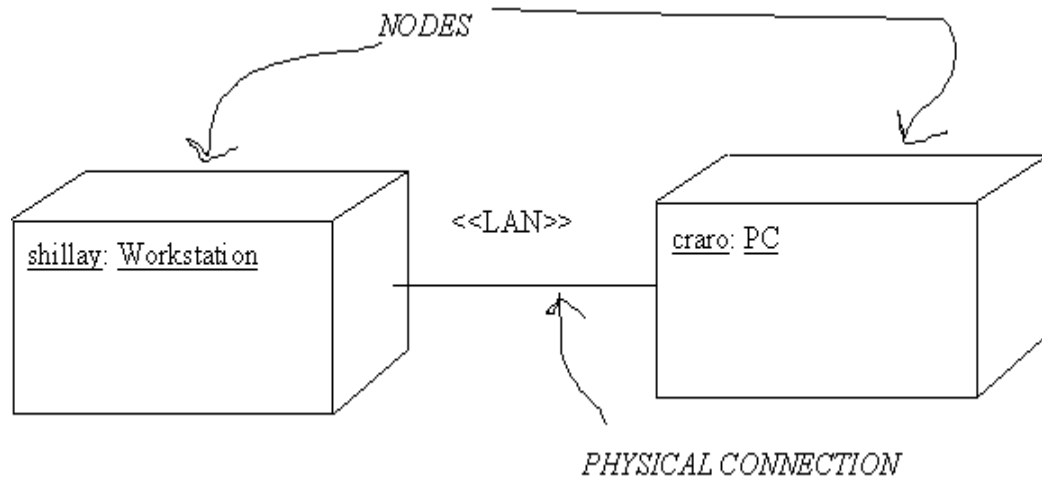


Figure 13.3 (of Ref. 1): A deployment model *without* the software

This type of diagram (model) allows depiction of physical connections between HW items.

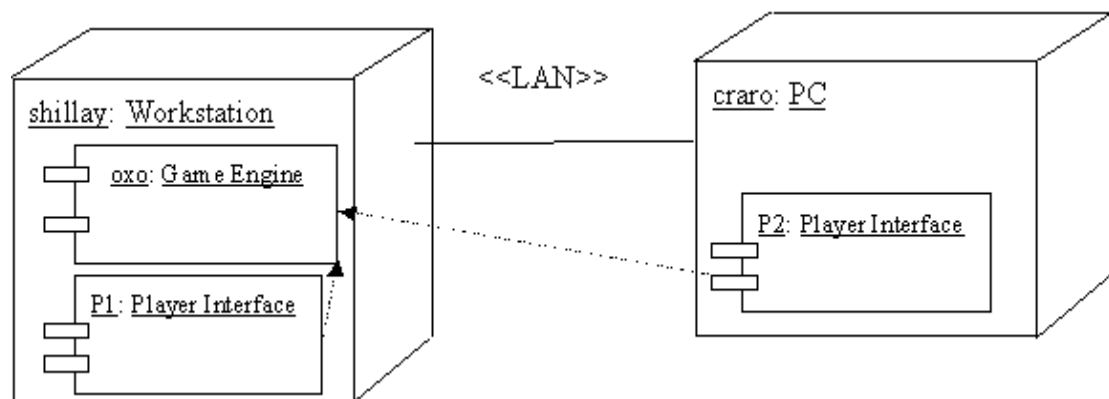
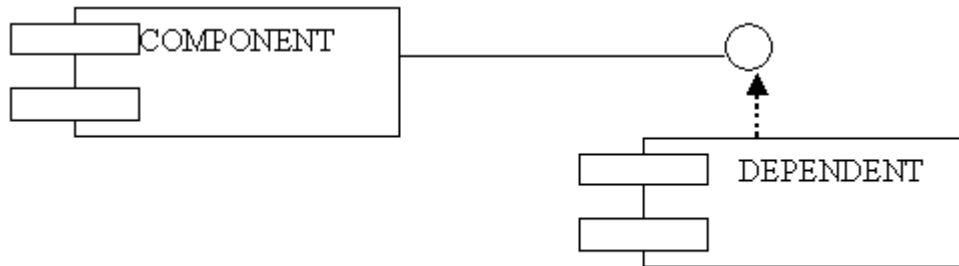


Figure 13.4 (of text): A deployment model *with* the software

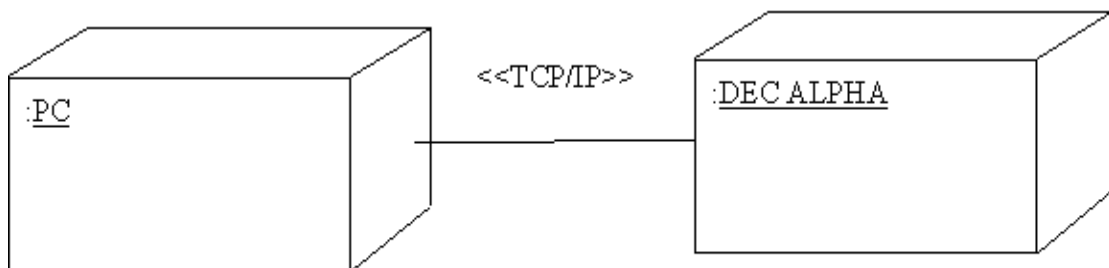
In this figure, the software components are running instances.

Suggestion: Read Panel 13.2 of Ref. 1 on when deployment issues are addressed in a project - should often be very early in a project's life-time.

9.1.5 Further examples of Component & Deployment Models



A Component Model: Dependency of a component on an interface of another component



A Deployment Model

9.2 Packages & Subsystems

- We discuss & define what is meant by a package and what its uses are.

Then, we define what is meant by a subsystem (Ref. 1, UML1.4), a special type of package.

Finally, we recapitulate the different kinds of models provided in UML.

- Packages overview:

- Why?
- Diagram (including hierarchy & "tree" option)
- Namespace control (including importing & accessing)

9.2.1 Packages (14.1 of Ref. 1)

9.2.1.1 Reasons for

- Convenience (e.g. to hide irrelevant detail)
- Allocate work among team members
- Specify and design a component

Probably use a "subsystem" package

9.2.1.2 Diagrams for packages

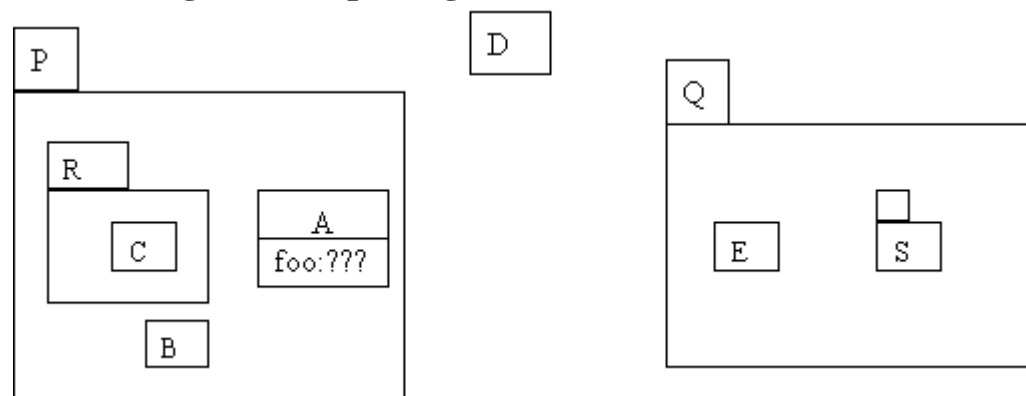


Fig 14.1 (of Ref. 1)

Note: It is implicit that there is a top-level package containing P, D and Q.

Points to note about package diagrams (as illustrated by Fig 14.1):

- A package is identified by "tab"
- May or may not (as desired) show what's inside a package
- Convenient convention is to write package name on its tab if its contents are shown, and within the "main" package rectangle otherwise

- Can use package symbols in all kinds of diagrams BUT an ordinary package (i.e. not a subsystem) cannot take part in an interaction as it is not a classifier
- One can also use a "tree" diagram for situations where packages contain sub-packages. For example, corresponding to the above diagram we have:

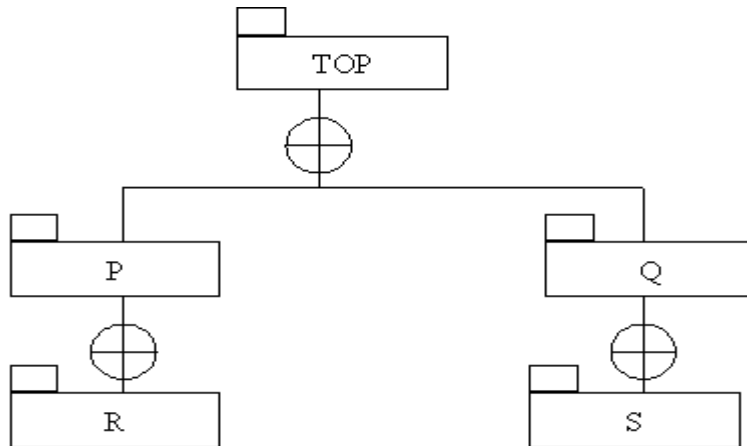


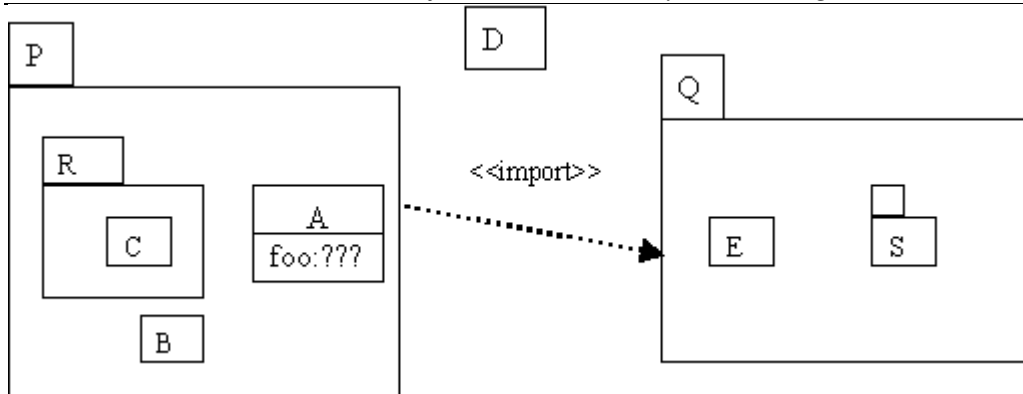
Fig. 14.2 (of Ref. 1): A hierarchy of packages

9.2.1.3 Name Space Control

- The only thing an ordinary package (i.e. not a subsystem) can do is to define the namespace of the elements in it. (This is similar to concepts of scope and visibility in programming languages).
- Notation "P::A" (refer to Figure 14.1) is a precise name for the class A contained in package P.
- Essentially, as for classes, you can have the same name for different things in different packages. No confusion results as full names are pre-fixed by the package name.
- *"Things outside a package cannot see in"* (but things inside a package can *"see out"*)
- In Fig. 14.1, the class of attribute "foo" could be "B" or "D" but not "C" or "E" (as the latter two are not in namespace of "A").

9.2.1.4 <<import>> or <<access>>

- Both of these stereotypes allow some element inside "P" to name something in another package "Q" that does not contain "P". For example, with reference to Figure 14.1, if we draw



then the effect would be to make Q's elements visible to those of P. Hence, "foo" could now have (*i.e. be of*) class "E".

- <<import>> - In the above case, where <<import>> is used, elements of "P" can refer to elements of "Q" just as if they were in "P" (e.g. "foo: E").
- <<access>> - On the other hand, if we had used <<access>> rather than <<import>>, elements of "P" would have to include a prefix in references to elements of "Q" (e.g. "foo: Q::E").

9.2.2 Subsystems (14.2 of Ref. 1)

- A subsystem is a package that has a specification and a realization part.
- Subsystem diagram: May use stereotype <<subsystem>> or a special symbol on the "tab" (as below) to identify a package as a subsystem:

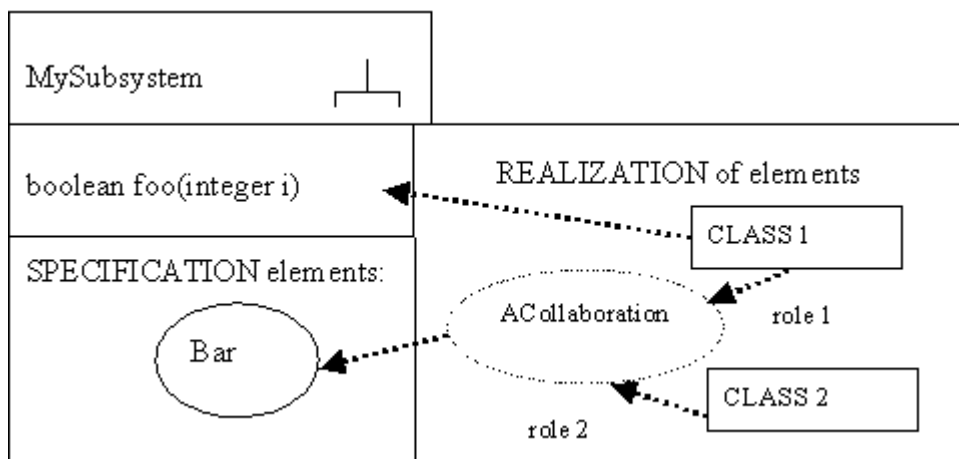


Fig. 14.3 of Ref. 1

Note: Here "dotted ellipse" is shorthand for a collaboration which is assumed to be defined in more detail elsewhere.

- **Specification part:**

-- describes operations that can be done without revealing internal structure (could include Use Cases)

-- can match interfaces (see earlier on components)

- **Realization part:**

-- May contains classes and other subsystems

Note: A subsystem may or may not be instantiable (depending on how it is defined)

9.3 Recap of UML Models

View	Models & Diagrams
Use Case view	Use Case model
Logical view	Class model (+ interaction and state diagrams as necessary)
Process view (threads of control)	Interaction & activity diagrams (as necessary) + Deployment diagrams
Development view	Component diagrams (+ (maybe) packages and subsystems)
Physical view	Deployment diagrams