WHITESTEIN Technologies

Unified Modeling Language

Auxiliary Constructs

Radovan Cervenka

Auxiliary Constructs

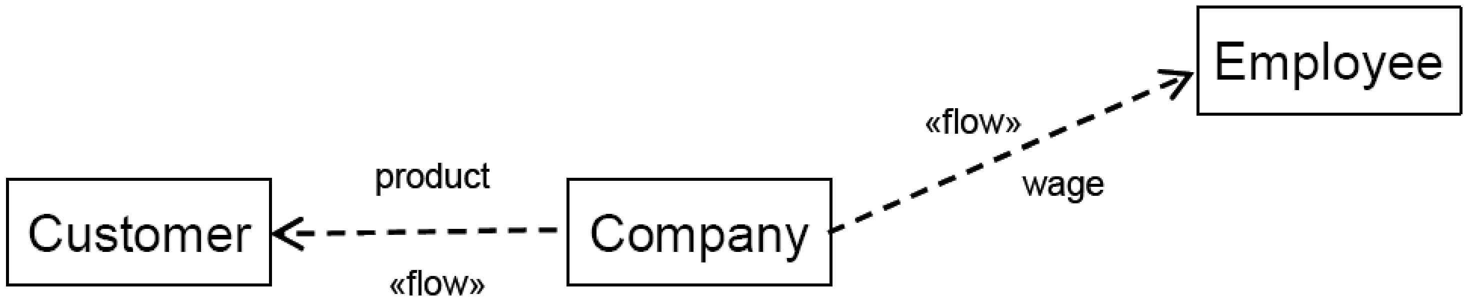
→ **Various auxiliary modeling mechanisms.**

Comprise:

- Information flows.
- Models.
- Primitive types.
- Templates.

Information Flows

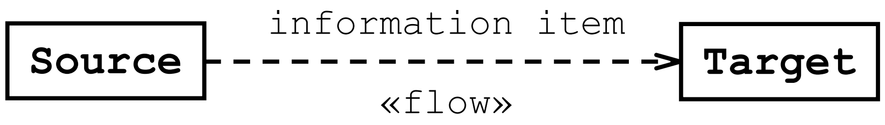
- Provides mechanisms for specifying the exchange of information between entities of a system at a high level of abstraction.
- Do not specify the nature of the information (type, initial value), the mechanisms by which this information is conveyed (message passing, signal, common data store, parameter of operation, etc.), nor sequences or any control conditions.



Information Flow

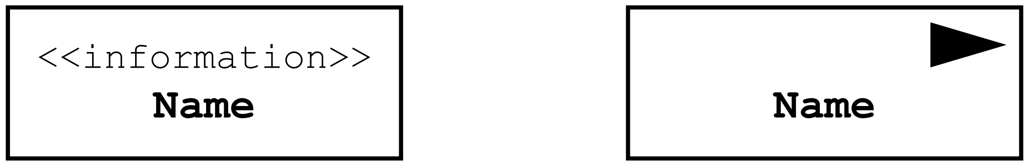
→ Specifies that one or more information items circulates from its sources to its targets.

- It is used to ***abstract*** the communication.
- Requires some kind of “information channel” for transmitting information items.
  - Connectors, links, associations, or even dependencies.
- When a source or a target is:
  - A *classifier*, the information flow represents *all the potential instances* of the classifier.
  - A *part*, the information flow represents *all instances* that can play the role specified by the part.
  - A *package*, the information flow represents all potential *instances* of the directly or indirectly *owned classifiers of the package*.



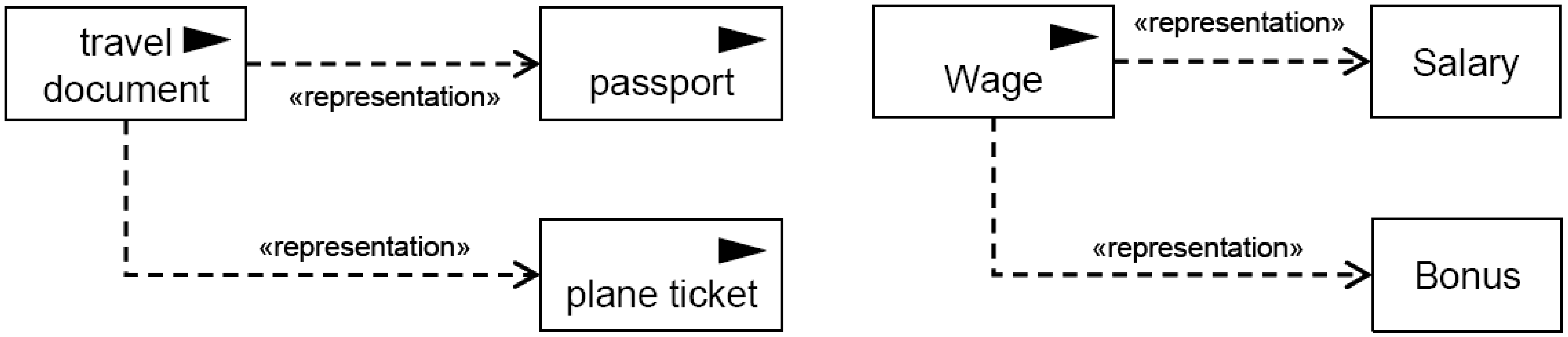
Information Item

- → An abstraction of all kinds of information that can be exchanged between objects.
- A kind of *classifier* intended for representing information at a very abstract way, one which *cannot be instantiated*.
- Used to define preliminary models, before having made detailed modeling decisions on types or structures.
- Encompasses all sorts of data, events, facts that are exploited inside the modeled system.
- An information item does not specify the structure (properties or associations), the type, or the nature of the represented information.
- Information items can be decomposed into more specific information items using ***representation links*** between them.

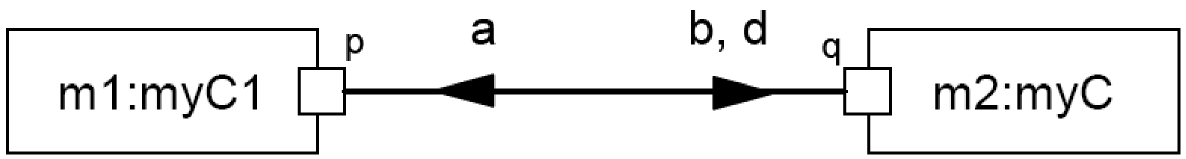


Examples of Information Flows

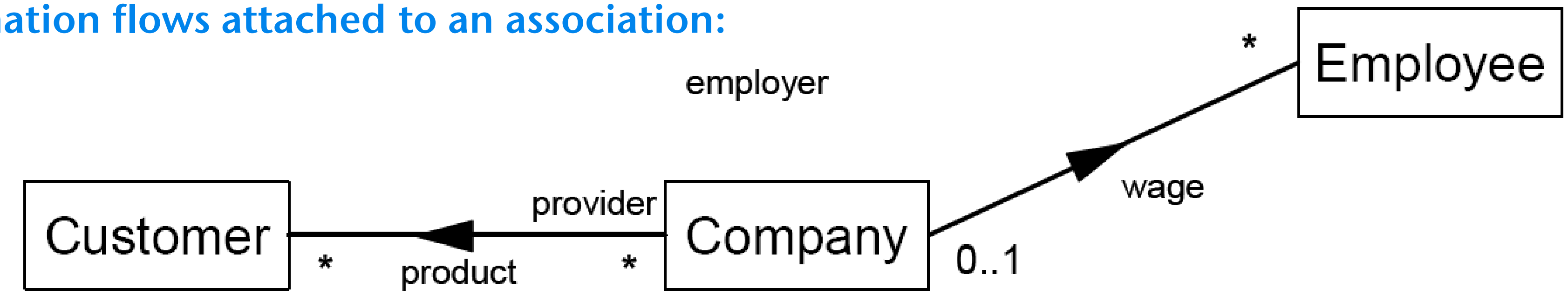
Definition of information items:



Information flows attached to a connector:

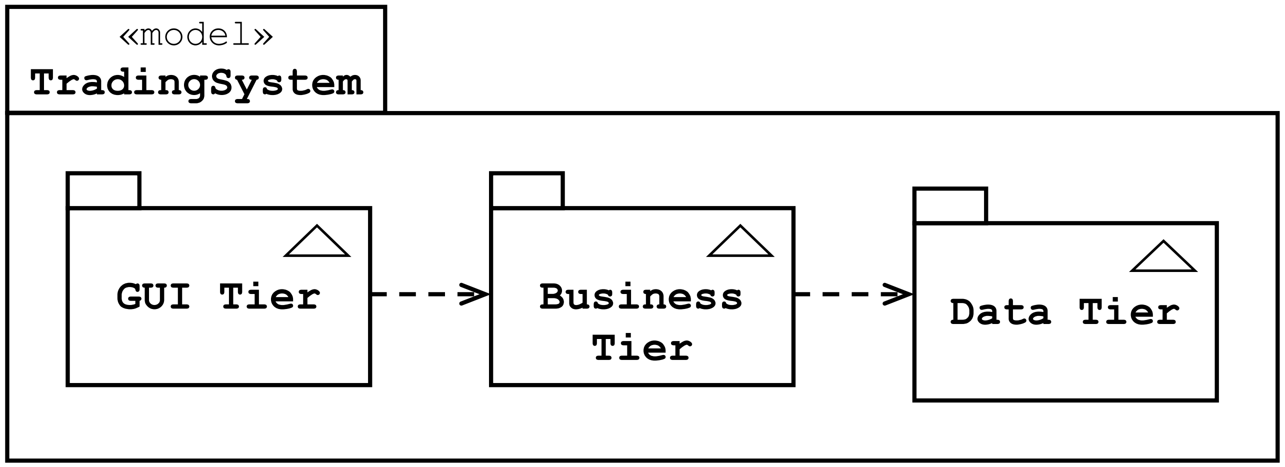


Information flows attached to an association:



Models

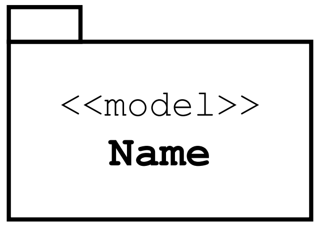
- Allows to explicitly represent models.

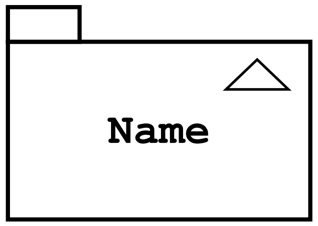


Model

→ Used to capture a view of a physical system.

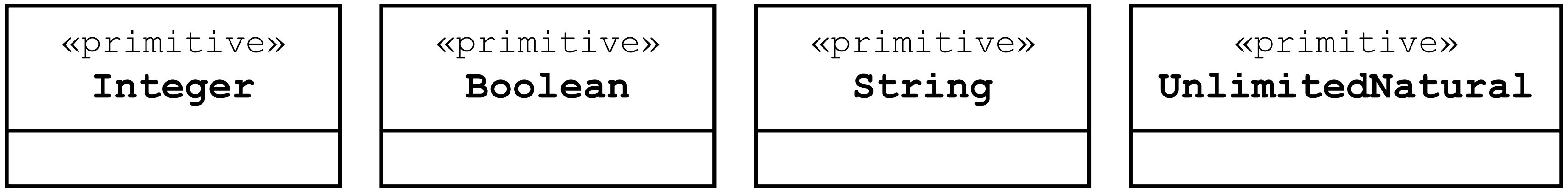
- It is an *abstraction* of the physical system, with a certain *purpose*.
  - This purpose determines what is to be included in the model and what is irrelevant. It also determines the readers of the model.
  - Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail.
- Examples of usages: logical model, behavioral model, deployment model, ...
- Models are “self-contained”.
- A specialized package.





Primitive Types

- A set of primitive data types used in the UML metamodel.
- These types are reused by both MOF and UML, and may potentially be reused also in user models.
- Tool vendors, however, typically provide their own libraries of data types to be used when modeling with UML.

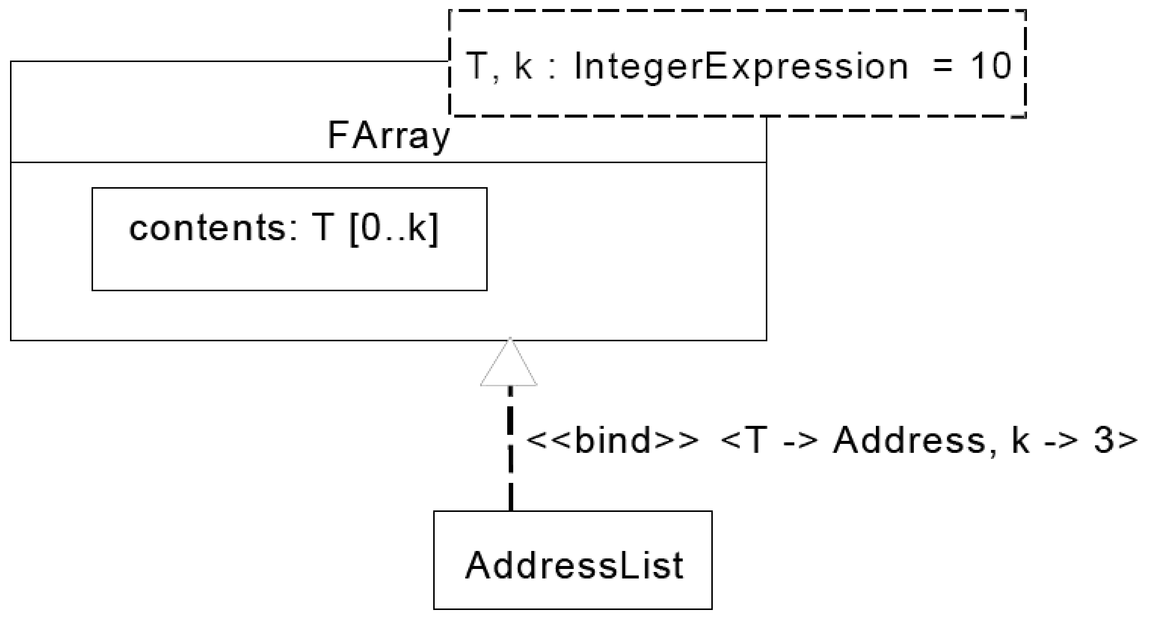


Primitive Types (cont.)

- *Integer*
  - A primitive type representing integer values.
- *Boolean*
  - Used for logical expressions. It has the predefined literals *true* and *false*.
- *String*
  - A sequence of characters in some character set. Defines a piece of text.
- *Unlimited natural*
  - An instance of unlimited natural is an element in the (infinite) set of naturals (0, 1, 2...). The value of infinity is shown using an asterisk (‘*’).
  - Appears as the type of upper bounds of multiplicities in the metamodel.

Templates

→ The mechanism for defining reusable “model patterns” (based on classifiers and packages) and binding them to concrete parts of the model.


T, k : IntegerExpression = 10
FArray
contents: T [0..k]
<<bind>> <T -> Address, k -> 3>
AddressList

Template Definition

→ Definition of a *template signature* for a *templateable elemement*.

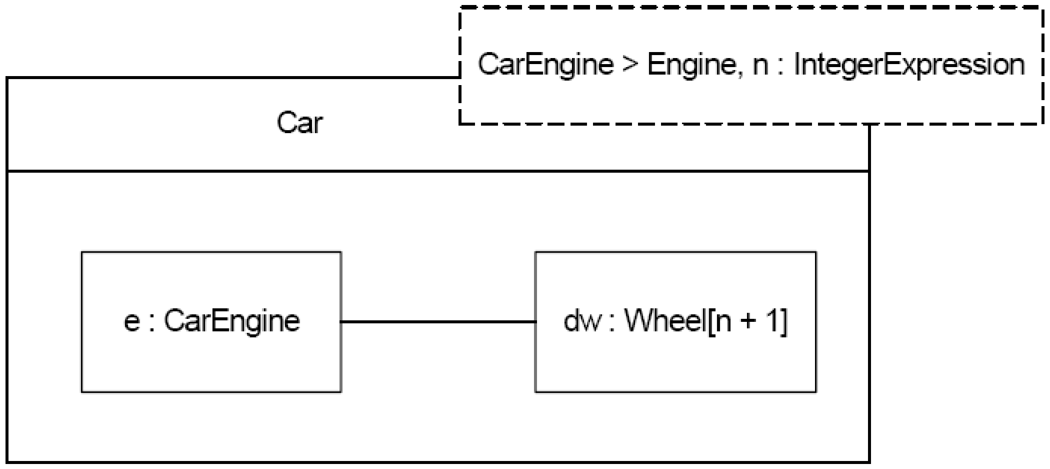
- ***Templateable element***
  - An element which can contain a template signature; therefore is often referred to as a ***template***.
  - It can be either Class, Package or Operation.
- ***Template signature***
  - Specifies a set of formal template parameters for a templated element.
  - Is owned by a templateable element and has one or more template parameters.
- ***Template parameter***
  - References a parameterable element that is exposed as a formal template parameter in the containing template.
  - Format:

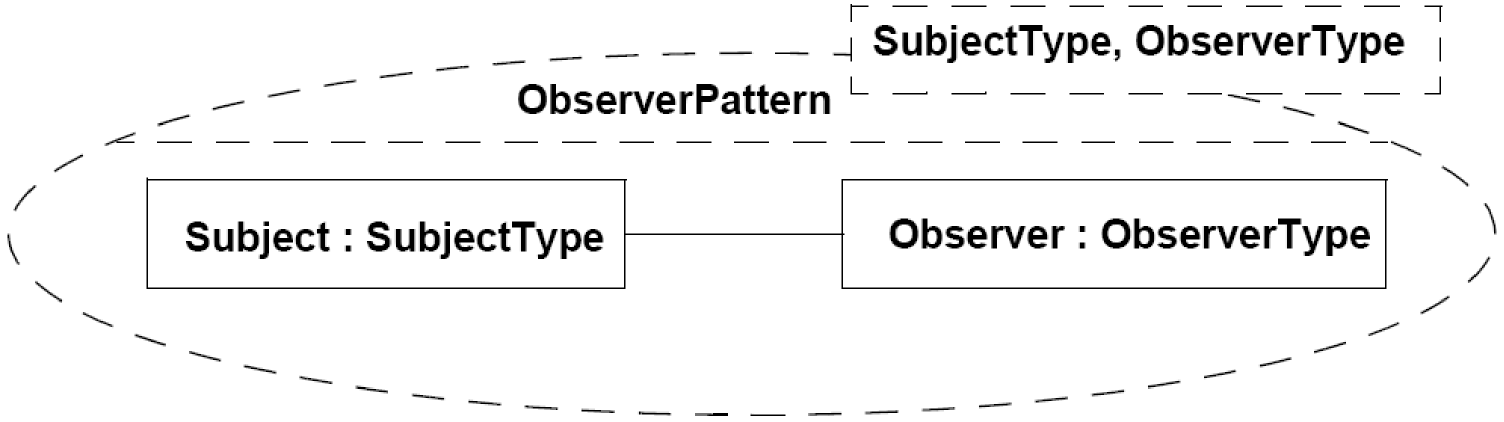
    template-parameter ::= *template-param-name* [‘:’ *parameter-kind*]
    [‘=’ *default*]

Template Definition (cont.)

- *Parameterable element*
  - An element that can be exposed as a formal template parameter for a template, or specified as an actual parameter in a binding of a template.
  - Informally speaking, it can be any sub-element of templatable element.
- *Named element (from Templates)*
  - A named element is extended to support using a *string expression* to specify its name.
  - Used to allow names of model elements to involve template parameters.
  - When a template is bound, the sub expressions are substituted with the actual values substituted for the template parameters to obtain the actual bound element name.
  - Format:
    - The string expression appears between “$” signs.
    - Template parameters in are enclosed in ‘<‘ and ‘>’ signs.

Examples of Templates





Template Binding

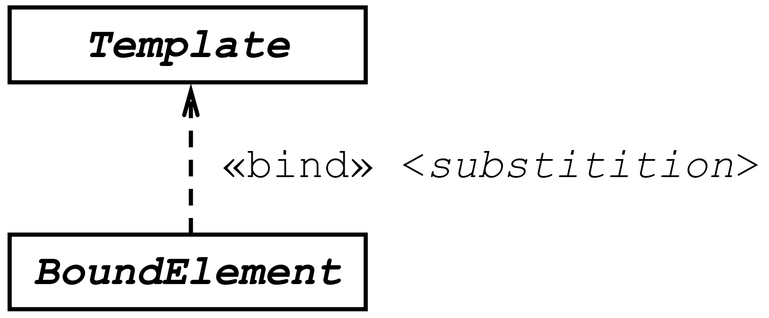
→ Represents a relationship between a templateable element and a template. A template binding specifies the ***substitutions*** of actual parameters for the formal parameters of the template.

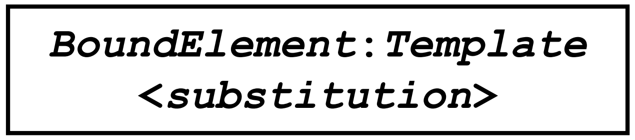
- Format :

  element-name ':' *binding-expression* [',' *binding-expression*]*

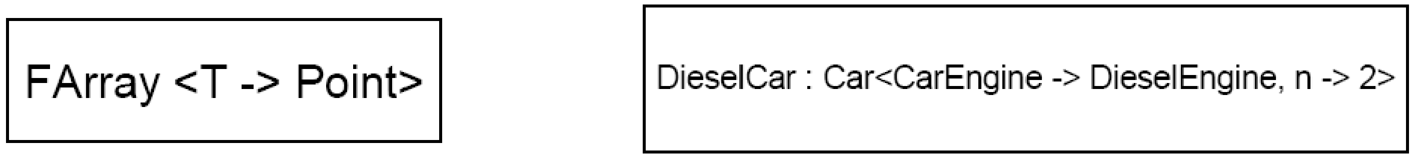
  binding-expression ::= *template-element-name* '<'
  template-parameter-substitution [',' *template-parameter-substitution*]* '>'

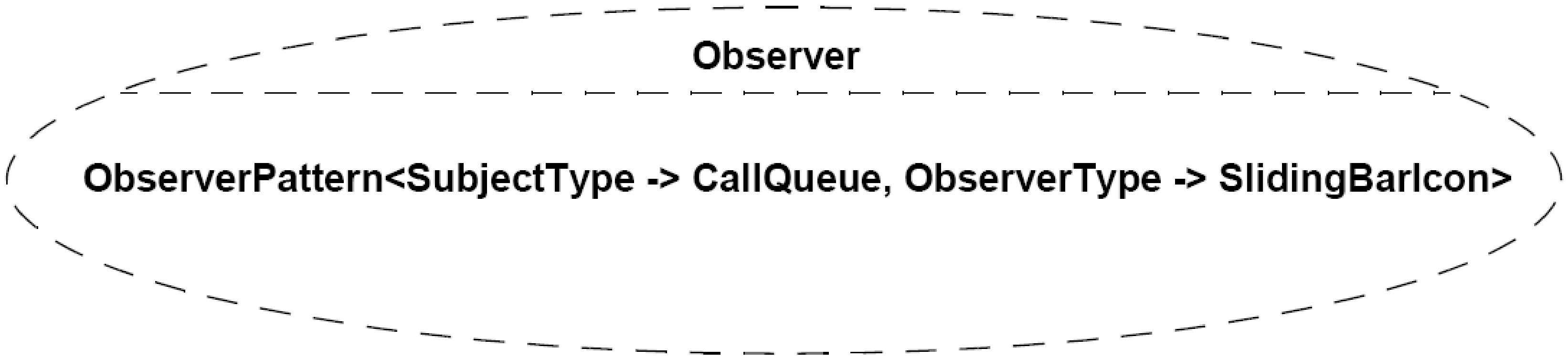
  template-param-substitition ::= *template-param-name* '->' *actual-template-parameter*



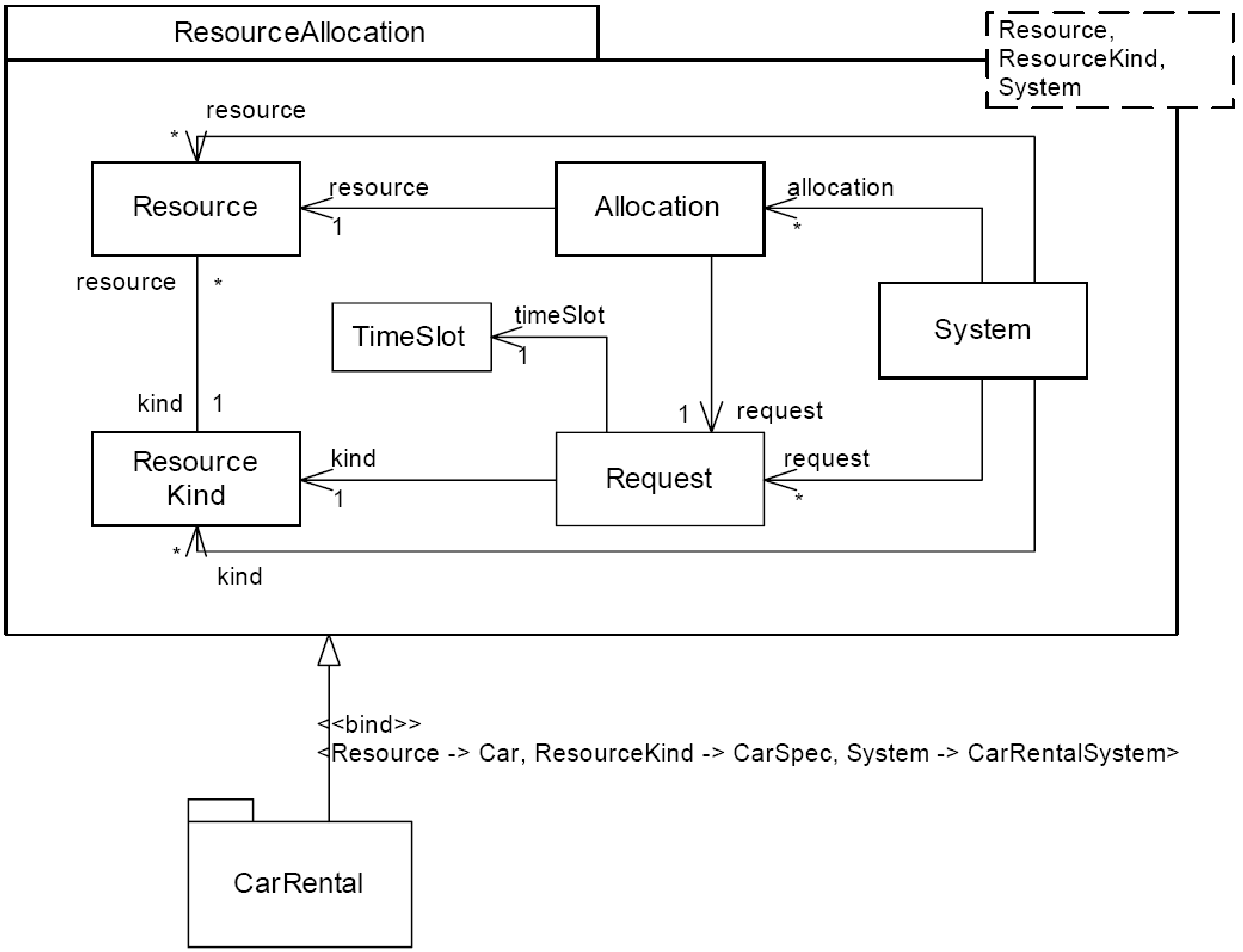


Examples of Template Bindings (1)

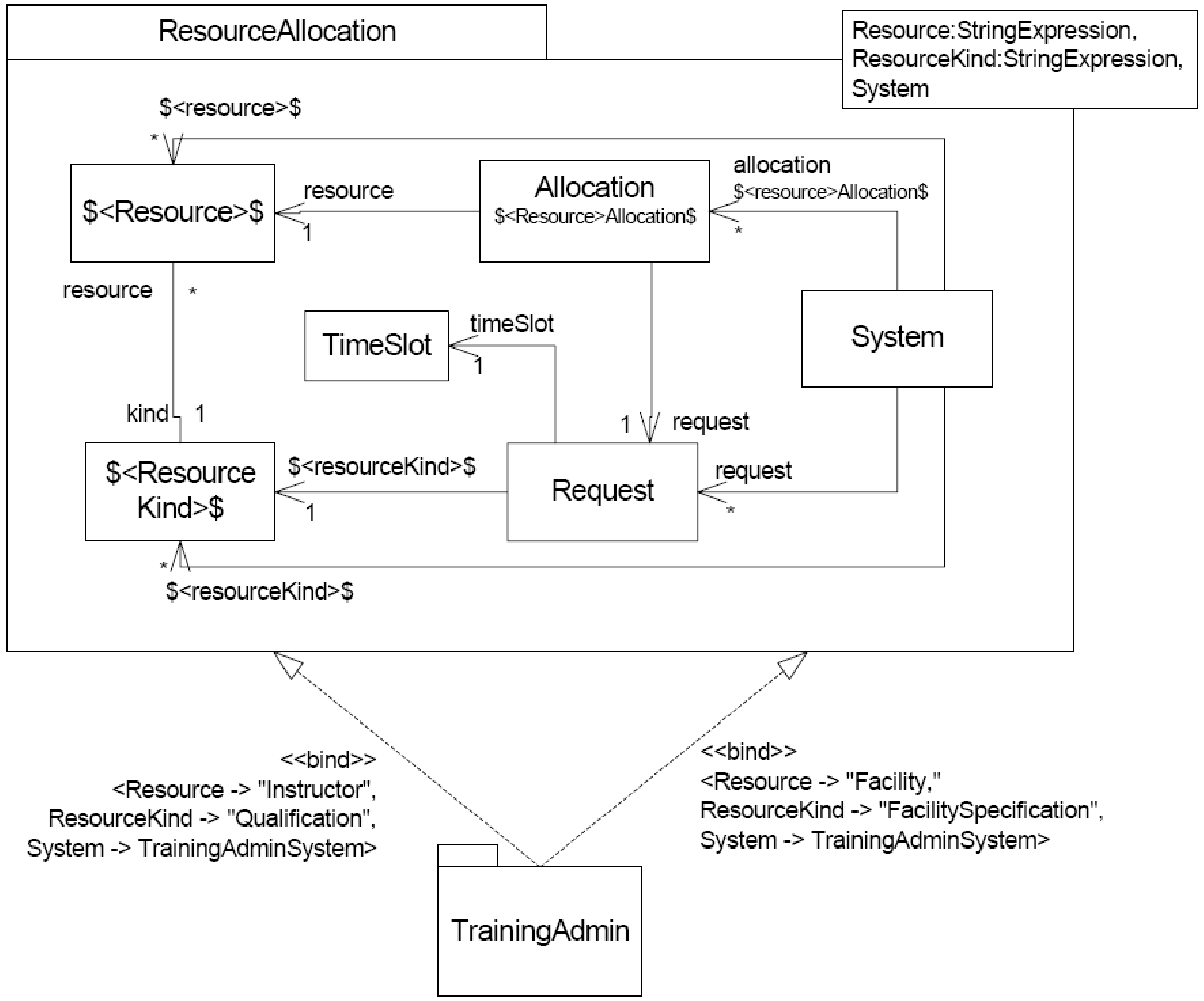



Observer
ObserverPattern<SubjectType -> CallQueue, ObserverType -> SlidingBarIcon>

Examples of Template Bindings (2)



Examples of Template Bindings (3)



ResourceAllocation
Resource:StringExpression,
ResourceKind:StringExpression,
System
$<resource>$
*
$<Resource>$
resource
1
Allocation
$<Resource>Allocation$
allocation
$<resource>Allocation$
*
resource
*
TimeSlot
timeSlot
1
System
kind
1
1
request
$<Resource
Kind>$
$<resourceKind>$
1
Request
request
*
*
$<resourceKind>$
<<bind>>
<Resource -> "Instructor",
ResourceKind -> "Qualification",
System -> TrainingAdminSystem>
<<bind>>
<Resource -> "Facility,"
ResourceKind -> "FacilitySpecification",
System -> TrainingAdminSystem>
TrainingAdmin