NET.OBJECTDAYS 2005

Aspect Oriented Programming with Views and Collaborations

The TOPPrax approach

Stephan Herrmann
Christine Hundt
Technische Universität Berlin

stephan@cs.tu-berlin.de
resix@cs.tu-berlin.de

www.ObjectTeams.org

Language & Method

PART 1:

ObjectTeams/Java – The Language

PART 2:

Patterns of Good Design with OT/J

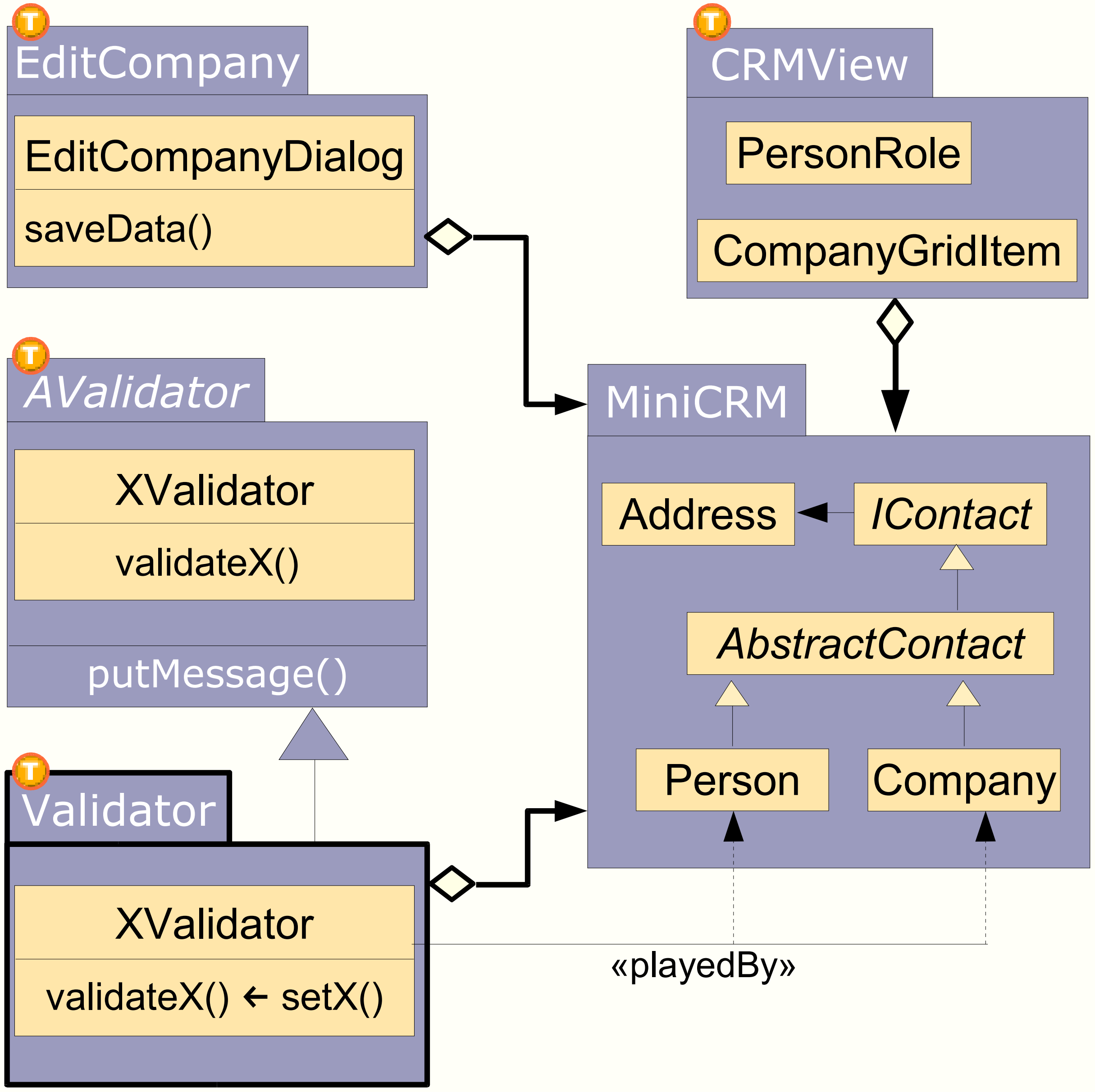
Outline Part 2

- **Patterns of good design with OT/J**

  Patterns found in existing applications:
  - Connector
  - Notification
  - Virtual Association
  - Virtual Restructuring
  - Variant

  Scalable Designs:
  - Nesting, stacking and layering of Teams.

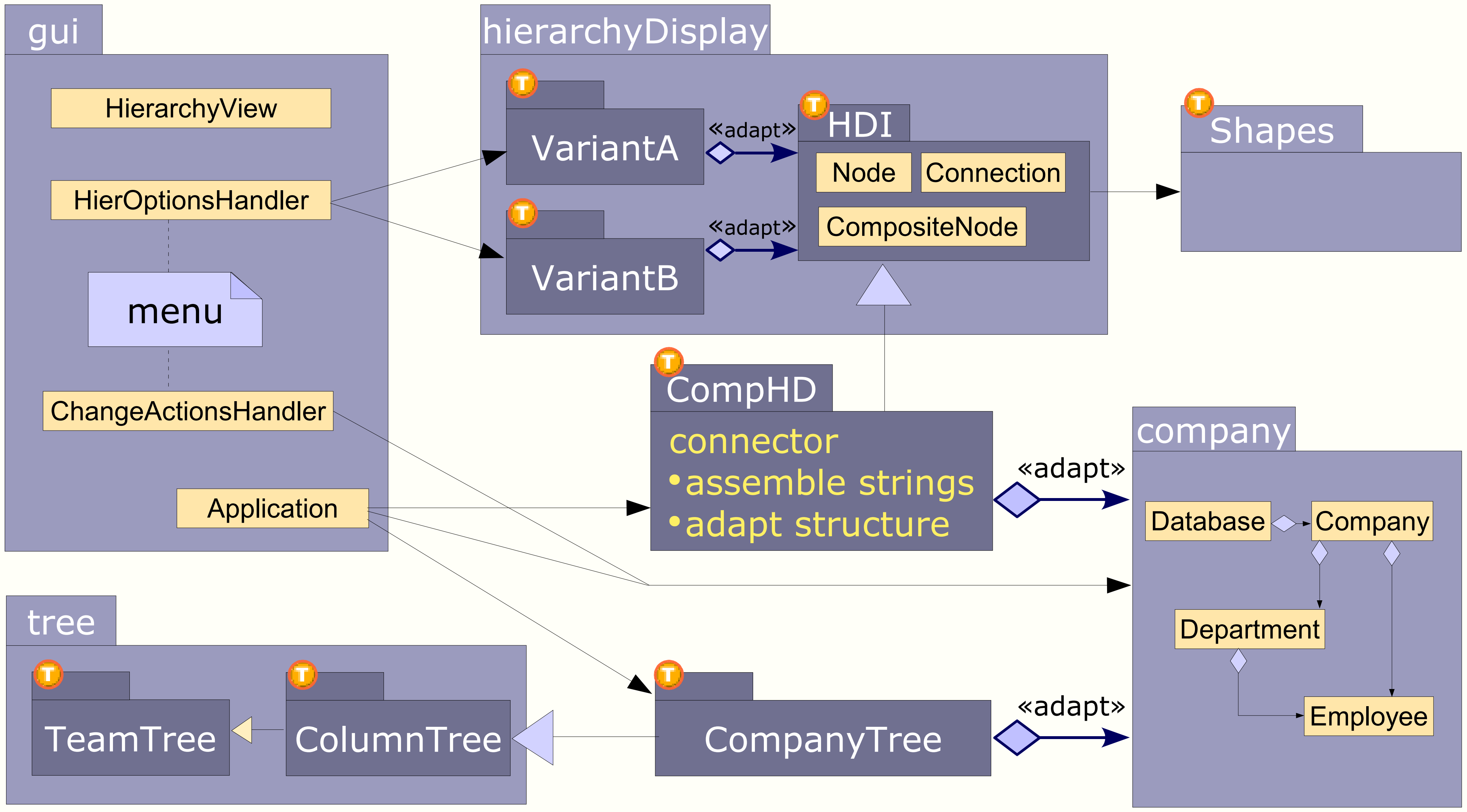
miniCRM explained



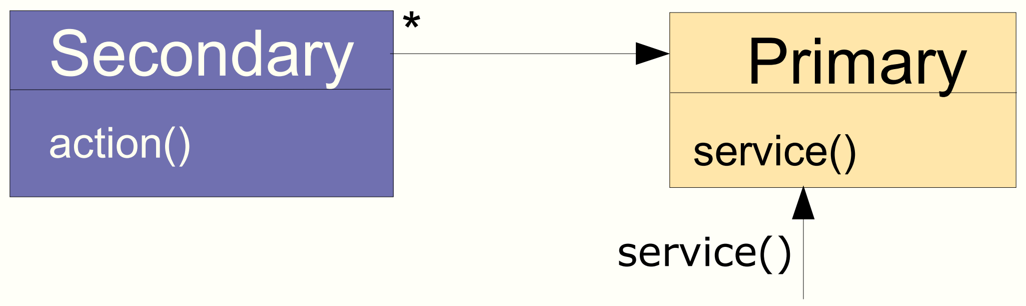
Connector

- extend a team
- bind roles-bases
- bind methods
- activate at launch time

Company Hierarchy



Notification

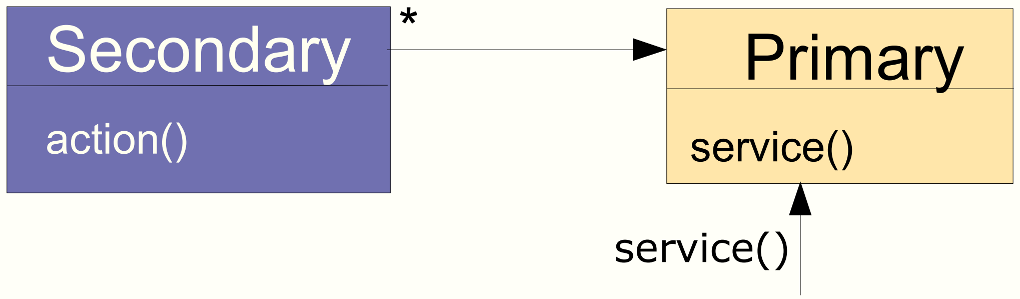


Motivation

- Some client invokes a method on Primary
- Secondary wants to be notified
- Primary does not know about Secondary
- There may be many Secondaries

(let's for a minute forget about Observer)

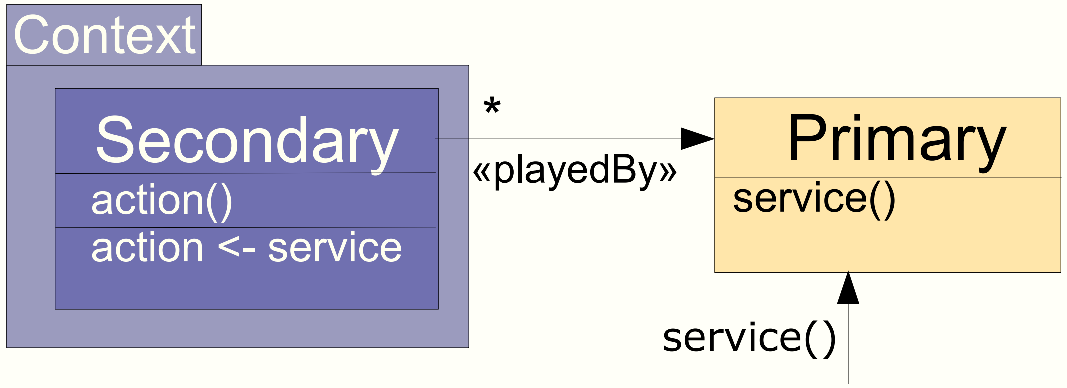
Notification



Applicability

- Secondary is explicitly **associated** to Primary
- It is known when Secondary's interest **starts/stops**
- Primary shall be independent of Secondary

Notification

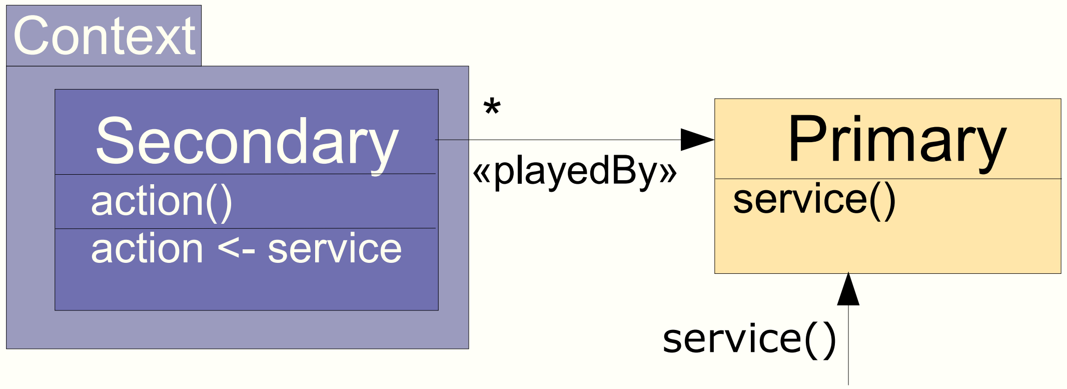


OT/J solution

- Secondary is a **role** of Primary within some Context
- Notification is implemented by a **callin binding**

how is start/stop of the protocol realized?
⇒ **Variants**

Notification



Variants: start/stop

1. Secondary is already a role of Primary
   Period of interest = full life-time of Secondary

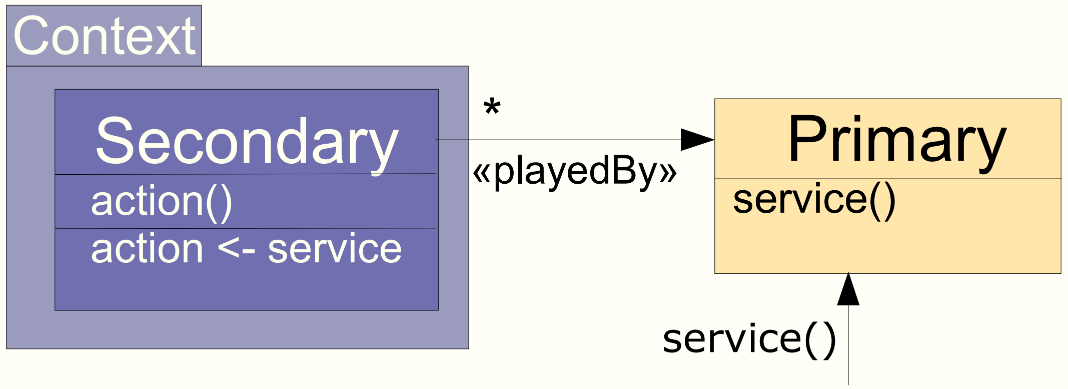
2. Secondary's interest is registered explicitly
   Need to force role creation by lifting (+unregisterRole()):

```
Context.register(Primary as Secondary obj) { }
ctx.register(aPrimary);
```

3. Interest is disabled for certain intervals
   Use activation/deactivation of Context:

```
ctx.deactivate();
```

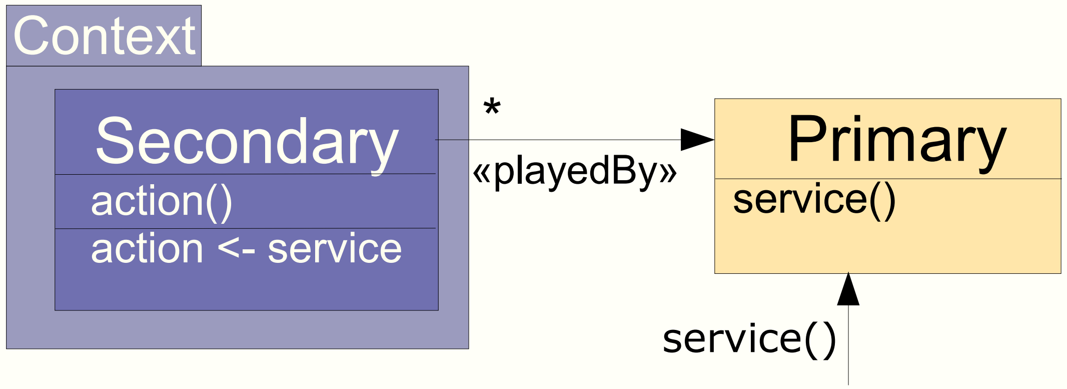
Notification



Known Uses

- Stopwatch
- MiniCRM
- (Company)Hierarchy
- *any OT/J application using MVC*

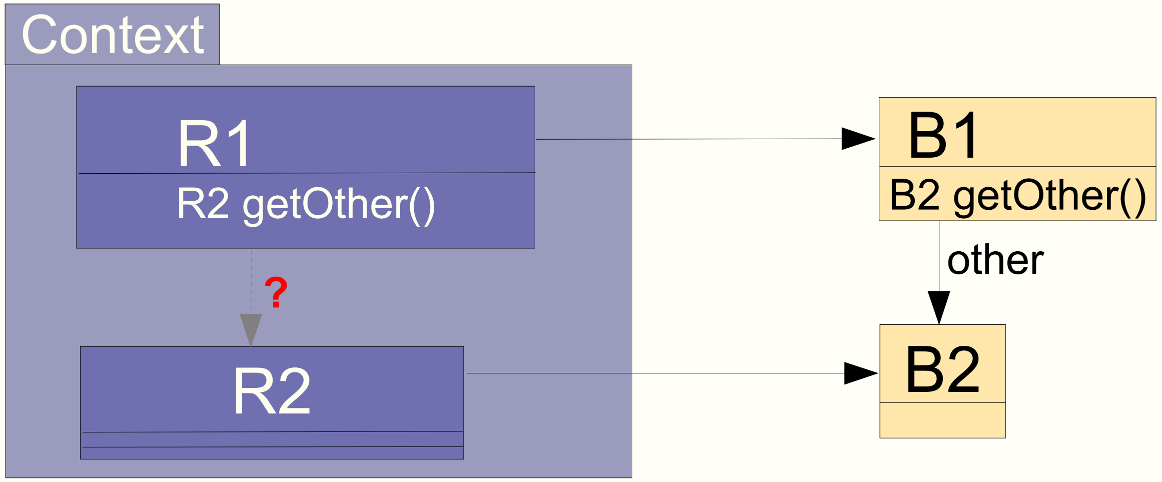
Notification



Known Uses

- Stopwatch
- MiniCRM
- (Company)Hierarchy
- *any OT/J application using MVC*

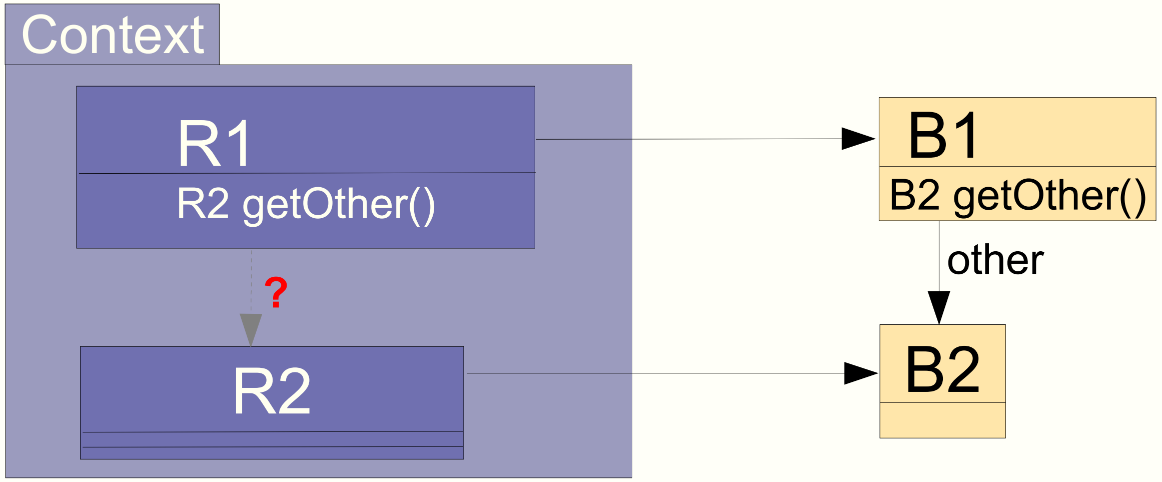
Virtual Association



Motivation

- B1 has an association to B2 (*other*)
- in another context these objects are seen as R1 and R2
- the reference shall not be duplicated
- in the new context it shall be possible to get the associated R2 of an R1

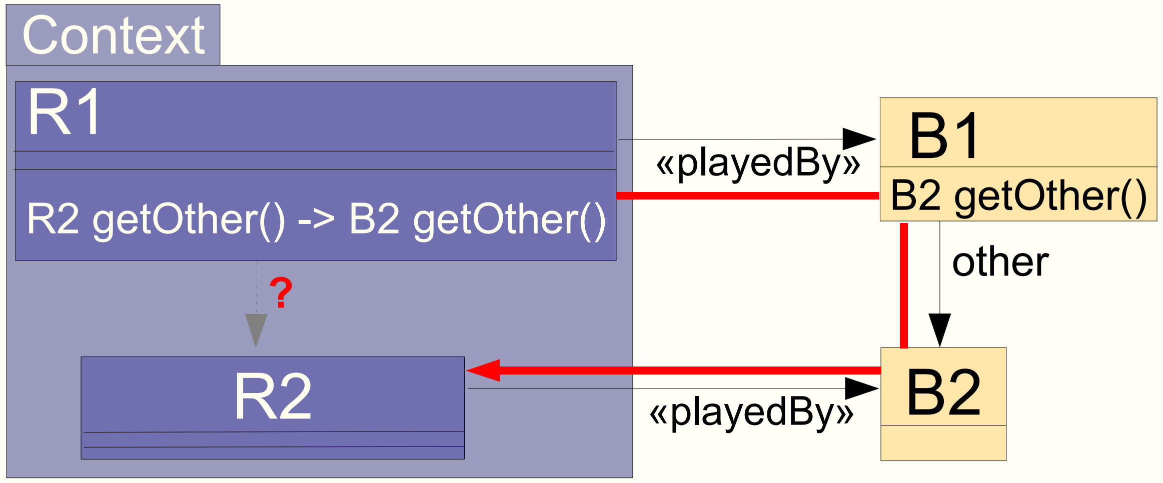
Virtual Association



Applicability

- collaboration with references between some objects
- in another context these objects have to collaborate as well

Virtual Association

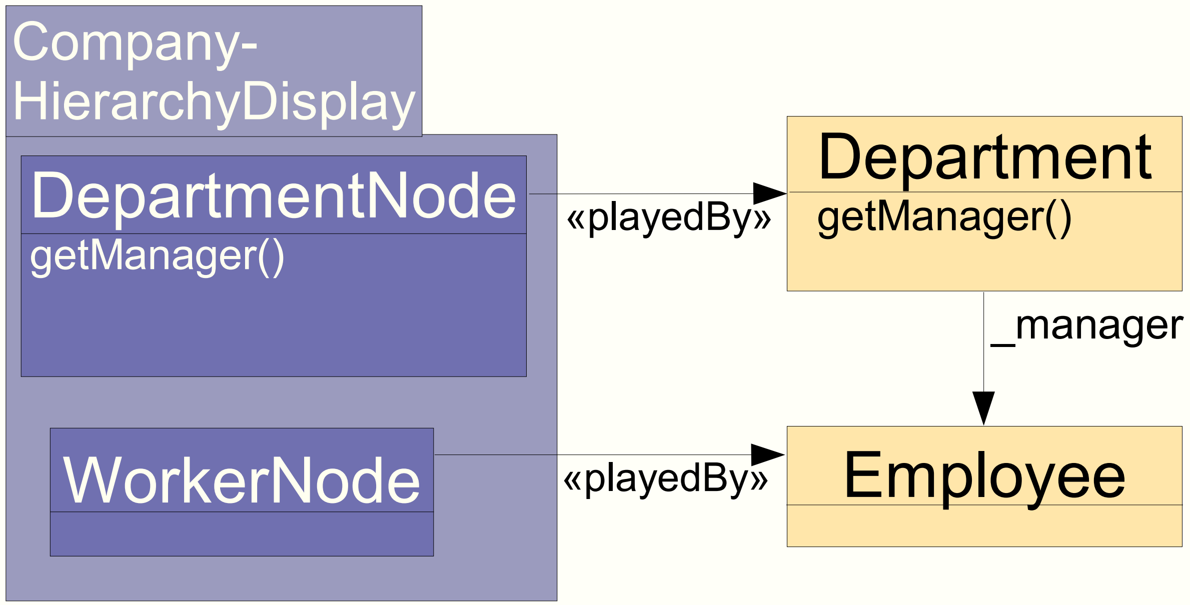


OT/J solution

- R1 is a **role** of B1, R2 is a **role** of B2 within some Context
- the association in virtually accessed by a **callout binding**
- the resulting B2 object is automatically **lifted** to the corresponding Role of type R2

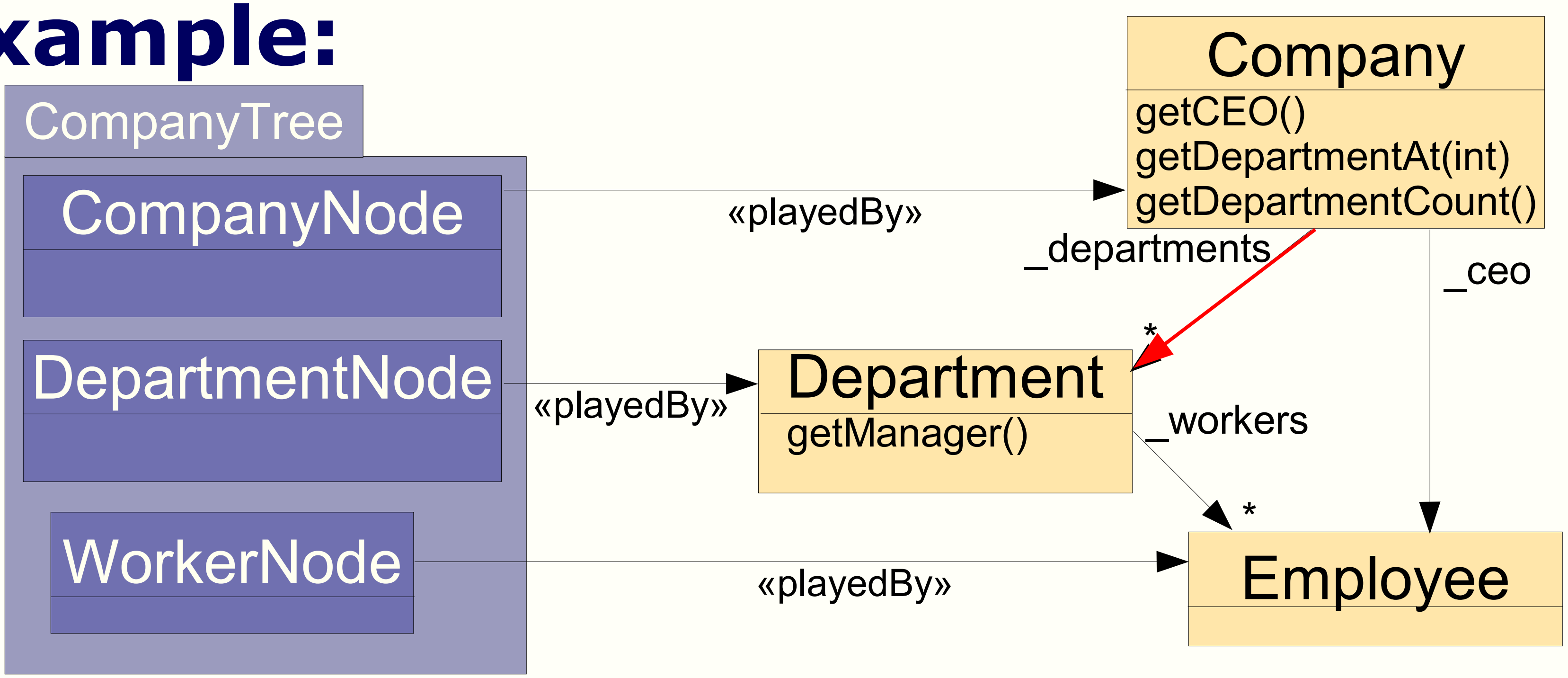
Virtual Association

Example:



Accessing structured types

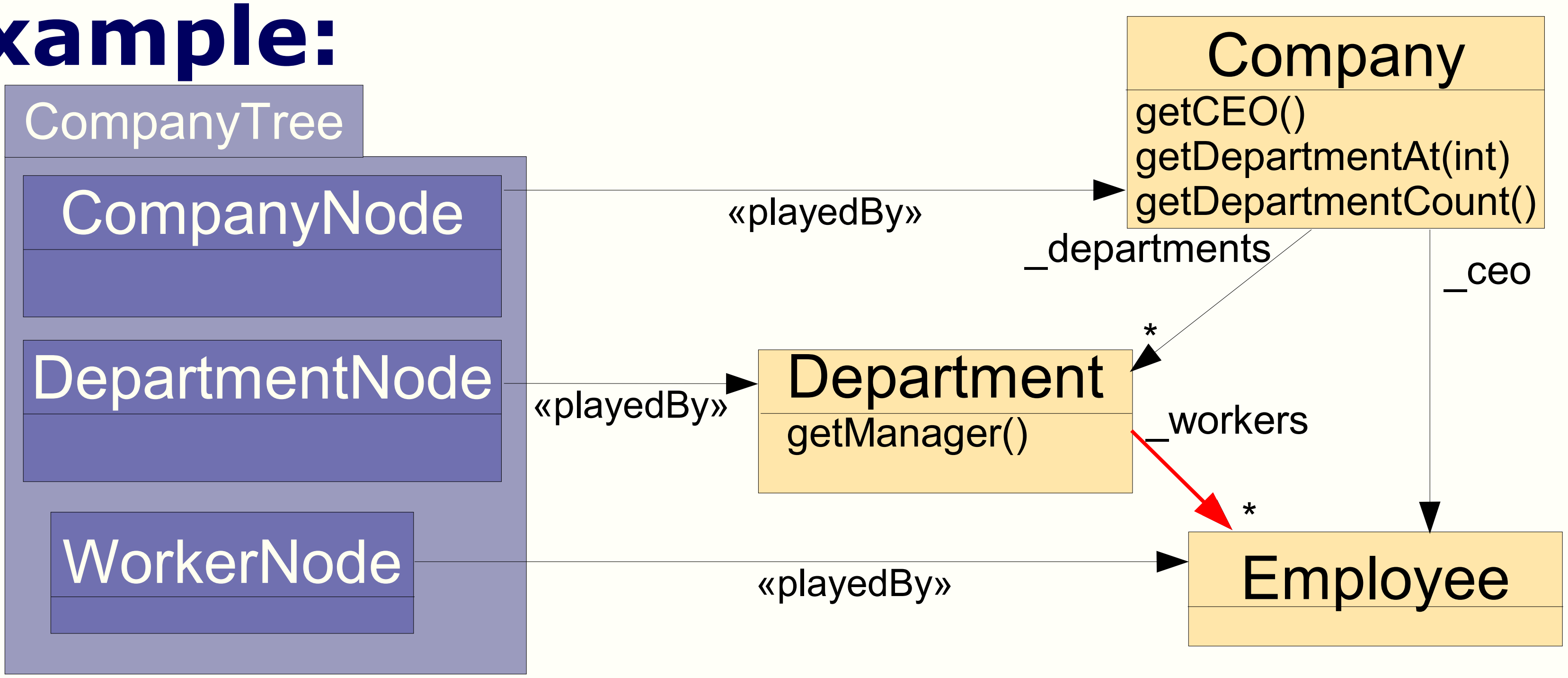
Example:



- **Collections**
  - access methods provided by the interface

Accessing structured types

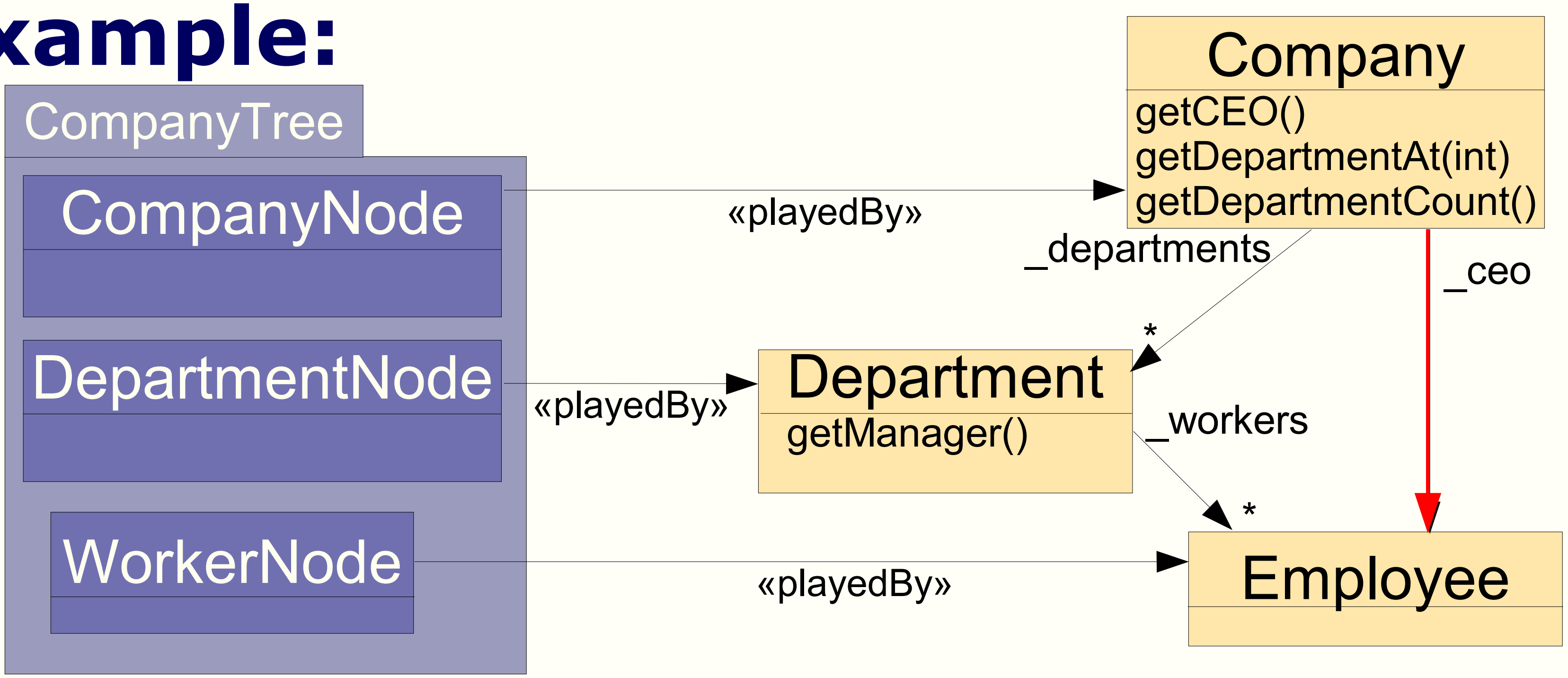
Example:



- **Collections**
  - access methods provided by the interface
  - what if no `getXXCount()` and `getXXAt(int)` are available? (Refactoring „encapsulate collection“)

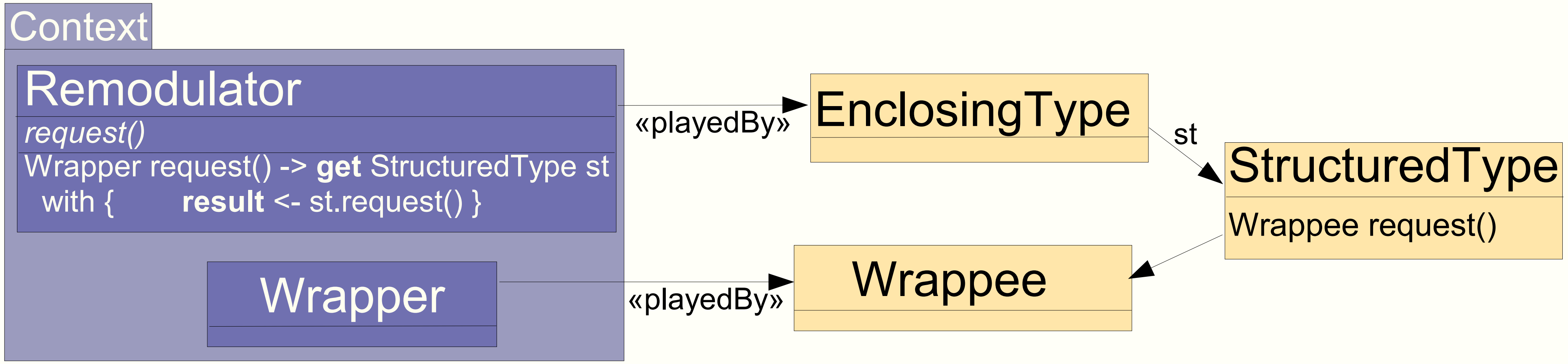
Accessing structured types

Example:



- **Other structured types**
  - inlining fields instead of mapping the complete structure

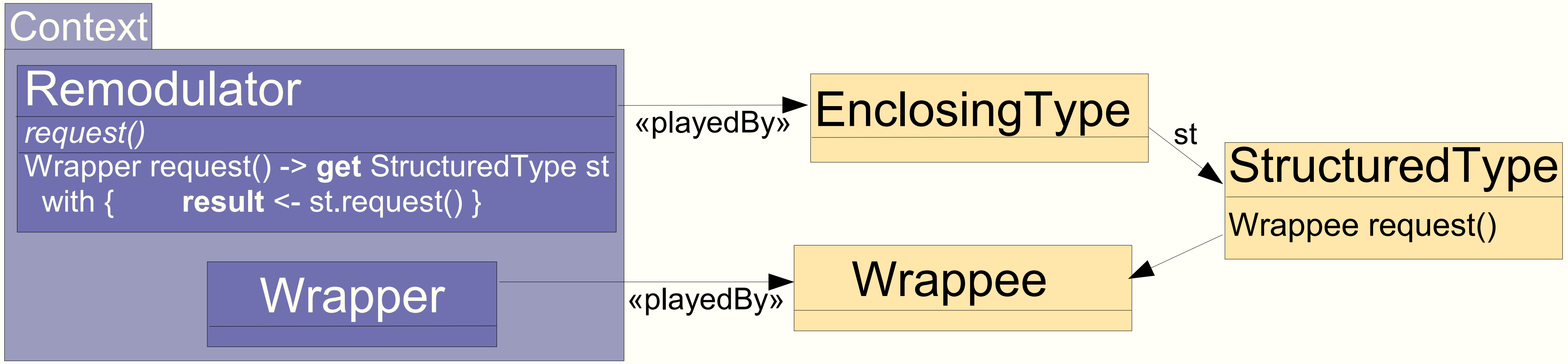
Virtual Restructuring



Motivation

- an object does not provide methods a client wants to use
- from its structure it is possible to get the needed information
- it may be reasonable to virtually restructure the interface

Virtual Restructuring

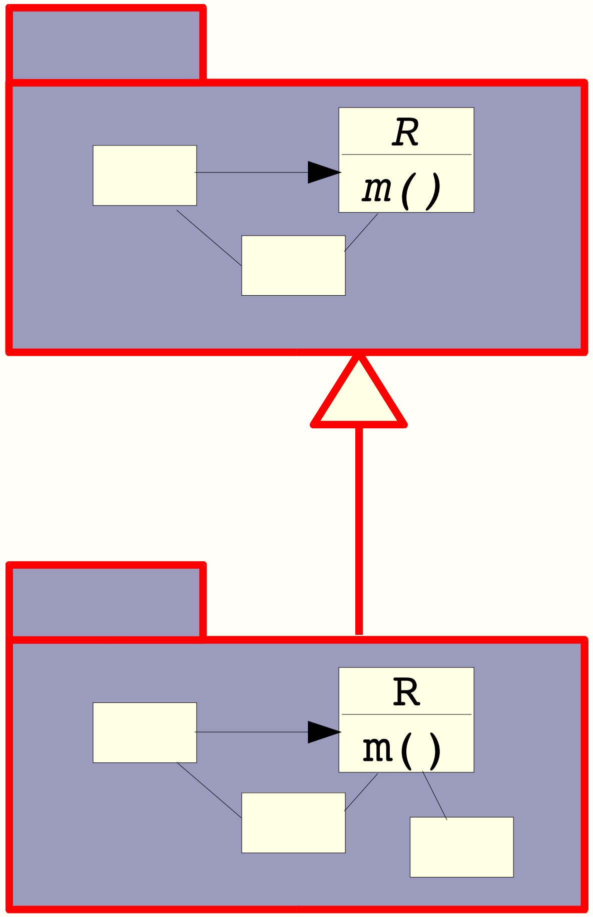


Applicability

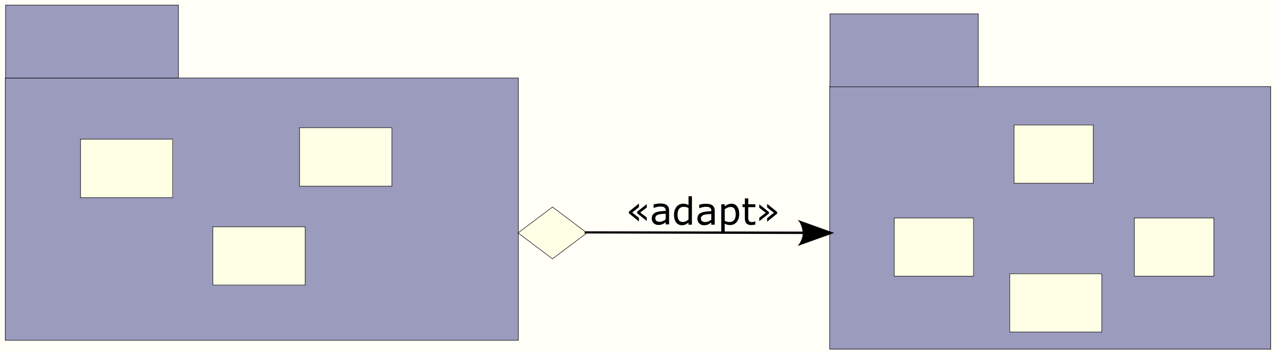
- an enclosing object references a structured type
- the structured type provides features which are not exposed by the interface of the enclosing type
- the existing structure shall not be modified

Adaptation?

- **Team inheritance**
  - Adaptations at hotspots
    - define/override methods
    - define/override role classes
- **Selecting the variant**
  - Team instantiation
  - Role instantiation follows the team
  - Once selected cannot change

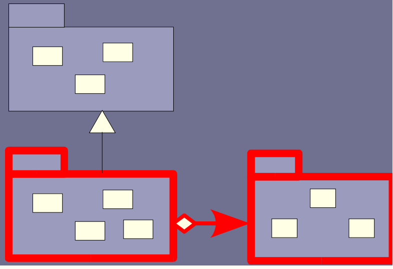


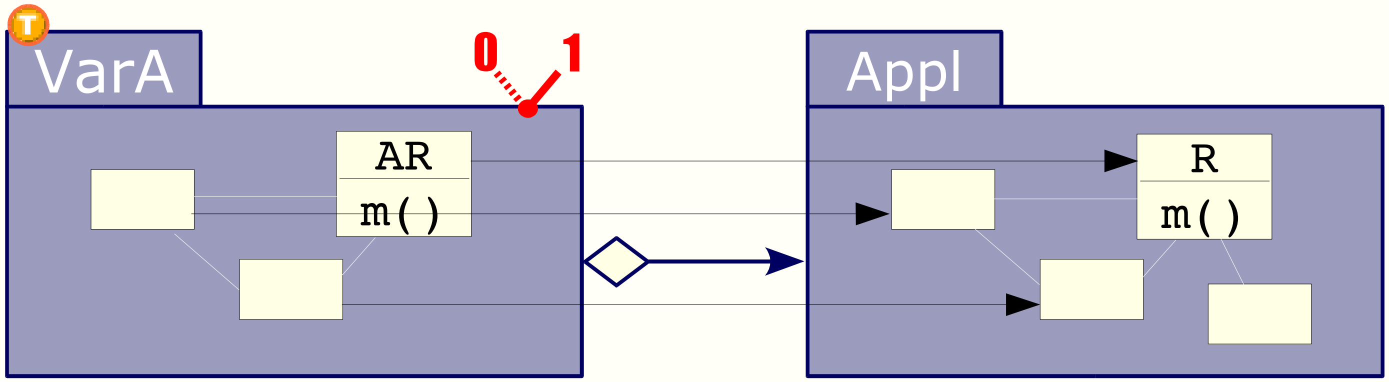
Variant



- **Motivation**
  - Adapt behaviour of a complex module (collaboration)
  - Possibly combine several atomic adaptations
  - Can not use team inheritance:
    - need free combinations of variants
    - need dynamic selection of variants

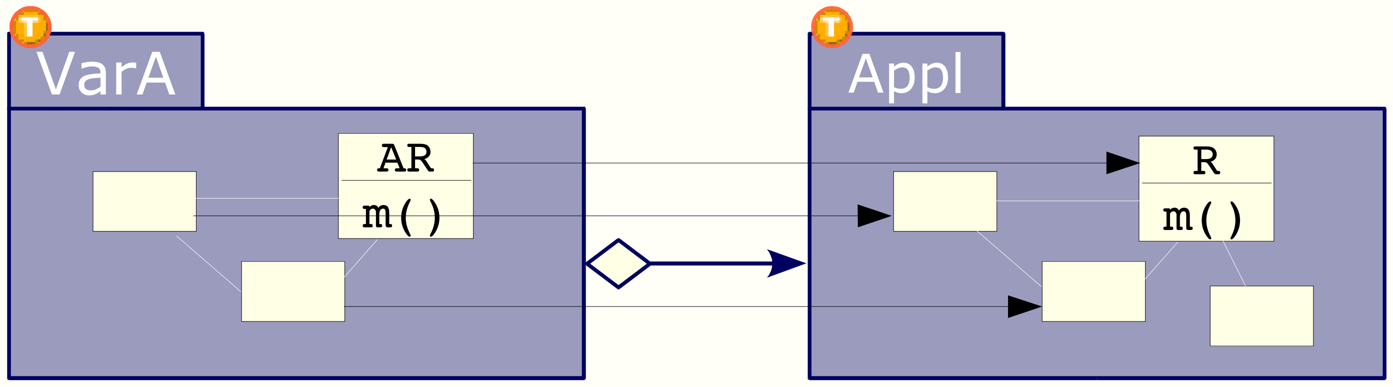
Variant





- **OT/J Solution**
  - Variant is a team with roles
    - bind roles to classes to be adapted
    - callin bind (replace) methods to be adapted
    - additional roles and callout bindings to access the application
  - Team activation selects variant
    - can be changed dynamically
    - multiple active variants possible

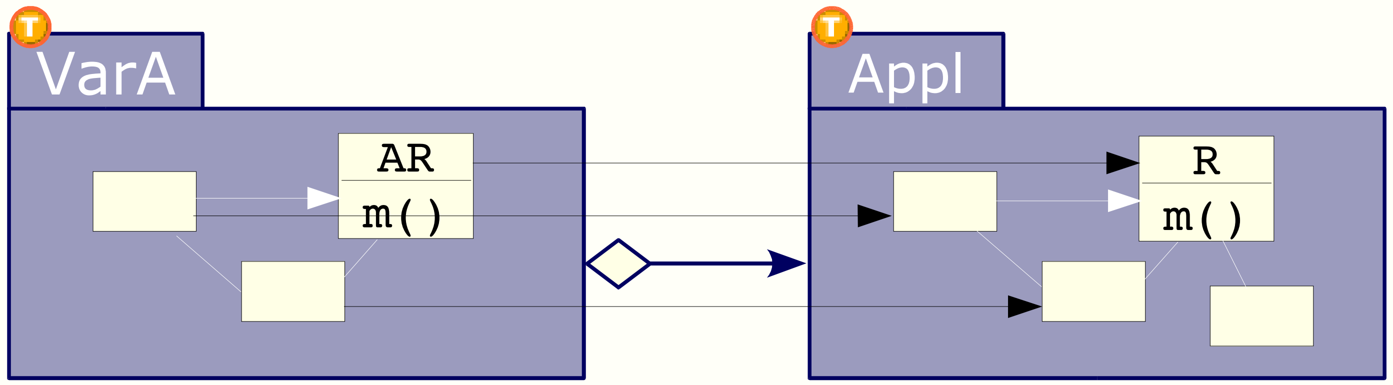
Variant



- **Variants**
  - Base is a plain package
  - Base is a team
    - Variant keeps a reference to the team instance
    - Role binding is relative to this team reference
    - Different team instances can be adapted differently

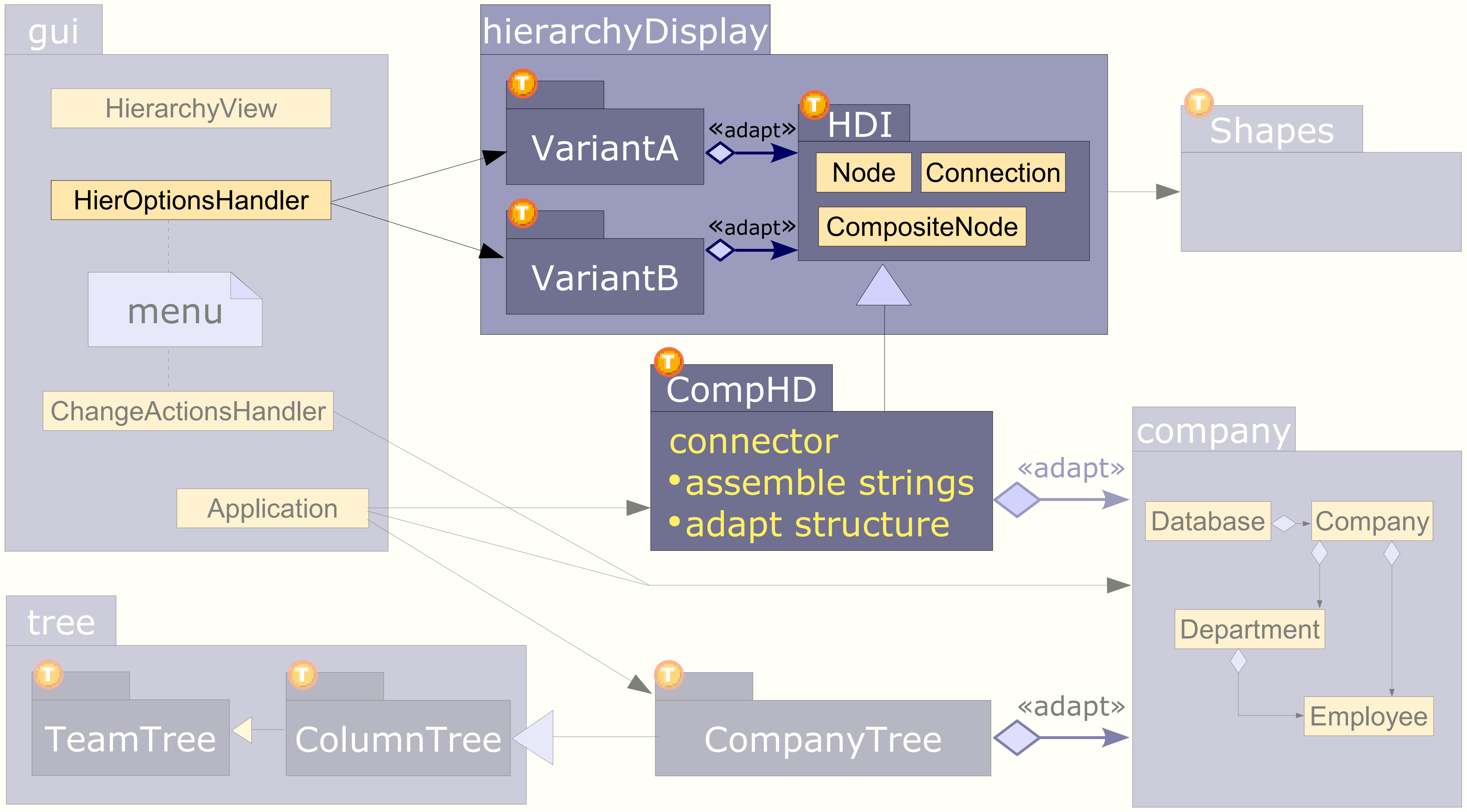
  ⇒ „Aspect of Aspect"

Variant



- **Examples (Company Hierarchy)**
  - Select connection style
    - straight (application default)
    - rectangular (VariantA)
  - Select rectangle sizes
    - fixed (application default)
    - adapting to text size (VariantB)

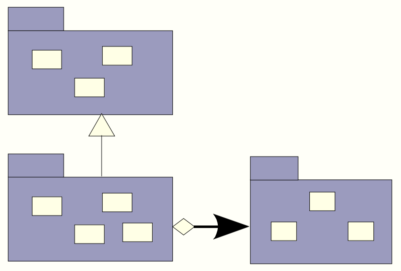
Variants @ Company Hierarchy



Pattern Summary

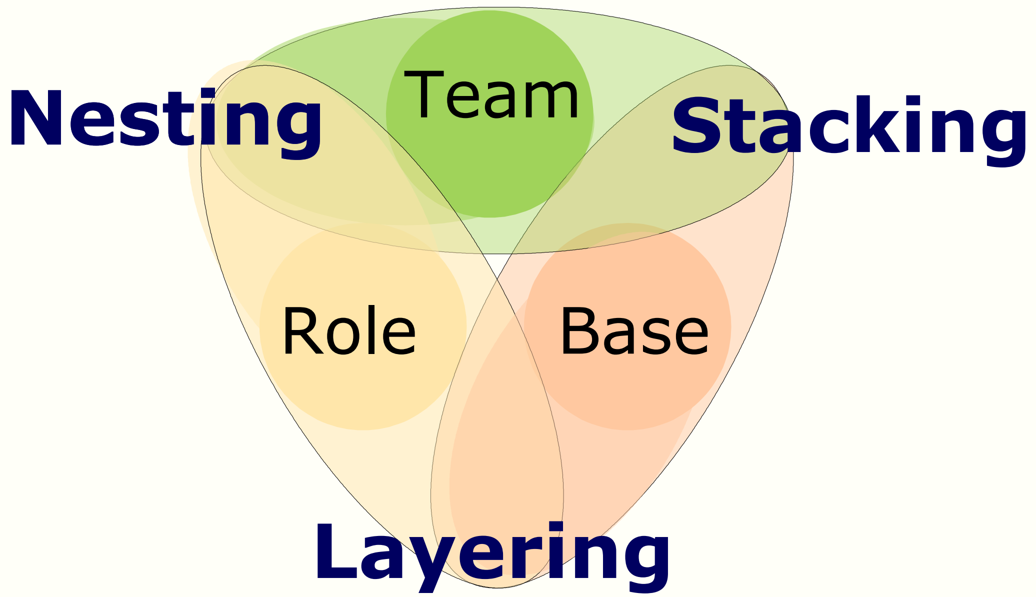
- **Connector**
  - A-posteriori integration of a collaboration into an application.
- **Notification**
  - Define an unanticipated notification protocol between entities.
- **Virtual Association**
  - Navigate between objects without replicating existing associations from another context.
- **Virtual Restructuring**
  - Virtually restructure an objects interface.
- **Variant**
  - Selective adaptation of behaviour to constitute a variant.

Larger Structures

Is  all we can do?

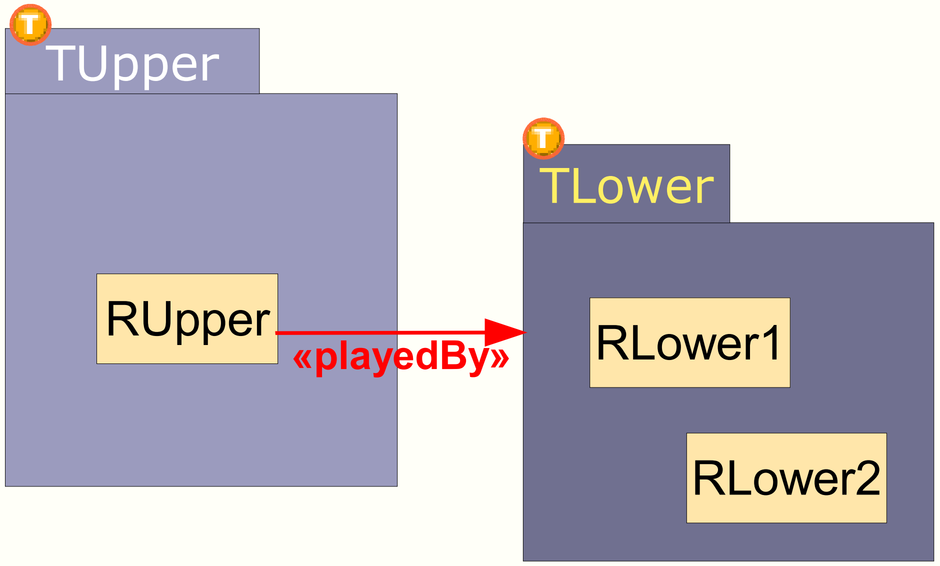
Theorie tells us we have 3 options:

A class can have different natures simultaneously



Stacking

Team & Base

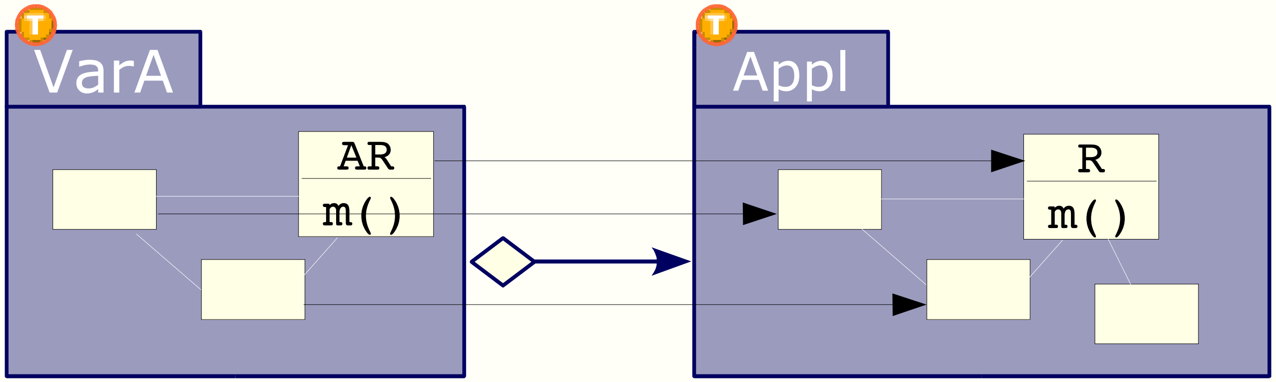


Consequence:

- team methods can be adapted

Layering

Role & Base



Requirement:

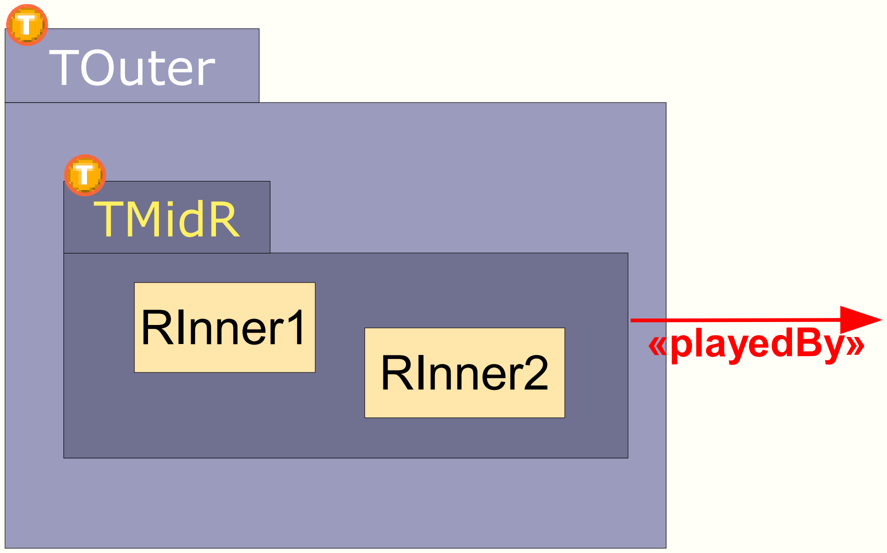
- link between teams

Consequences:

- consistent adaptation of a set of roles
- adapt roles only of a specific team instance
- activation cascading

Nesting

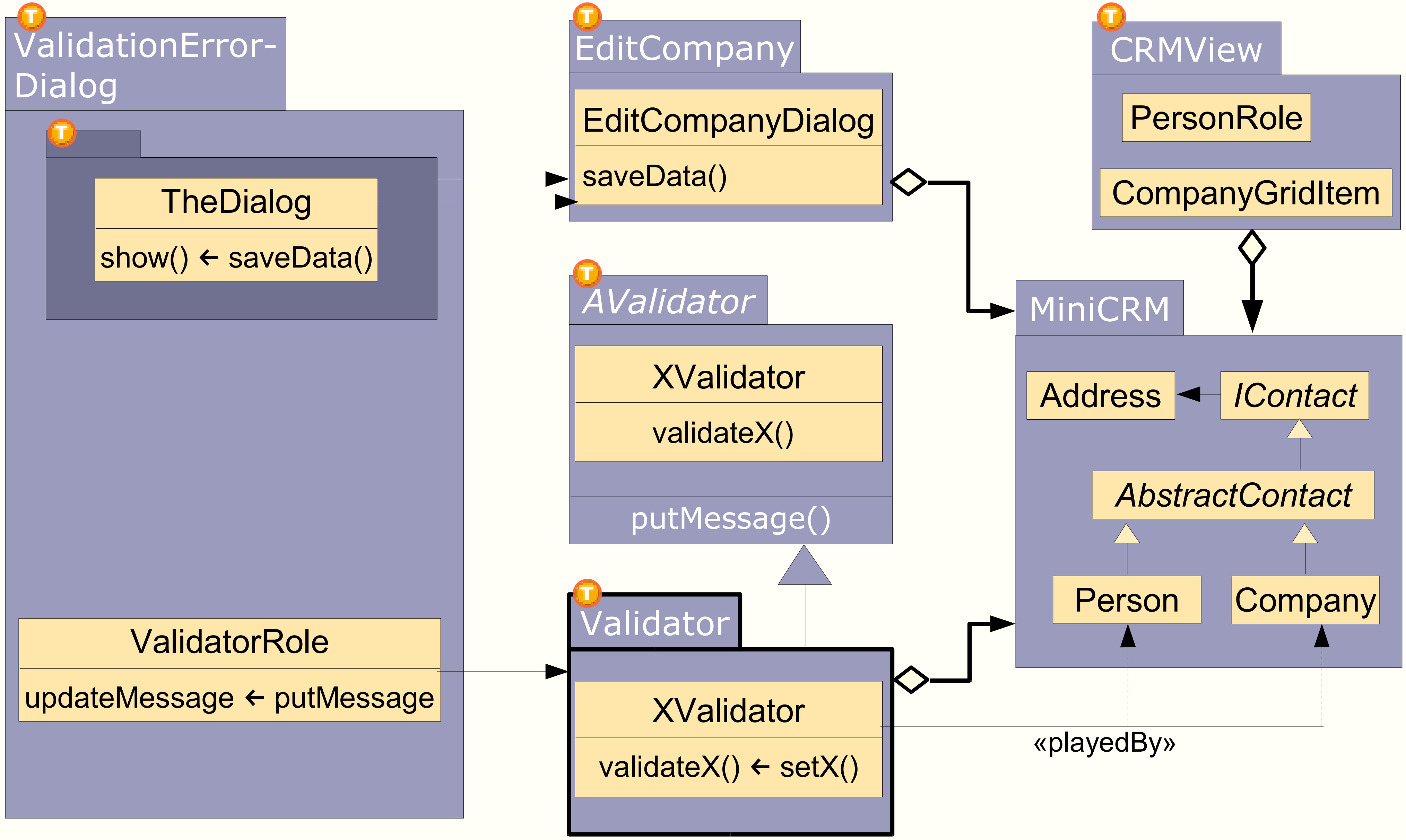
Team & Role



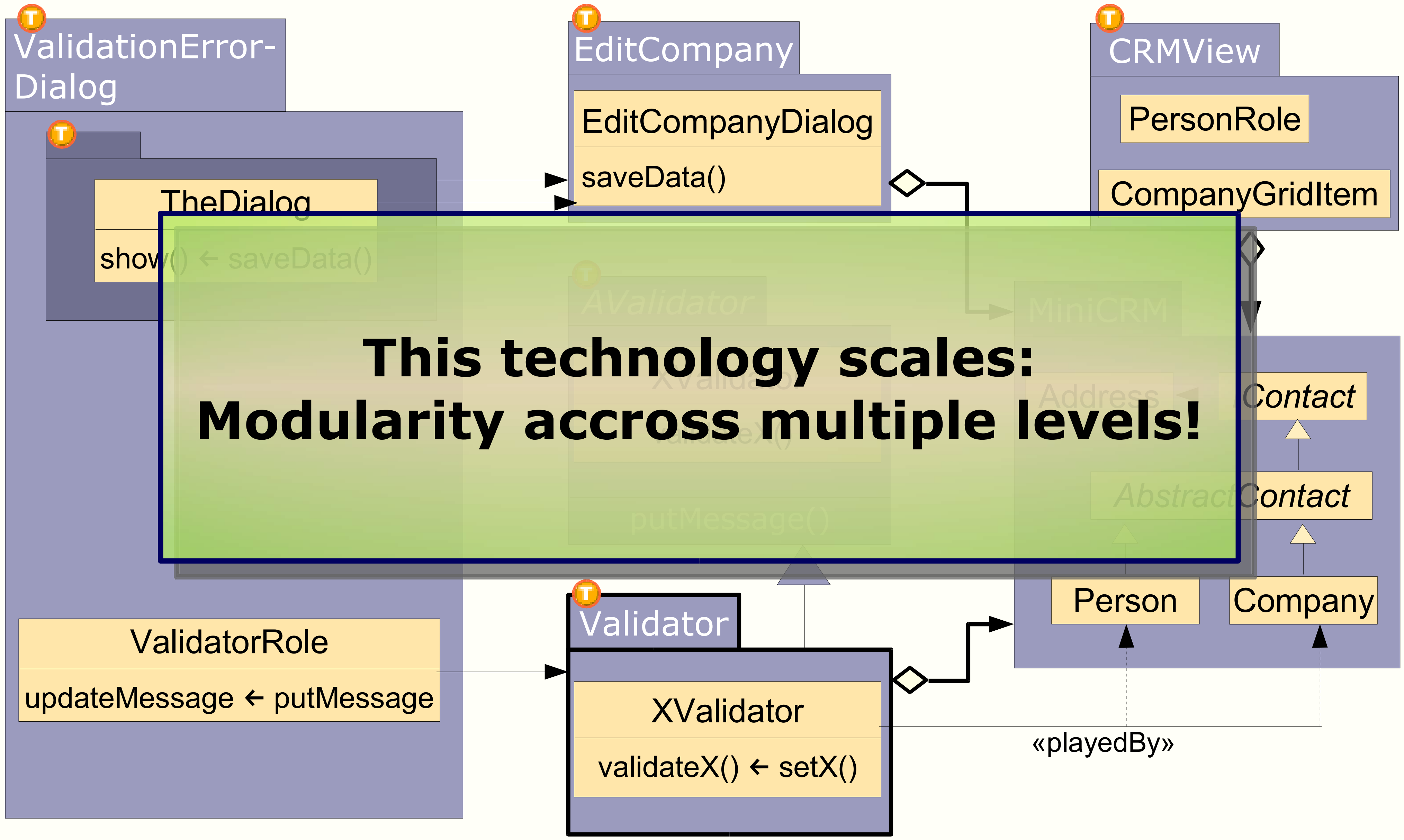
Consequences

- containment
- team may be played by some base class
- lifting to a team

miniCRM revisited



miniCRM revisited



This technology scales:
Modularity accross multiple levels!

Conclusion

- **Concepts explained:**
  - modules larger than classes
  - relations for those modules (adapt, inherit)
  - support different structures simultaneously
- **Aspect Oriented Programming with Views and Collaborations**
- **Rich toolset for ~~Optimal~~ Modularity**
- **Most suitable structure for each concern**

Please give us a visit: **http://www.objectteams.org**

[otj-users] mailing list