



The Module View Type

Part 6



What is a Module

- A module is an implementation unit of software that provides a coherent unit of functionality
 - Packages
 - Modules
 - Classes
 - General grouping of source units



The Module View Type (I)

Elements	The element of a module view is a module, which is a unit of functionality that implements a set of responsibilities
Relations	<p>Relations shown in a module view will be some form of <i>is part of</i>, <i>depends on</i>, or <i>is a</i>.</p> <ul style="list-style-type: none">• <i>A is part of B</i> defines a part-whole relation between the submodule A (the part, or child) and the aggregate module B (the whole, or parent).• <i>A depends on B</i> defines a dependency relation between A and B. Specific module styles will elaborate what dependency is meant.• <i>A is a B</i> defines a generalization relation between a more specific module (the child A) and a more general module (the parent B).



The Module View Type (II)

Properties of elements	<p>Properties of a module include:</p> <ul style="list-style-type: none">• name, which may have to comply to rules such being a member of a namespace• responsibilities of the module• visibility of the module's interface(s) [applies when the relation is a form of <i>is part of</i>]• implementation information, such as the set of code units that implement the module
------------------------	---



The Module View Type (III)

Properties of relations	<ul style="list-style-type: none">• the <i>is part of</i> relation may have a visibility property associated with it that defines if a submodule is visible outside the aggregate module• the <i>depends on</i> relation can have constraints assigned to specify in more detail what the dependency between two modules is.• The <i>is a</i> relation may have an implementation property, denoting that a more specific module (the child A) inherits the implementation of the more general module (the parent B), but does not guarantee to support the parent's interface and thereby does not provide substitutability for the parent.
Topology	The module viewtype has no inherent topological constraints.



Uses of Module View Type

- Construction
 - Blueprints for the source code
- Analysis
 - Traceability
 - Impact analysis
- Communication
- Not used for inference about runtime behavior

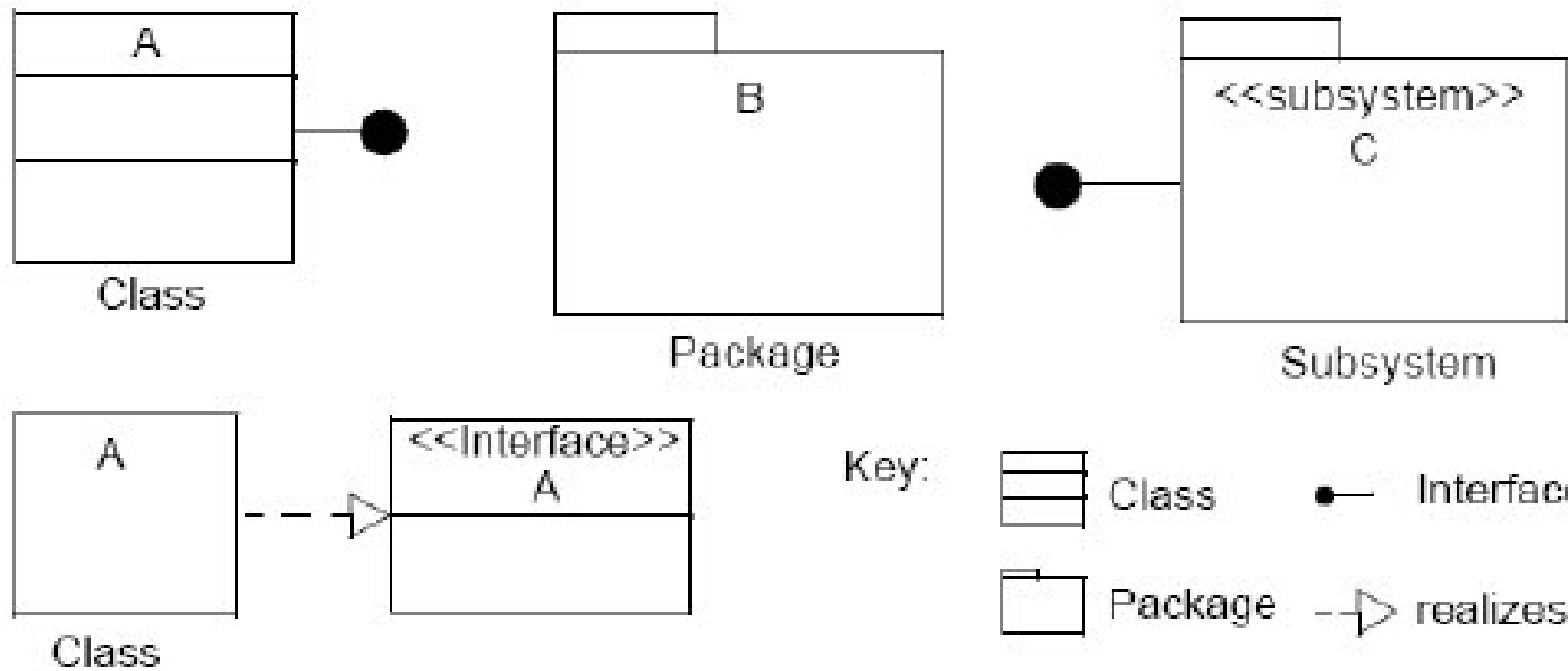
Notations for Module View Types



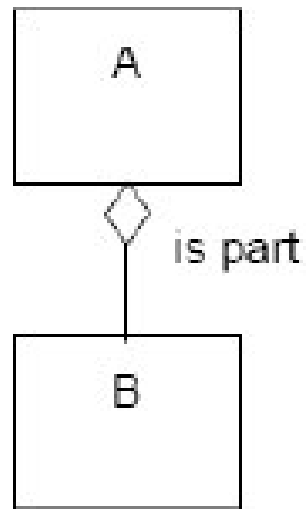
Informal Notations:

- Box and line
 - Lines represent some kinds of depends on relationship
 - Lollipop interfaces
 - Containment represents is part of relationship
- Textual representation
 - Indention or outline numbering for is part of relationship

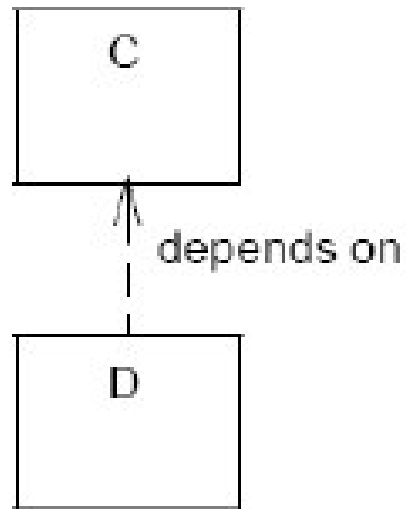
Modules in UML



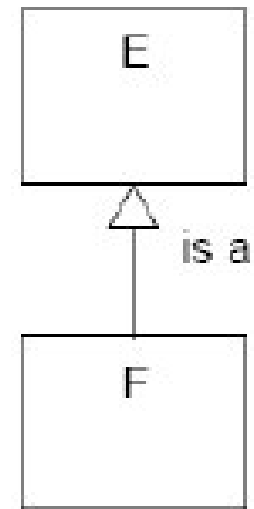
Relationships in UML



Aggregation



Dependency



Generalization

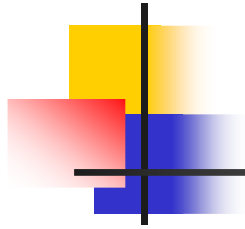
Key:

 Class



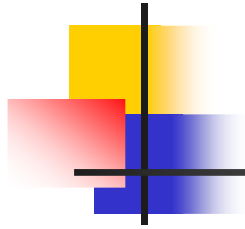
Relation with other view types

- Module views commonly mapped into C&C views
 - one-to-one mapping
 - one-to-many mappings
 - more complex mappings
- Overloading module views with C&C information



Styles of Module View Type

- Decomposition
- Uses
- Generalization
- Layers



Decomposition Style

- Focusing on is part of relation on module view type
- How system responsibilities are partitioned across modules
- How modules are decomposed into submodules



A Useful Style

- Almost all architectures begin with it
- Favorite tool for communication with newcomers
- Allocating functionality to specific places in architecture



Decomposition Criteria

- Achievement of certain qualities
- Build versus buy
- Product lines



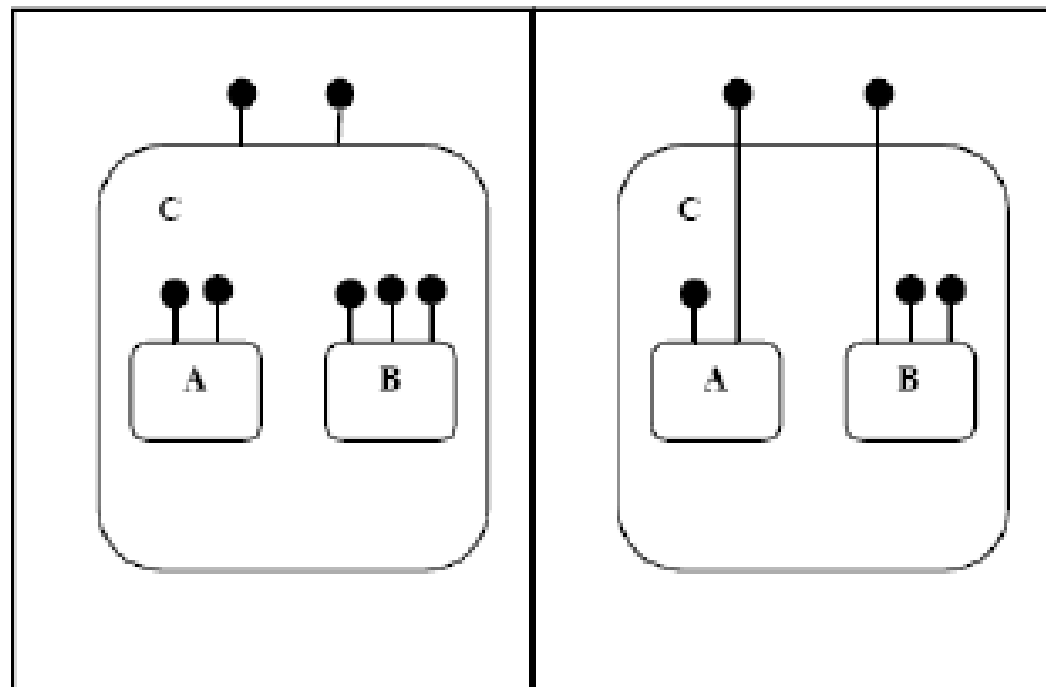
Elements, Relationships, Properties

Elements	Module, as defined by the module viewtype. A module that aggregates other modules is sometimes called a subsystem.
Relations	The relation is the decomposition relation, which is a refined form of the is-part-of relation. A documentation obligation includes specifying the criteria used to define the decomposition.
Properties of elements	As defined by the module viewtype
Properties of relations	<ul style="list-style-type: none">• Visibility, the extent to which the existence of a module is known, and its facilities available, to those modules outside its parent.
Topology	<ul style="list-style-type: none">• No loops are allowed in the decomposition relation.• A module cannot be part of more than one module.

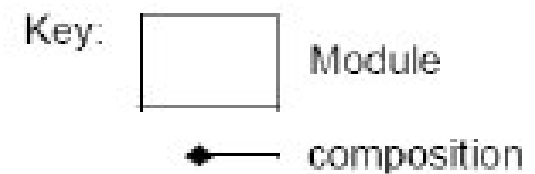
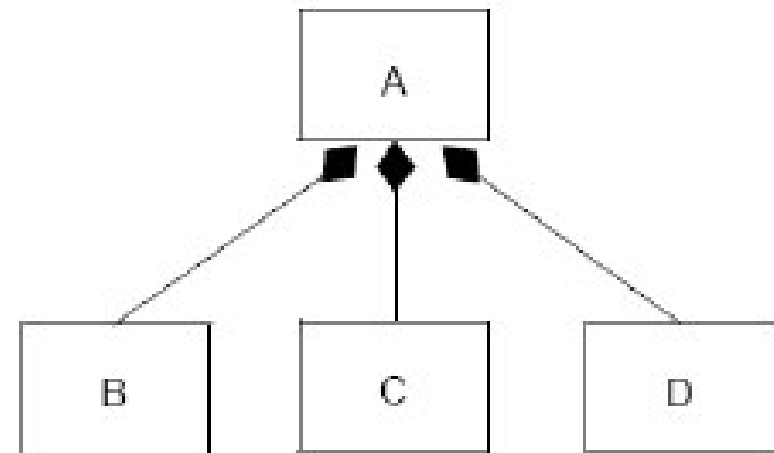
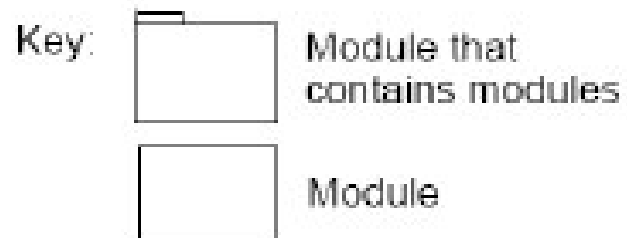
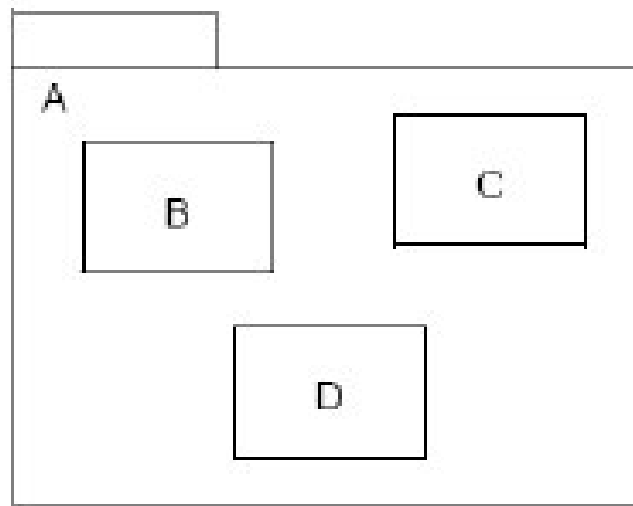


Informal Notations

Key:  Module
 Module interface



UML Notations





Uses Style

- Focusing on depends-on relations
- Constraining the implementation of architecture
- Enabling incremental development and deployment



Elements, Relationships, Properties

Elements	Module as defined by the module viewtype.
Relations	The relation is the uses relation, which is a refined form of the depends-on relation. Module A uses module B if A depends on the presence of a correctly functioning B in order to satisfy its own specification.
Properties of elements	As defined by the module viewtype.
Properties of relations	The uses relation may have a property that describes in more detail what kind of uses one module makes of another.
Topology	The uses style has no topological constraints. However, if there are loops in the relation that contain many elements, the ability of the architecture to be delivered in incremental subsets will be impaired.



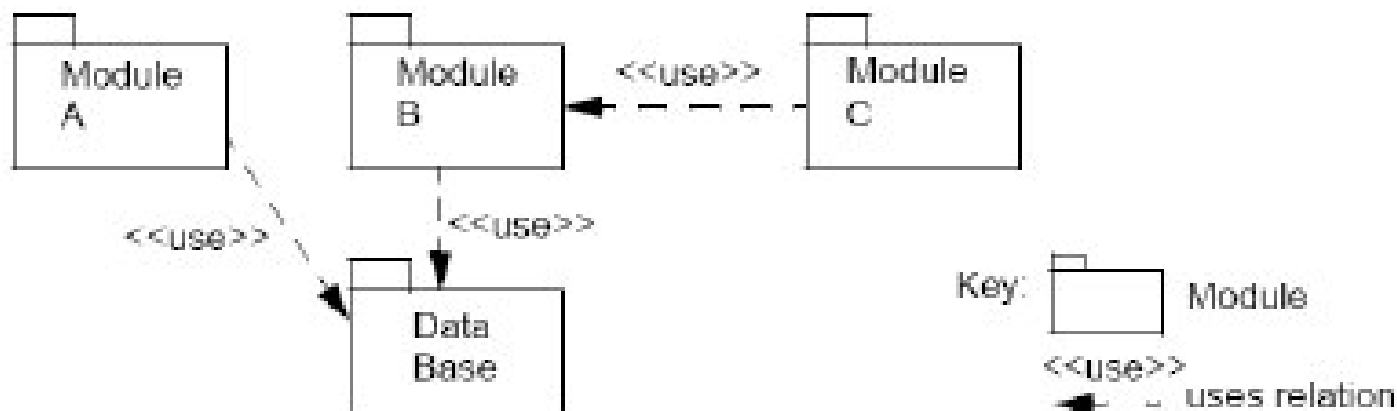
Informal Notations

- Cross-references

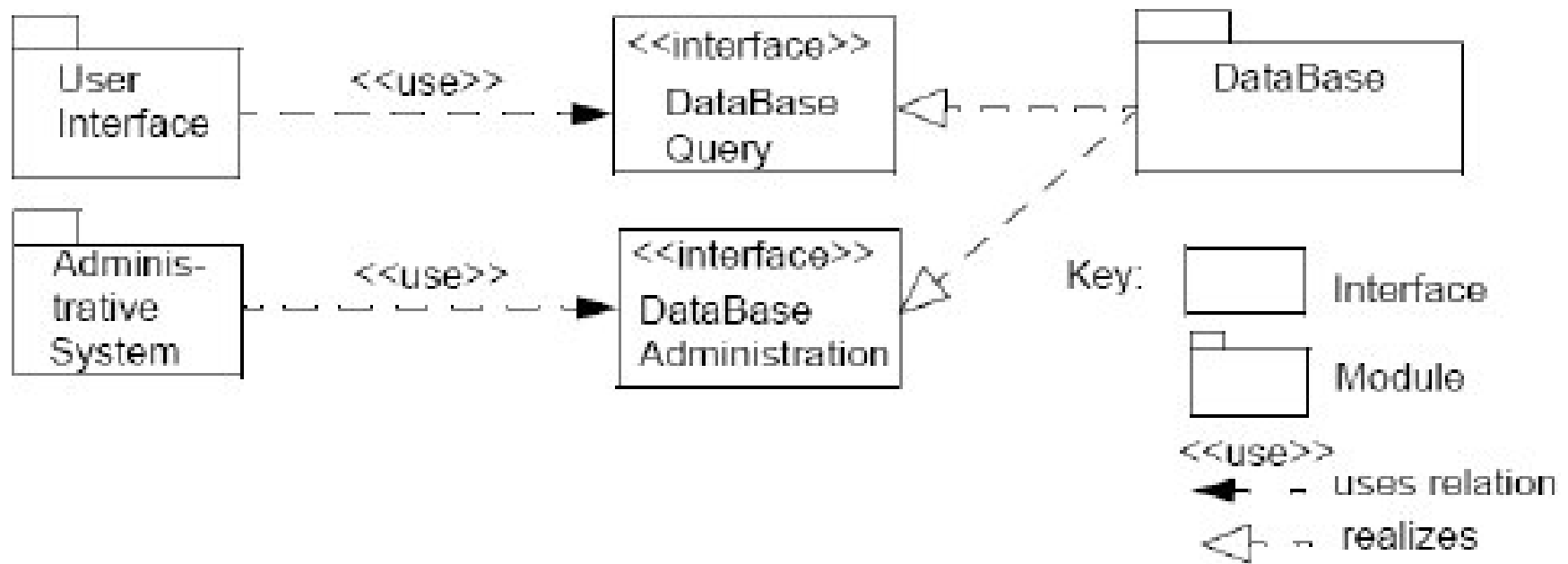
	Mod. A	Mod. B	Mod. C
Mod. B	X		
Mod. C		X	

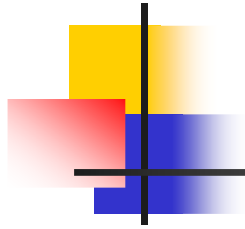
- Two-column tables
- Box and line!

UML Notations (I)



UML Notations (II)





Generalization Style

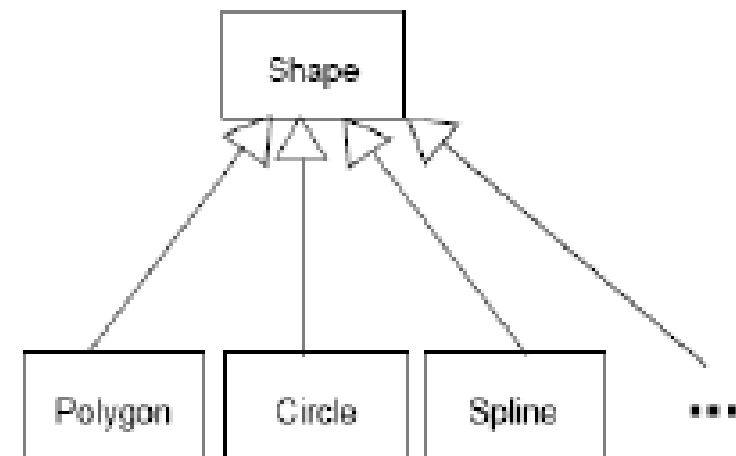
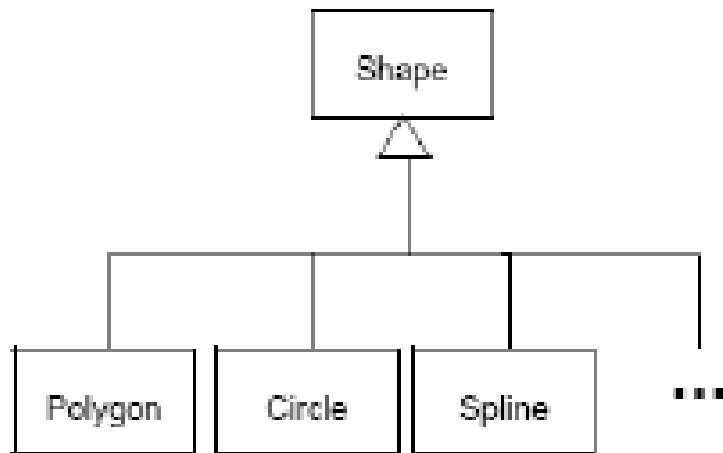
- Focusing on is-a relation
- Supports extension and evolution
- Implies inheritance of implementation and interface



Elements, Relationships, Properties

Elements	Module as defined by the module viewtype.
Relations	Generalization, which is the <i>is a</i> relation as in the module viewtype
Properties of elements	Beside the properties defined for a module in the module viewtype, a module can have the "abstract" property, that defines a module with interfaces but no implementation.
Properties of relations	The generalization relation can have a property that distinguishes between interface and implementation inheritance. In case a module is defined as an abstract module (the abstract property) then restricting the generalization relationship to implementation inheritance is not meaningful.
Topology	<ul style="list-style-type: none">• a module can have multiple parents, although multiple inheritance is considered a dangerous design approach in many places.• Circles in the generalization relation are not allowed, that is any of the child modules cannot be a generalization of one or more of its parent modules.

UML Notations (I)



Key:

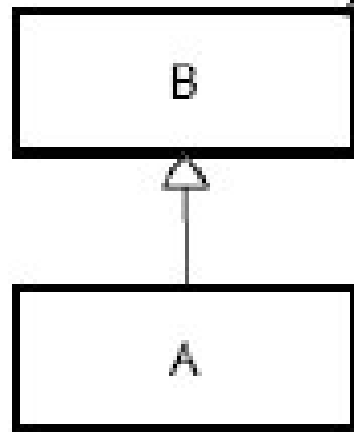


Generalization

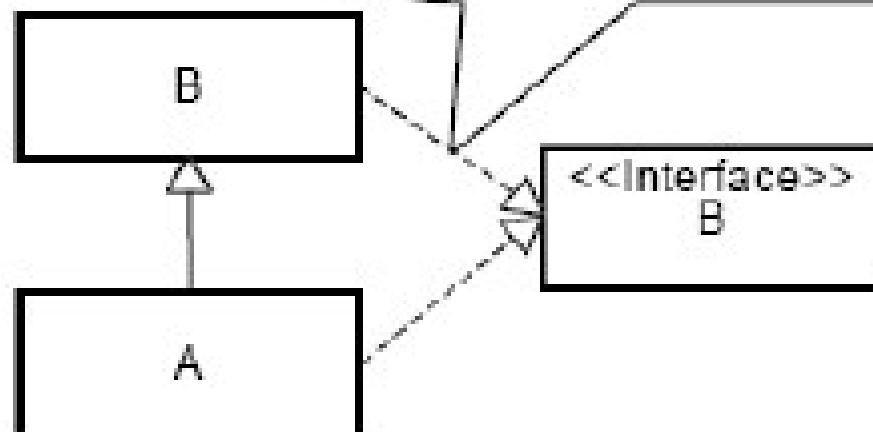
Indicator that more modules may be added

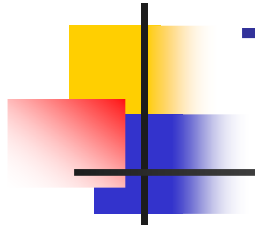
UML Notations (II)

Example 1: Generalization
relation without properties
A representation like this ...



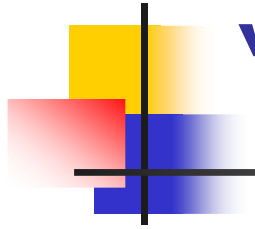
... usually means Module A
inherits the implementation
of module B and realizes the
same interface as module B





The Layered Style

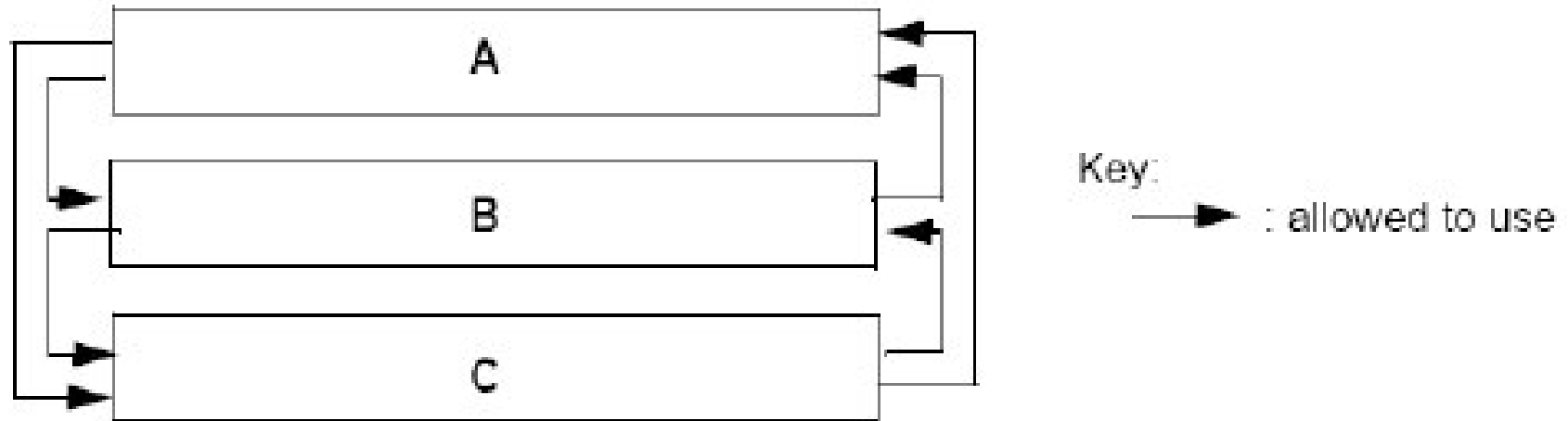
- A division of software into units (layers)
- Each layer is a virtual machine
 - Enhancing portability
- Constraints on the relationships between layers



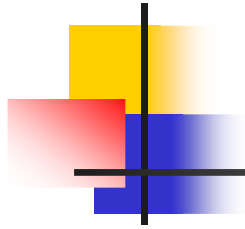
“allowed-to-use” Relations

- Using facilities of the immediately lower layer
- Using facilities of any lower layer
 - Bridging layers
 - Many of these: poorly structured system

Bad Example



- Callbacks are used to eliminate upward usage



Some Observations

- Layers cannot be determined by examining source code
- A layer may provide unused services
 - More general design
 - Imported from somewhere else

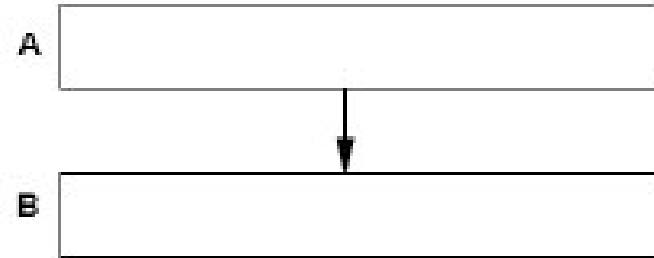


Elements, Relationships, Properties

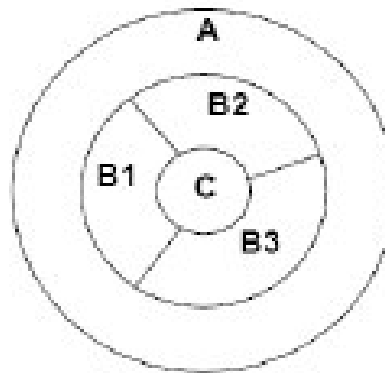
Elements	Layers.
Relations	Allowed to use, which is a specialization of the module viewtype's generic "depends on" relation. P1 is said to use P2 if P1's correctness depends upon having a correct version of P2 present as well.
Properties of elements	<ul style="list-style-type: none">• Name of layer• The units of software the layer contains• The software a layer is allowed to use: (software in next lower layer, software in any lower layer, exceptions to downward-only allowable usage)• The interface to the layer.• The cohesion of the layer -- a description of the virtual machine represented by the layer
Properties of relations	<ul style="list-style-type: none">• The relation is transitive.
Topology	<ul style="list-style-type: none">• Every piece of software is allocated to exactly one layer.

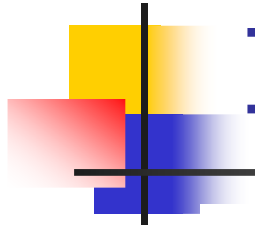
Informal Notations (I)

- Stacks



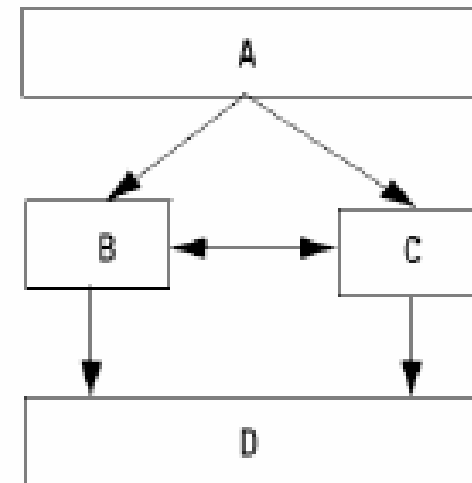
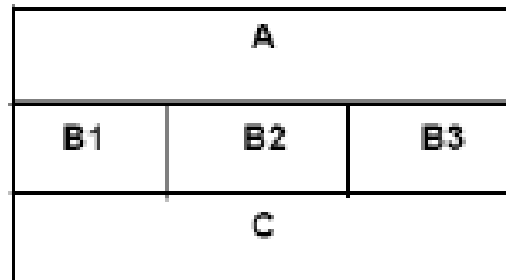
- Rings



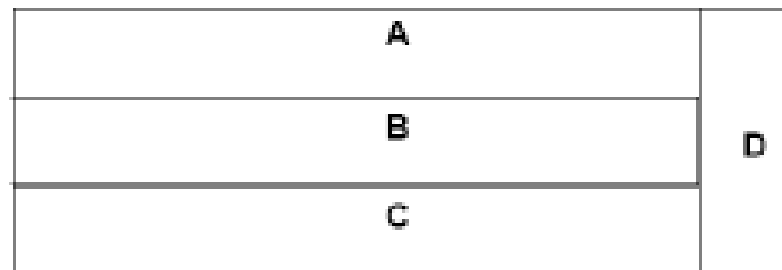


Informal Notations (II)

- Segmented Layers

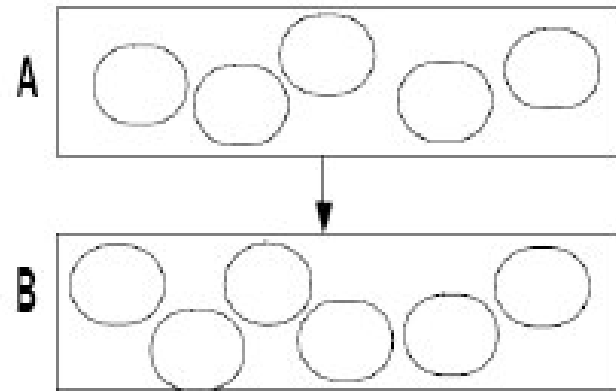


- Layers with a sidecar

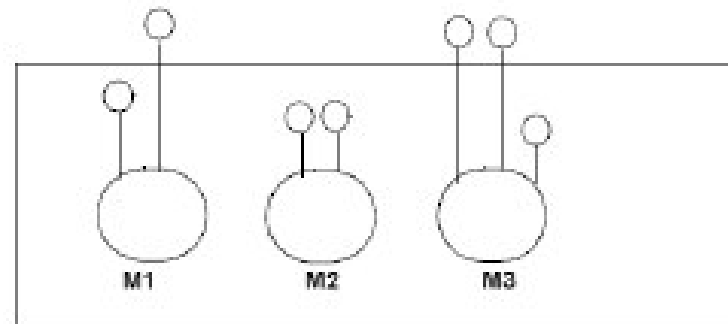


Informal Notations (III)

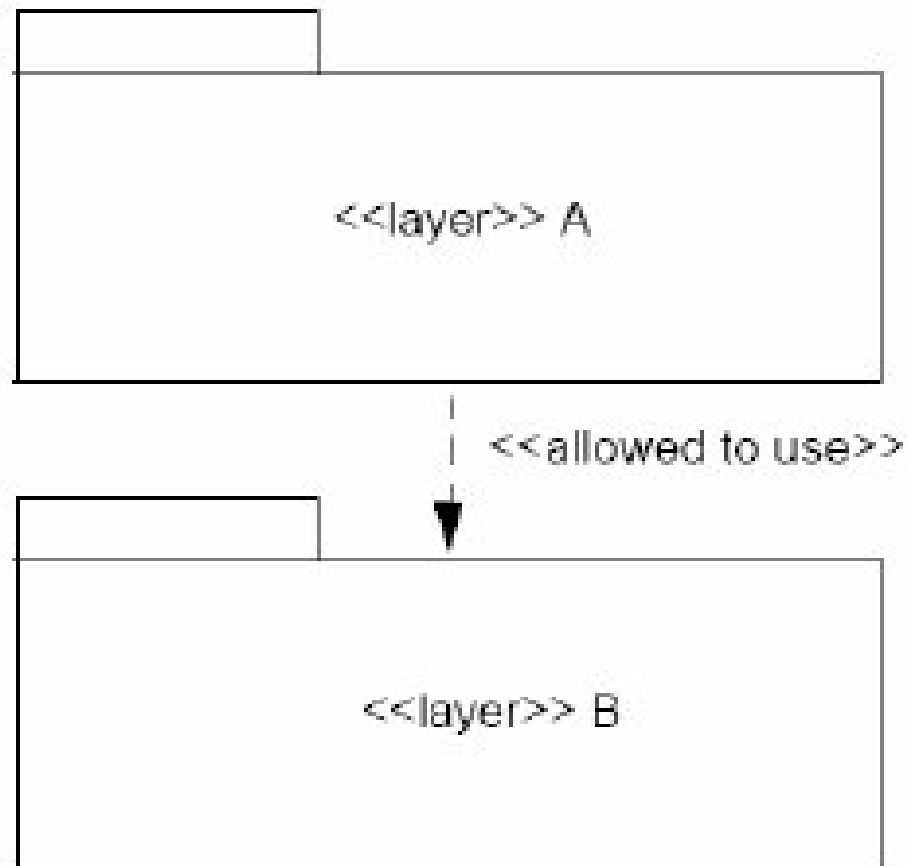
- Contents



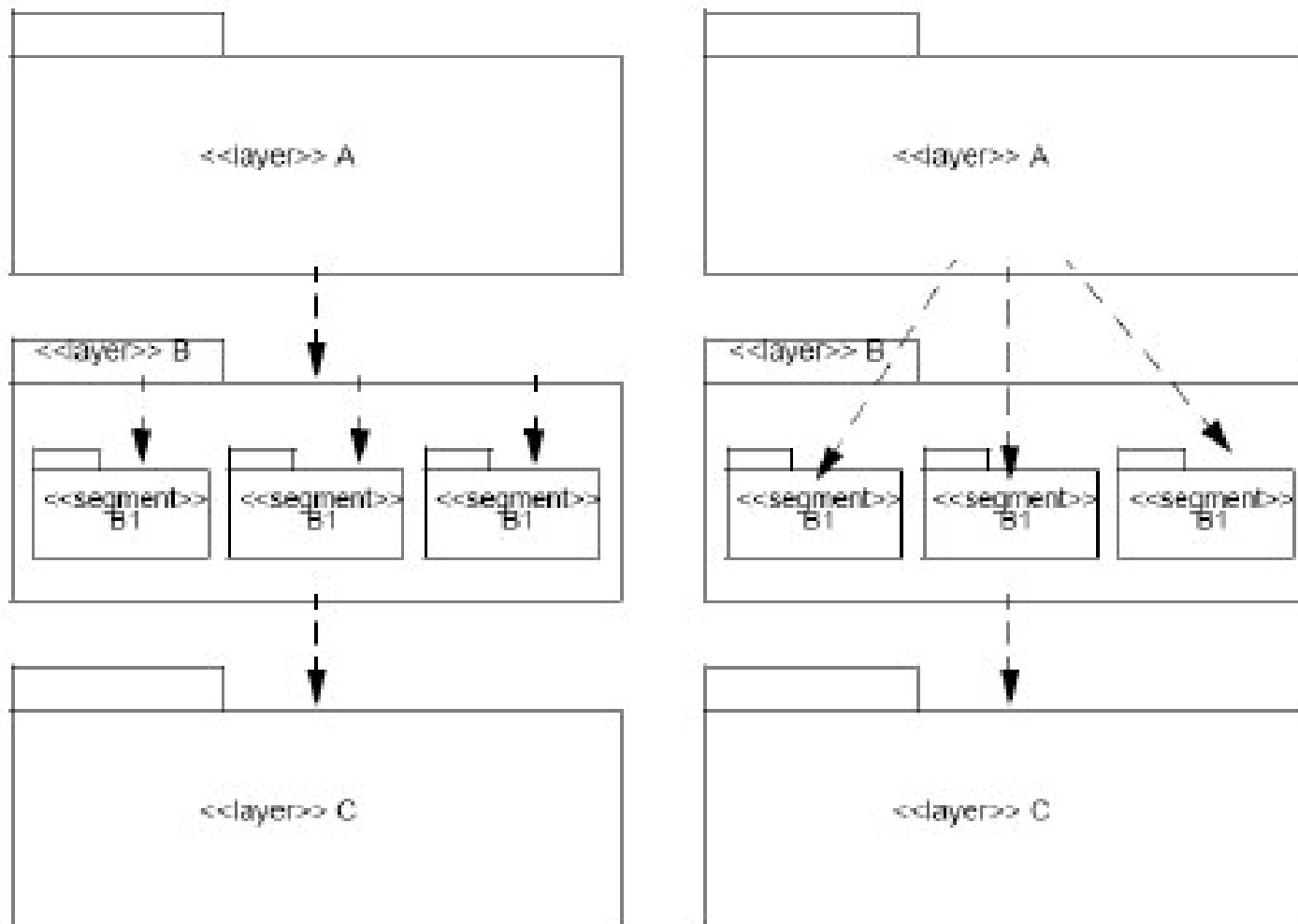
- Interfaces
- Size and Color



UML Notations (I)



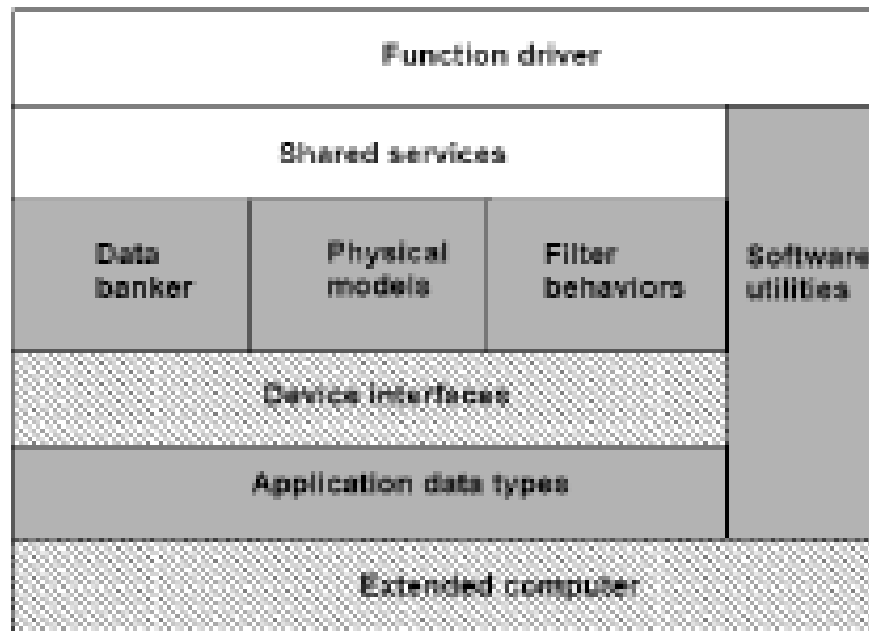
UML Notations (II)





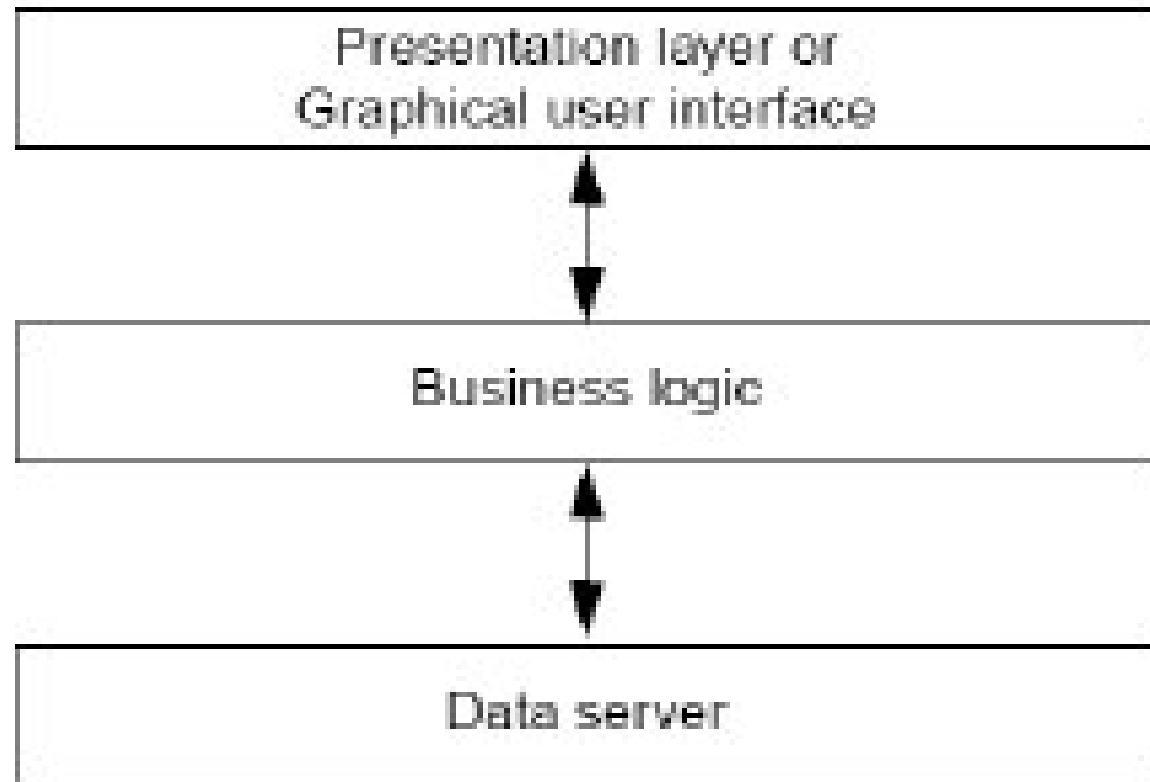
Relation to Other Styles (I)

- Module Decomposition
 - Recursive decomposition is not allowed
 - Segments map to modules



Relation to Other Styles (II)

- Tiers





Relations to Other Styles (III)

- Module "uses" style
 - Differentiation between "allowed-to-use" and "uses"
- Subsystems