WHITESTEIN Technologies

Unified Modeling Language

# Generic Modeling Mechanisms

*Radovan Cervenka*

## Model, Element, Diagram and  Element View

**Model**

- → A set of modeling elements and diagrams used to represent the relevant aspects of the modeled system.
- Specialized package.

**Element**

- → A fundamental constituent of a model.
- An abstract common superclass for all metaclasses in UML.

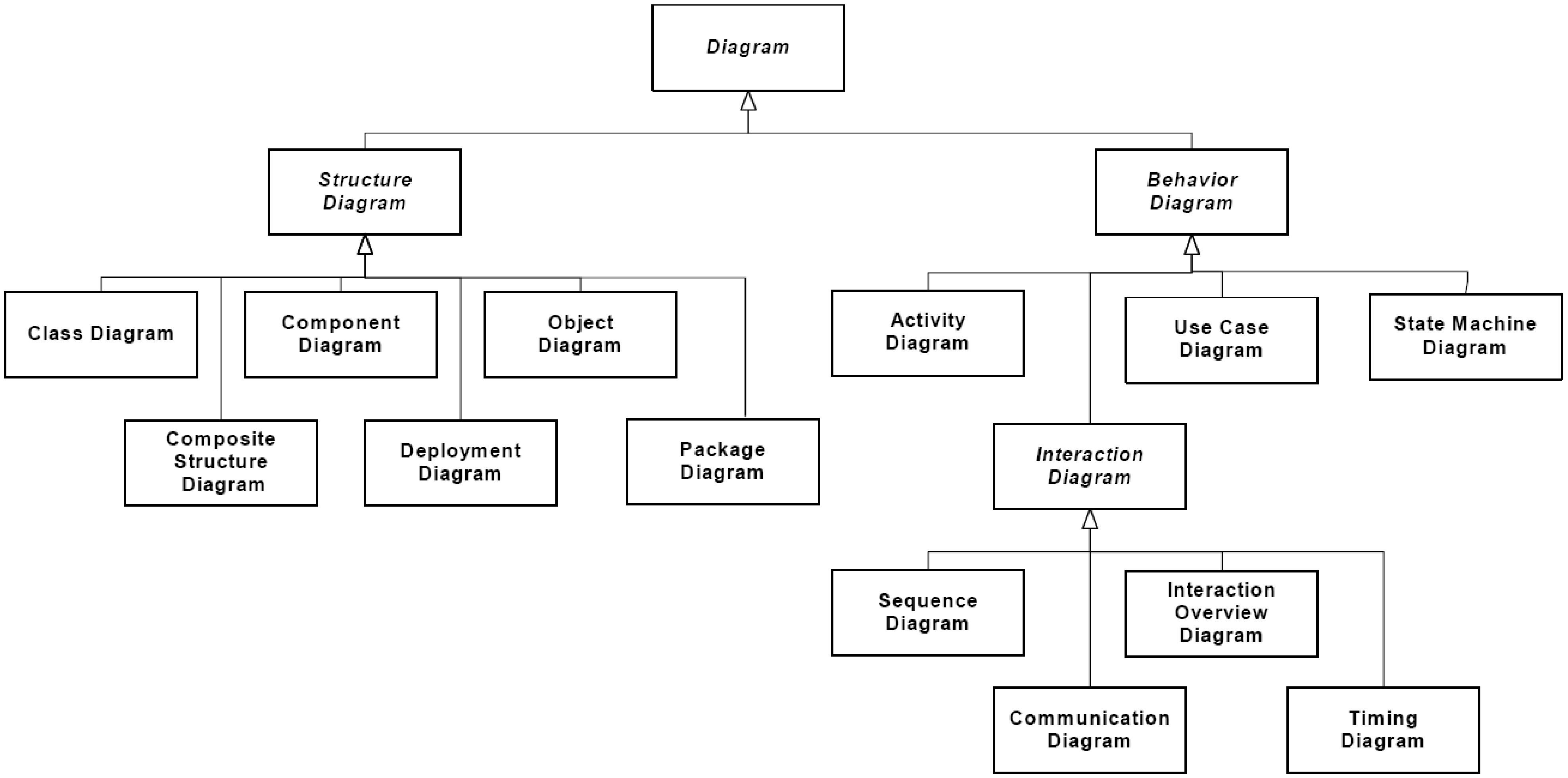
**Diagram**

- → Graphical representation of parts of the UML model.
- UML diagrams contain graphical elements (nodes connected by paths) that represent elements in the UML model.

**Element view**

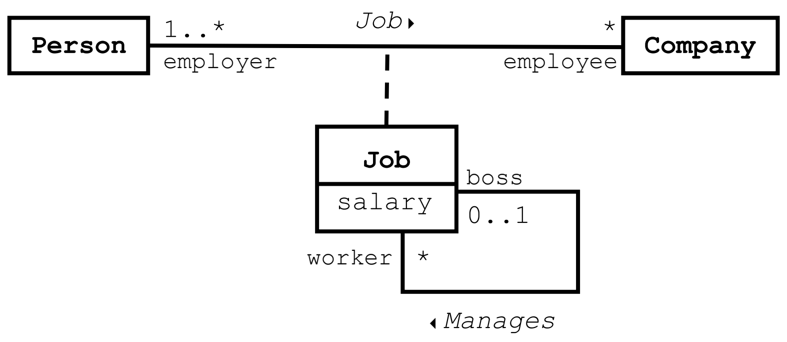
- → Graphical representation of a single element depicted in a diagram.
- One element can have several views, possibly placed in different diagrams.

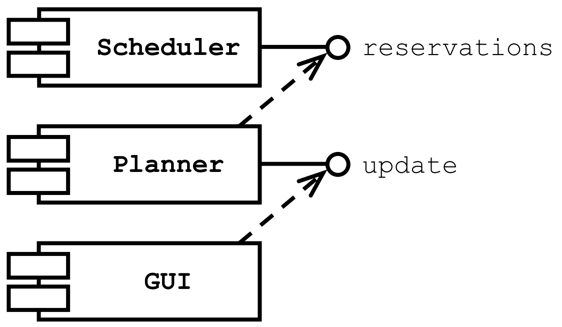
## Taxonomy of UML Diagrams

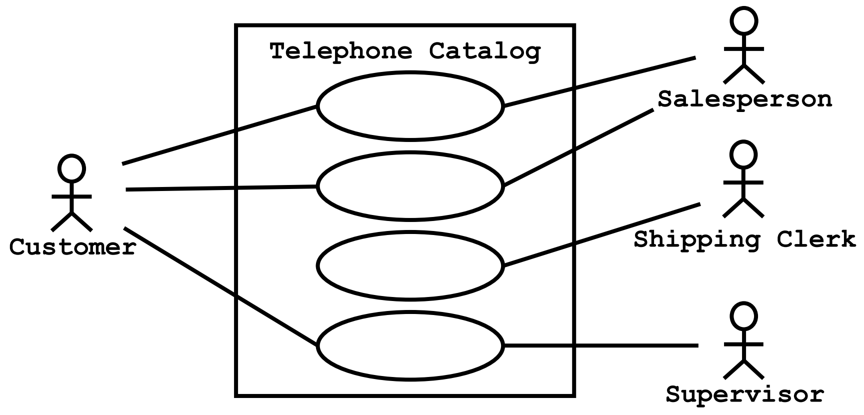


## Diagram Elements

→ Generic notational mechanisms used in various ways in other parts of the language.







## Graphs, Drawing Paths, Hyperlinks, ...

**Graphs and their content**

- UML diagrams are mainly graphs.
- Information is mostly in the topology.
- Graphical constructs: icons, 2-d symbols, paths and strings.

**Drawing Paths**

→ A series of line segments whose endpoints coincide.

**Invisible hyperlinks and the hole of tools**

- Arrangement of model information into a “hyperdocument”.
- Dynamic notation is specific for a particular tool.
- Out of the scope of UML.

**Background information**

- Suppression of a model/element information.
- Textual or tabular format of background information.
- Out of the scope of UML.

## String, Name and Label

**String**

→ A sequence of characters (of any character set).

**Name**

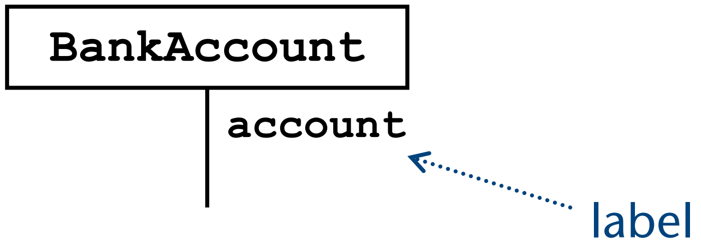
→ A string uniquely identifying a *named element*.

- Defined within a *namespace*.
- May be linked together by delimiters into a pathname.

```
BankAccount, controller, long_underscored_name,
MathPack::Matrices::BandedMatrix.dimension
```

**Label**

→ A string that is attached to a graphical symbol.



## Keyword and Expression

**Keyword**

- → A name reserved by UML.
- • Used in stereotypes and tagged values.

«*keyword*»

{ *keyword* }

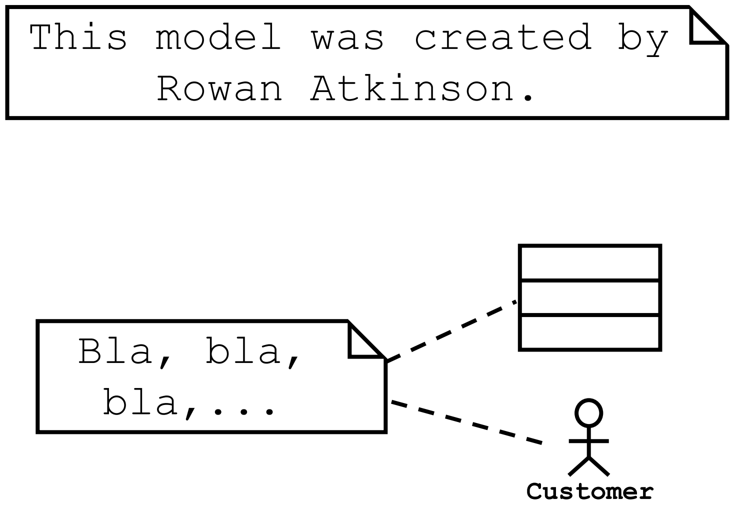
**Expression**

- → Linguistic formulas that yield values when evaluated in run-time.
- • Language-dependent.

```
BankAcount
BankAccount * (*) (Person*, int)
array [1..20] of range(-1.0 .. 1.0) of Real
[i > j and self.size > i]
```

## Comment

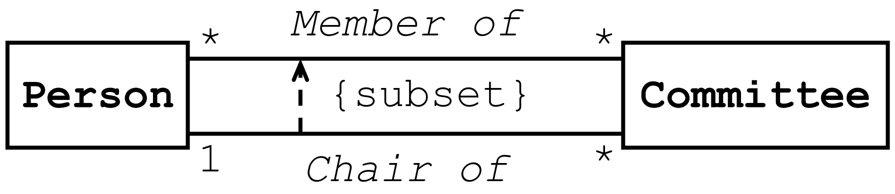
→ A textual annotation that can be attached to a set of elements.

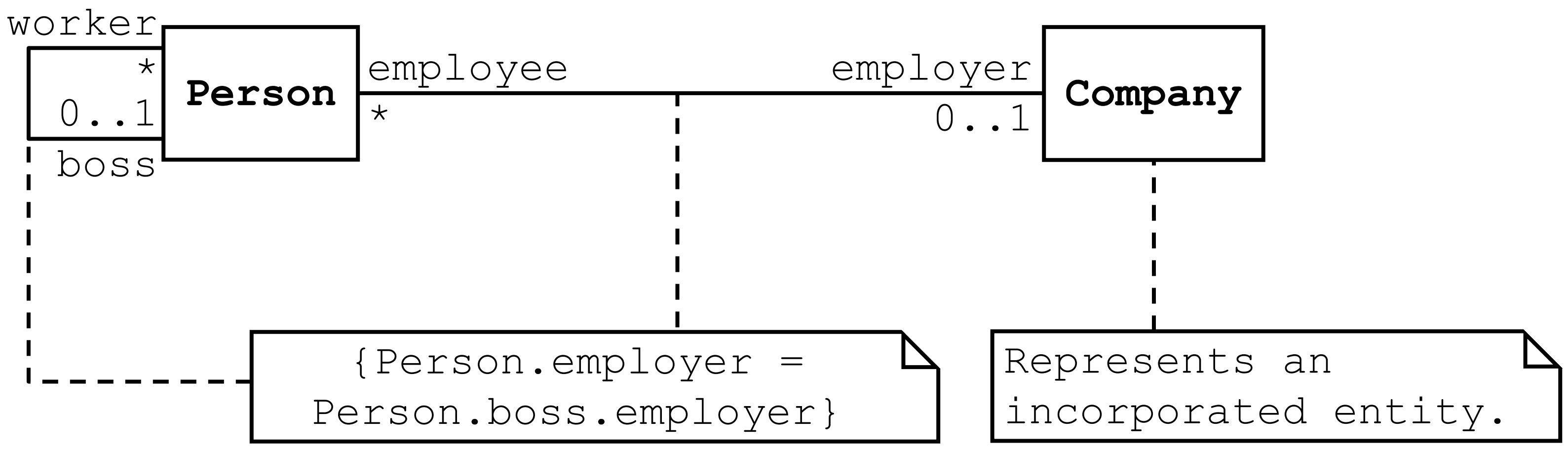


## Constraint

→ A condition or restriction expressed in natural language text or in a machine readable language (e.g. OCL) for the purpose of declaring some of the semantics of an element.

{*name*: *boolean-expression*}




worker
boss
Person
employee
employer
Company
*
0..1
*
0..1
{Person.employer = Person.boss.employer}
Represents an incorporated entity.

## Tagged Values (Property String)

- → A set of keyword-value pairs attached to a model element.
- ■ Keyword (tag) identifies the type of a property.
- ■ Value determines the property's value.
- ■ If the type is Boolean and the value is omitted ⇒ True
- ■ Can be used as an element in a list.
  - It applies to all subsequent elements.

{*keyword* = *value*, *keyword* = *value*, ... }

```
{author=„John“,deadline=15-June-98,status=Design}
{abstract}
```

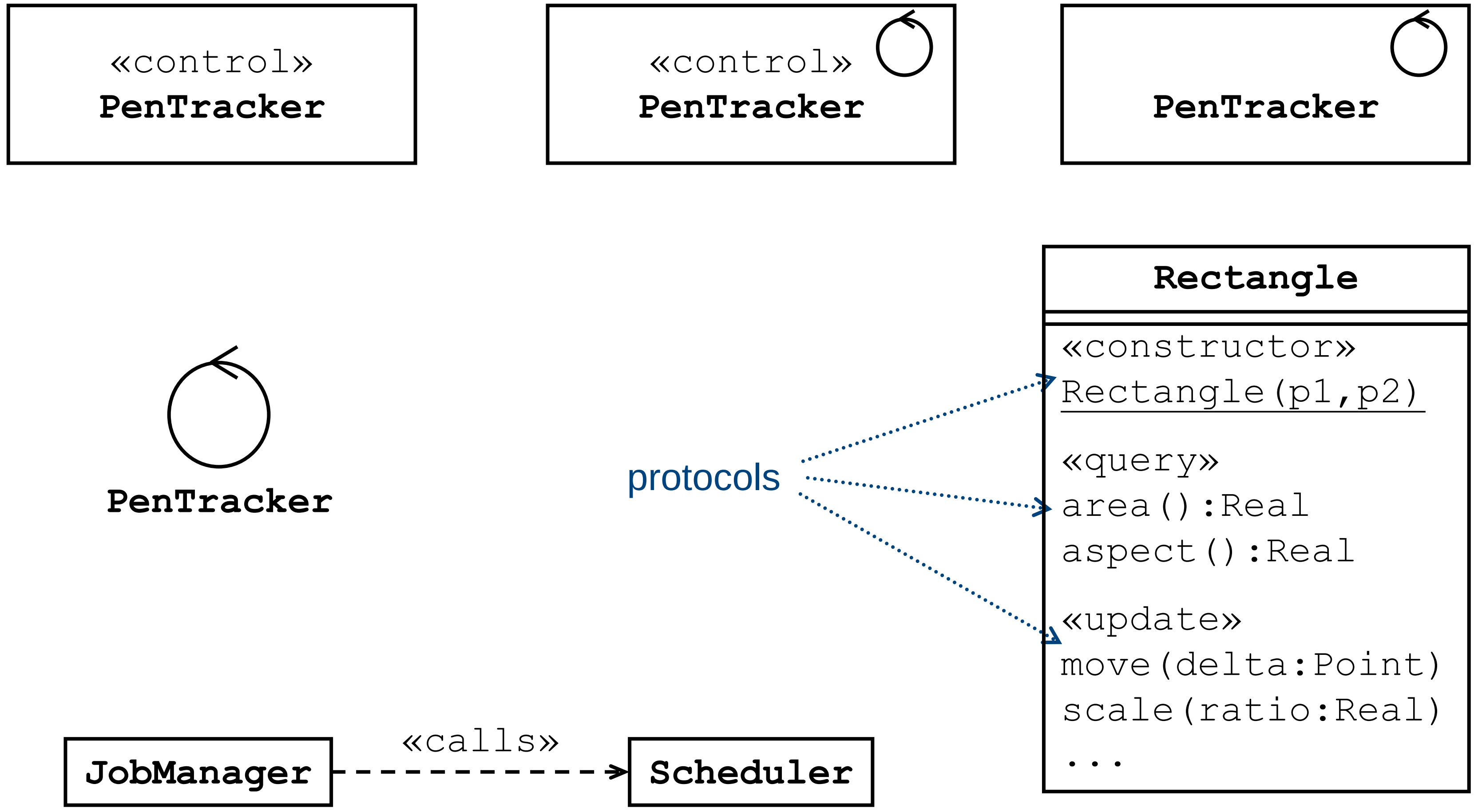
## Stereotypes

- → A new type of modeling element introduced at modeling time.
- → Specialization (a special meaning) of existing modeling element type with the same form but a different intent/semantics.
- ■ Can be used with any standard UML element type.
- ■ Enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended element types.
- ■ Can be used as an element in a list.
  - It applies to all subsequent elements.

*«stereotype name»*

and/or an icon

# Examples of Stereotypes



## Classifier-Instance Correspondence

**Classifier**

- → A classification of instances, it describes a set of instances that have features in common.

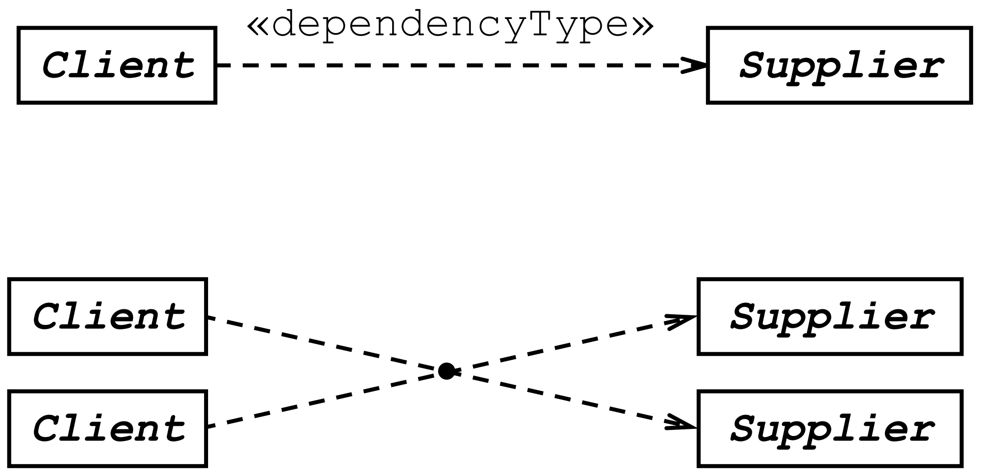
**Instance**

- → An instance in a modeled system. It can be classified by one or more classifiers.

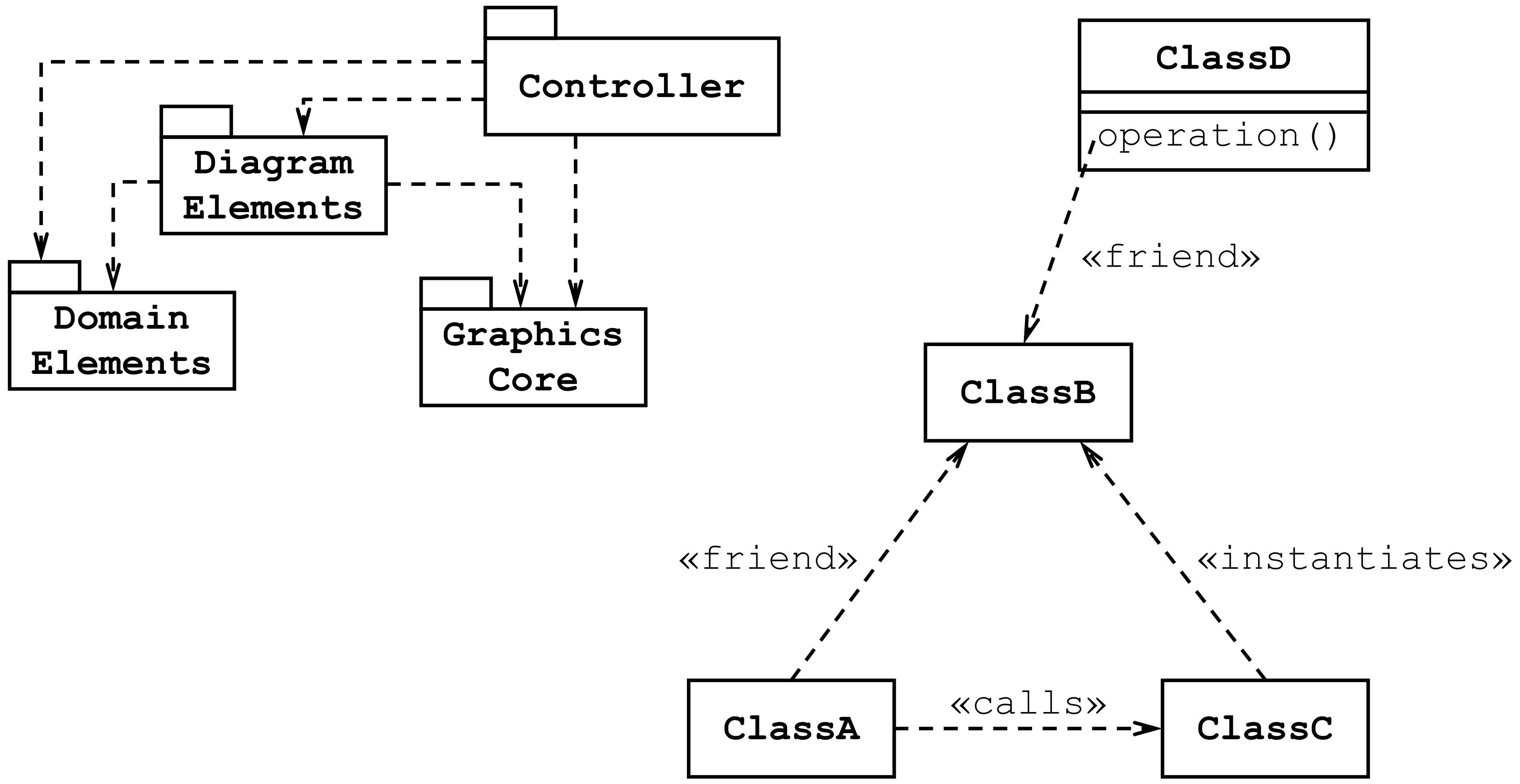
- Dual form of modeling elements: classifier and instance.
- Notation of the instance form uses the same geometrical symbol as the classifier but name is underlined .
- Examples: class-instance specification, association-link, parameter-value, operation-call, ...

## Dependency

- → A relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.
- ■ The complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s).

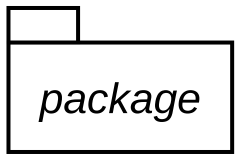


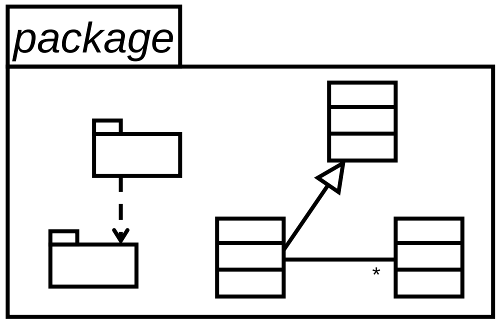
## Examples of Dependencies



## Package

- → A package is used to group elements, and provides a namespace for the grouped elements.
- ■ Only packageable elements can be owned members of a package.
- ■ May contain other packages.
- ■ Package can be used in:
  - Use Case View ⇒ functional decomposition
  - Static Structure View ⇒ logical high-level architecture
  - Component View ⇒ modular decomposition
  - Deployment View ⇒ physical hardware decomposition
- ■ Structure diagrams containing only packages are called ***Package Diagrams***.

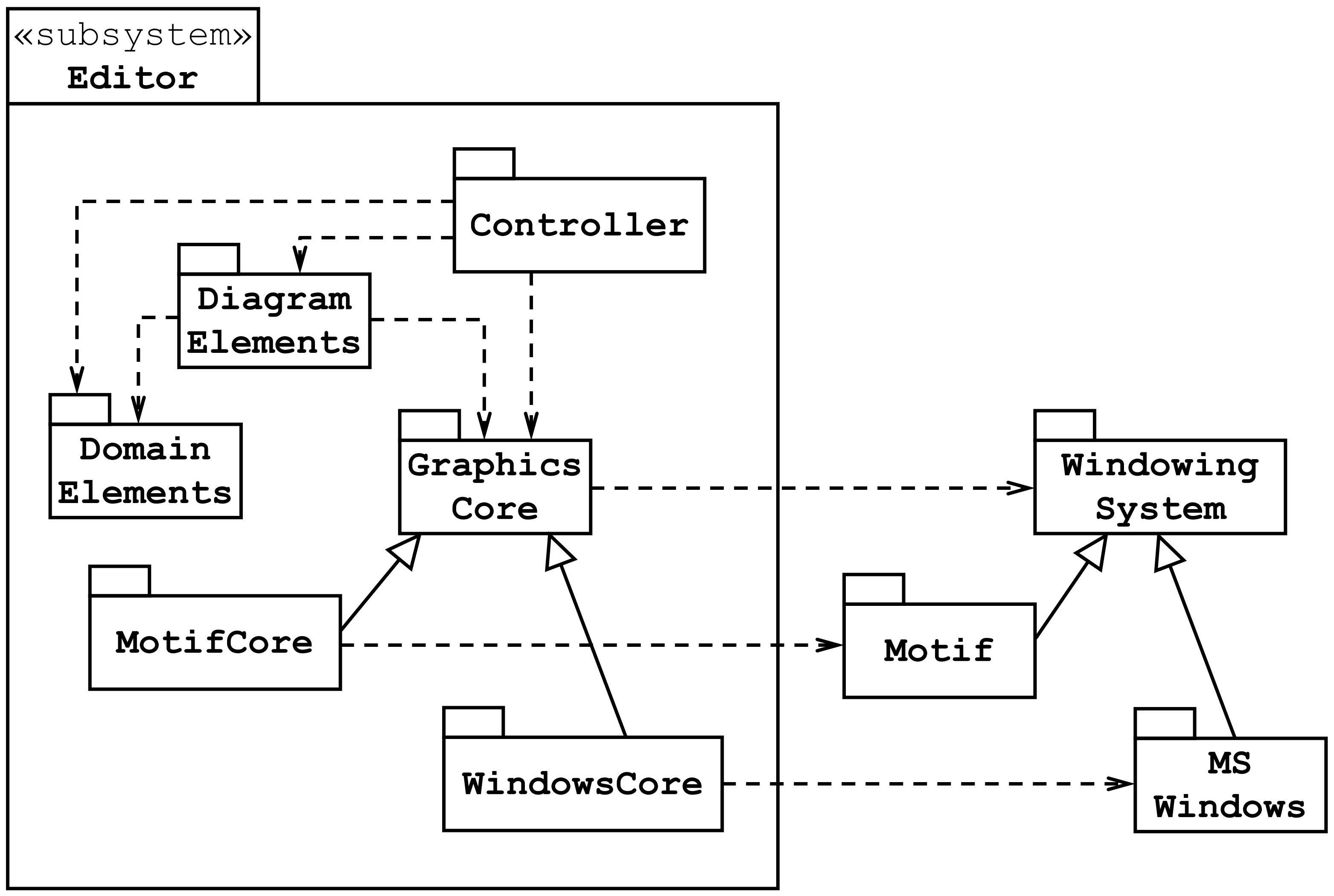




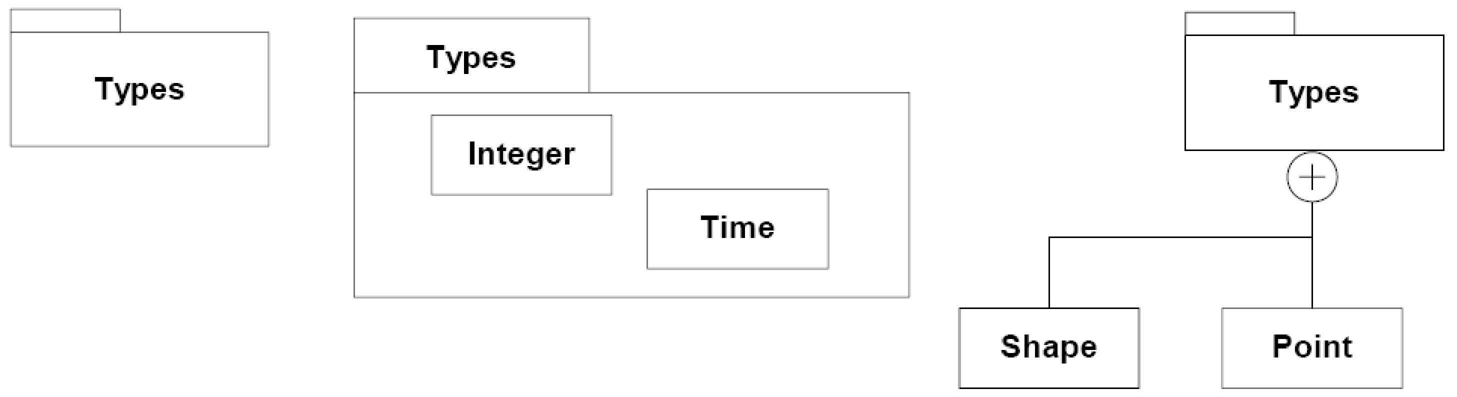
## Standard Types of Packages

- «framework»
  - A package that contains model elements that specify a reusable architecture for all or part of a system. Frameworks typically include classes, patterns, or templates. When frameworks are specialized for an application domain they are sometimes referred to as application frameworks.
- «modelLibrary»
  - A package that contains model elements that are intended to be reused by other packages. A model library is analogous to a class library in some programming languages.

## Example of Package Diagram



## Example Package Composition



## (Packageable) Element and its Visibility

**Packageable Element**

- → A named element that may be owned directly by a package.
- Must always have a visibility.

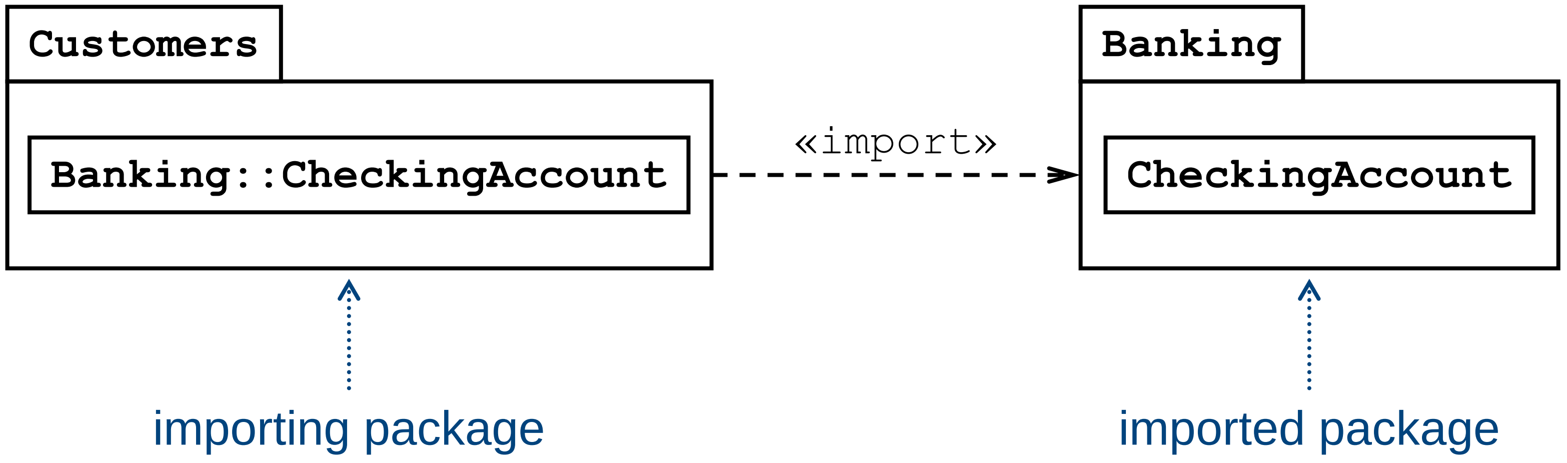
**Visibility Kind**

- Public ‘+’
  - → A public element is visible to all elements that can access the contents of the namespace that owns it.
- Private ‘-’
  - → A private element is only visible inside the namespace that owns it.
- Protected ‘#’
  - → A protected element is visible to elements that have a generalization relationship to the namespace that owns it.
- Package ‘~’
  - → A package element is owned by a namespace that is not a package, and is visible to elements that are in the same package as its owning namespace.

## Package Import

→ A directed relationship that allows the use of unqualified names to refer to package members from other namespaces.

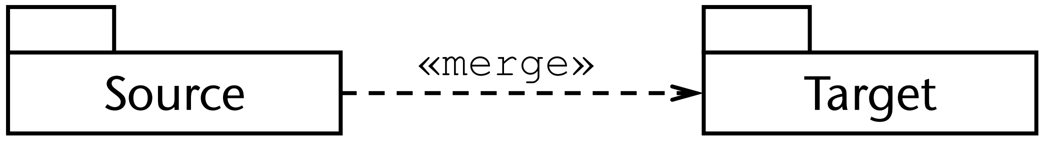
- The import visibility–visibility of the imported packageable elements within the importing namespace.
  - Can only be public (the default value) or private.
  - If the package import is public, the imported elements will be visible outside the package, while if it is private they will not.
- Notation: binary dependency relationship with stereotype «import».
- Full element identification: *package*:: ... ::*package*::*element*



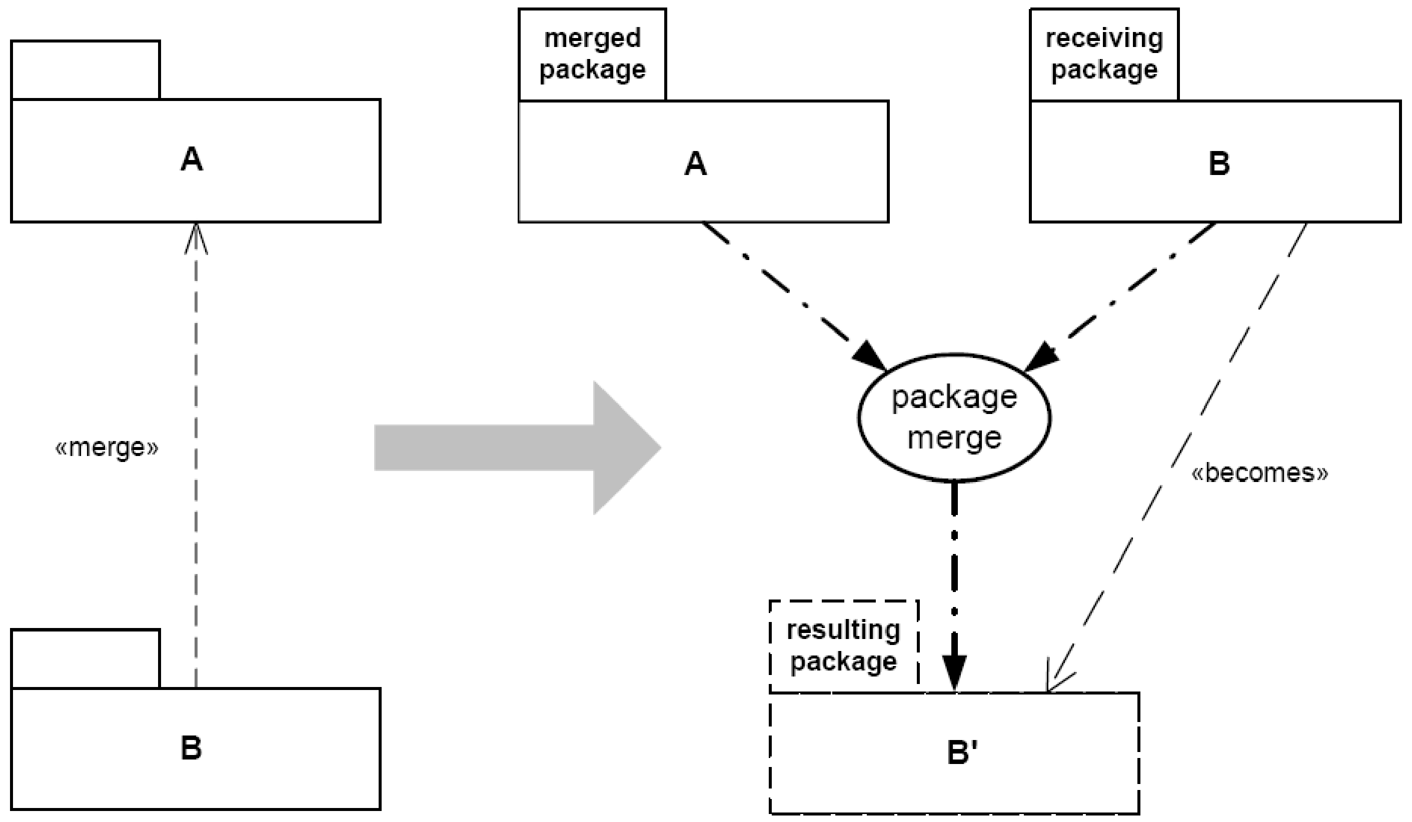
## Package Merge

→ A directed relationship between two packages that indicates that the contents of the two packages are to be combined.

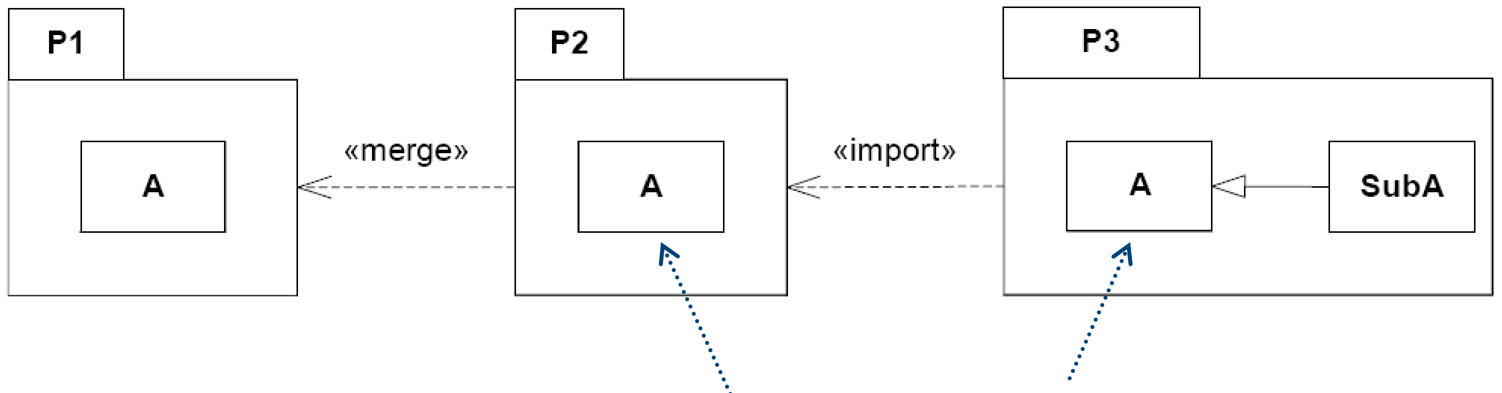
- "Package generalization".
- Extending/specialization of elements with same names in source and target packages.
- Transformation rules and constraints for the contained packages, classes, data types, properties, associations, operations, constraints, enumerations, and enumeration literals.



# Conceptual View of the Package Merge

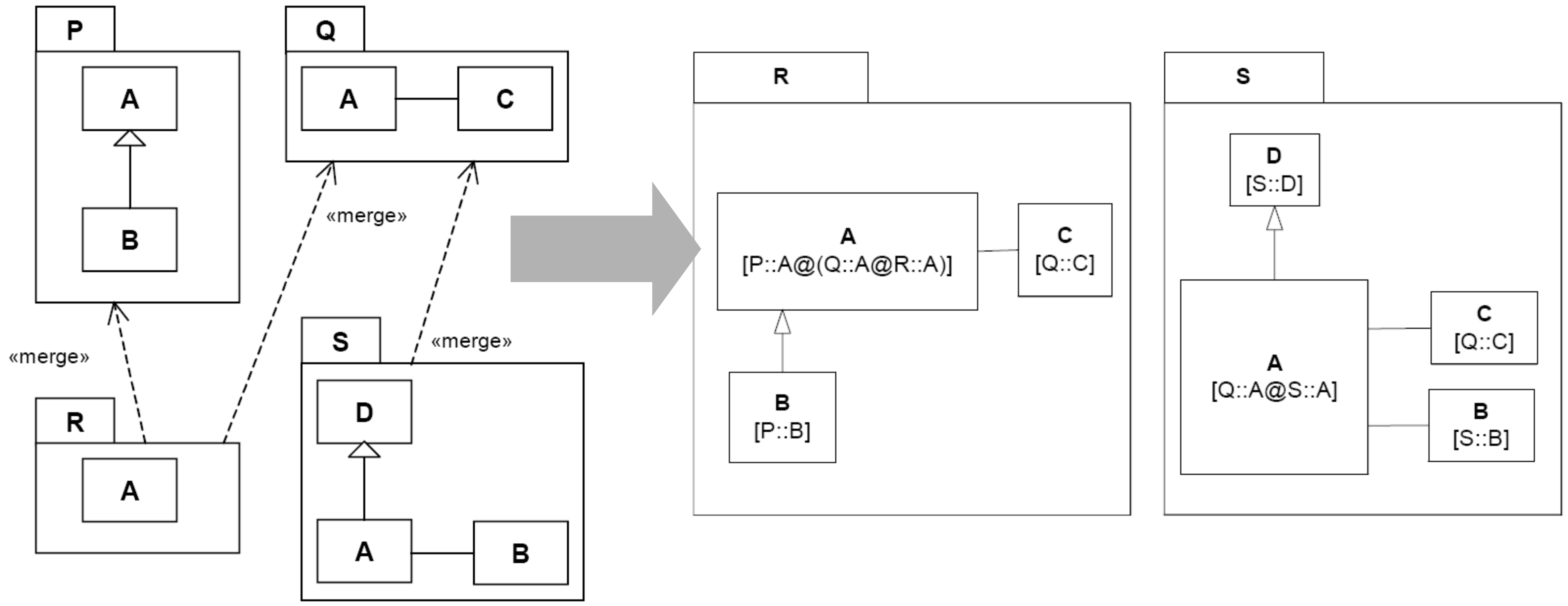


## Examples of Package Merge

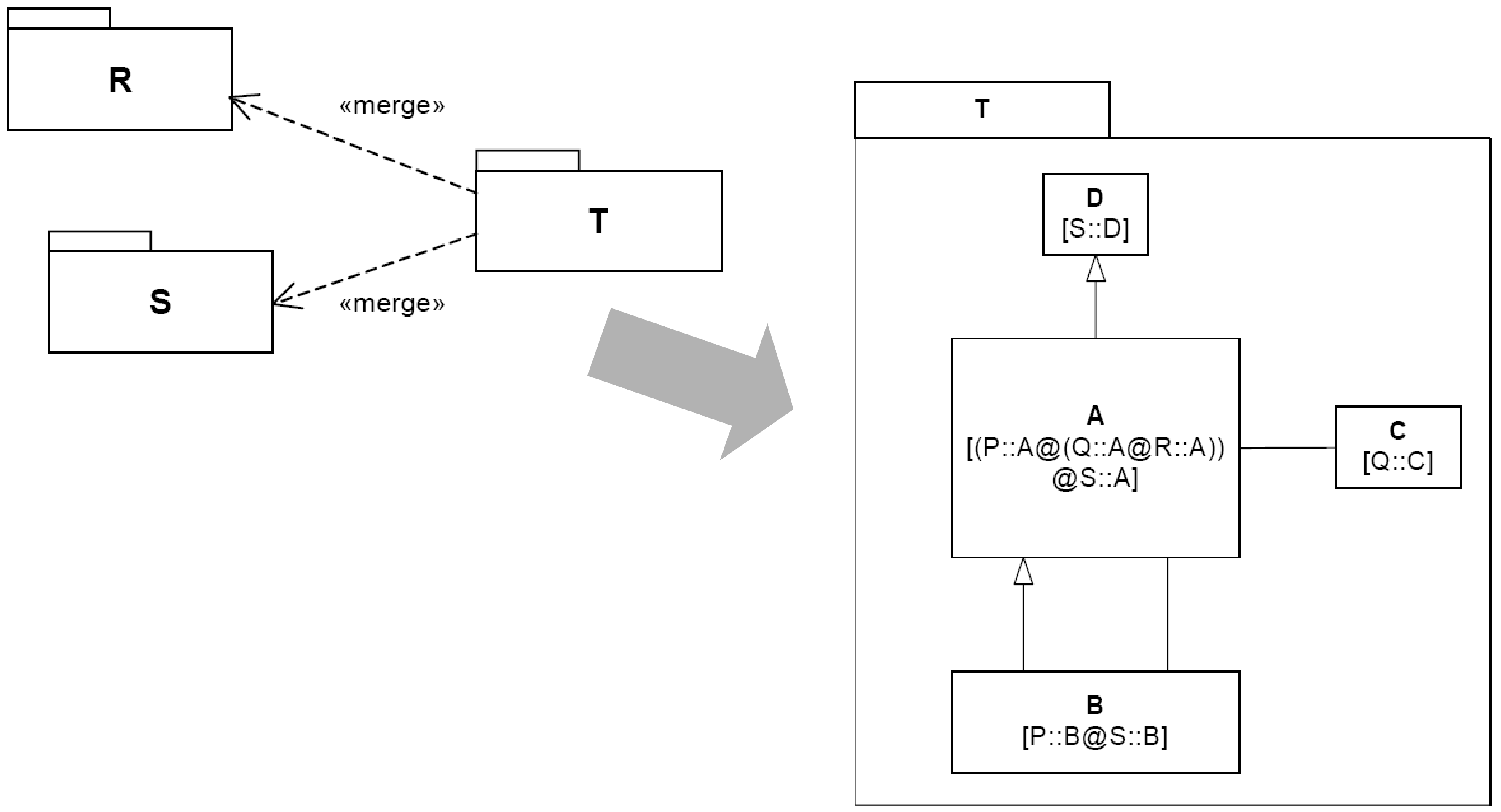


merge of P1::A into P2::A

## Examples of Package Merge (cont.)



## Examples of Package Merge (cont.)


R
«merge»
T
S
«merge»
T
D
[S::D]
A
[(P::A@(Q::A@R::A))
@S::A]
C
[Q::C]
B
[P::B@S::B]