

Purpose of these pages:

To supplement the material in the course section “UMLessentialsIntroClassModels”

5.1.4 Real world objects Vs their system representation(supplement)

It is interesting to read the classic paper “On the Criteria To Be Used in Decomposing Systems into Modules” (D.L. Parnas) ([here](#)) which is considered to have been one of the first to look at the disadvantages of functional decomposition, the merits of information hiding, etc. Its conclusion reads:

“We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing. To achieve an efficient implementation we must abandon the assumption that a module is one or more subroutines, and instead allow subroutines and programs to be assembled collections of code from various modules.”

5.2 Associations (supplement)

The text book (section 5.2) aims to give a fairly brief explanation of associations and dependencies in UML but may perhaps be a bit too compressed. The following notes are intended to clarify the UML terminology by taking some extracts from /*/.

References:

/*/ OMG Unified Modeling Language Specification, Version 1.5, March 2003

/**/ “Software Design, from Programming to Architecture”, Braude (Wiley)

(a) Overall Clarification of terms (from Braude //):**

“**Dependency**, denoted by a dotted line arrow, means that one class depends upon another in the sense that if the class at an arrow’s end were to change, this would affect the dependent class. Strictly speaking, dependency includes inheritance and aggregation. However, these relationships have their own notation, and so we usually reserve dependency to indicate that a method of one class utilizes another class.” [*but there are other types of dependency, see (c) below*]

“**Association**, denoted with a solid line between two classes, commonly means that objects of each class depend on objects of the other in a structural way. ... Associations are useful in the early stages of building class relationships when we know a pair to be related, but are postponing the ultimate design of the relationship.”

(b) Some definitions from Glossary of /*/

association The semantic relationship between two or more classifiers that specifies connections among their instances.

behavioral feature A dynamic feature of a model element, such as an operation or method.

classifier A mechanism that describes behavioral and structural features. Classifiers include interfaces, classes, datatypes, and components.

dependency A relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).

generalization A taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed. See: *inheritance*.

relationship A semantic connection among model elements. Examples of relationships include associations and generalizations

structural feature A static feature of a model element, such as an attribute.

trace A *dependency* that indicates a historical or process relationship between two elements that represent the same concept without specific rules for deriving one from the other.

usage A *dependency* in which one element (the client) requires the presence of another element (the supplier) for its correct functioning or implementation.

Note: **Permission** is a kind of dependency. It grants a model element permission to access elements in another namespace.... The predefined stereotypes of Permission are access, import, and friend. ... In the case of the **friend** stereotype, the client is granted permission to reference elements in the supplier namespace, regardless of visibility.

(c) From *UML*, Section “3.51 Dependency”

3.51 Dependency

3.51.1 Semantics

A dependency indicates a semantic relationship between two model elements (or two sets of model elements). It relates the model elements themselves and does not require a set of instances for its meaning. It indicates a situation in which a change to the target element may require a change to the source element in the dependency.

3.51.2 Notation

A dependency is shown as a dashed arrow between two model elements. The model element at the tail of the arrow (the client) depends on the model element at the arrowhead (the supplier). The arrow may be labeled with an optional stereotype and an optional individual name.

...

The following [Table A] kinds of Dependency are predefined and may be indicated with keywords. ... All of these are shown as dashed arrows with keywords in guillemets [i.e. <<...>>].

Note: We will come across several of these dependency types as we progress through UML. We just note their existence for now.

Keyword	Name	Description
access	Access	The granting of permission for one package to reference the public elements owned by another package ...
bind	Binding	A binding of template parameters to actual values to create a nonparameterized element. ...
derive	Derivation	A computable relationship between one element and another ...
import	Import	The granting of permission for one package to reference the public elements of another package, together with adding the names of the public elements of the supplier package to the client package. ...
refine	Refinement	A historical or derivation connection between two elements with a mapping (not necessarily complete) between them. ...
trace	Trace	A historical connection between two elements that represents the same concept at different levels of meaning. ...
use	Usage	<u>A situation in which one element requires the presence of another element for its correct implementation or functioning.</u> May be stereotyped further to indicate the exact nature of the dependency, such as <u>calling an operation of another class</u> , granting permission for access, instantiating an object of another class, etc. Maps into a Usage. If the keyword is one of the stereotypes of Usage (call, create, instantiate, send), then it maps into a Usage with the given stereotype.

Table A: Kinds of Dependency

Note – The connection between a note or constraint and the element it applies to is shown by a dashed line without an arrowhead. This is not a Dependency. {We mention it here as it occurs in the following example]

3.51.4 Example

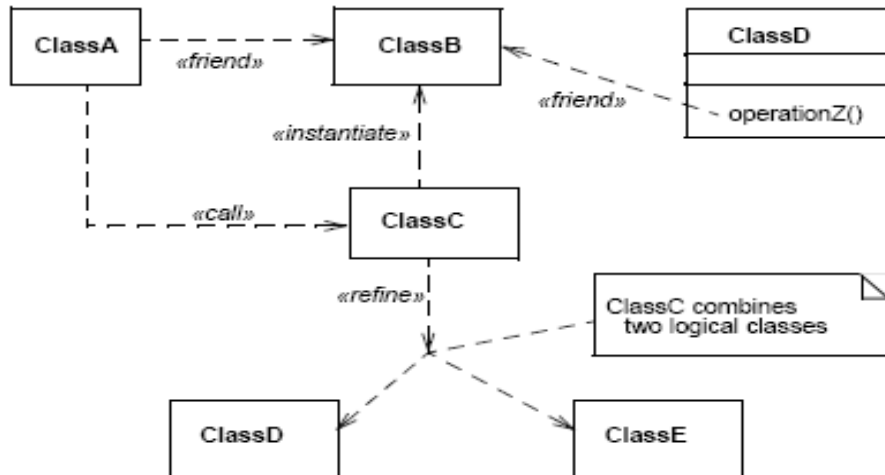


Figure 3-50 Various Dependencies Among Classes

Re 5.4.2 Implementing generalization: inheritance [or delegation?](supplement)

From Glossary of /*/, we have the definitions

delegation The ability of an object to issue a message to another object in response to a message. Delegation can be used as an alternative to inheritance.

[Also elsewhere in /*/ we have the further explanation:

An operation map (a set of operation-method pairs) is attached to an object. If a request type matches an operation in the map, the corresponding method is executed. If the operation is not found, then a link points to another object that is then searched. This is the concept of *delegation*.]

inheritance The mechanism by which more specific elements incorporate structure and behavior of more general elements related by behavior.