Collaboration-Based Design

Dirk Riehle, FAU Erlangen

ADAP D03

Licensed under CC BY 4.0 International

Agenda

1. Collaboration in UML
2. File system collaborations
3. Collaboration and class models
4. History and related concepts

1. Collaboration in UML

Collaboration-Based Design

Collaboration-based design is

- An approach to modeling and implementing using collaborations

A **collaboration specification** (UML: Collaboration) is

- A model of how objects collaborate for a particular purpose

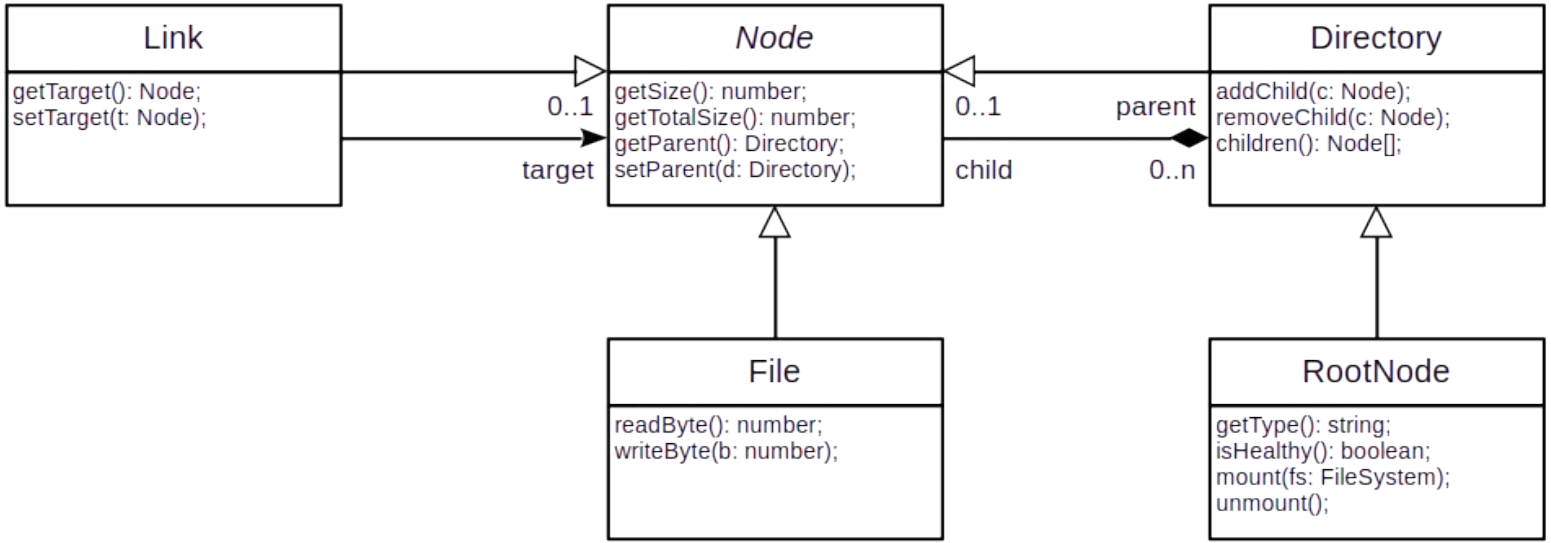
A **collaboration instance** (UML: Collaboration Use) is

- A set of objects collaborating according to a collaboration specification

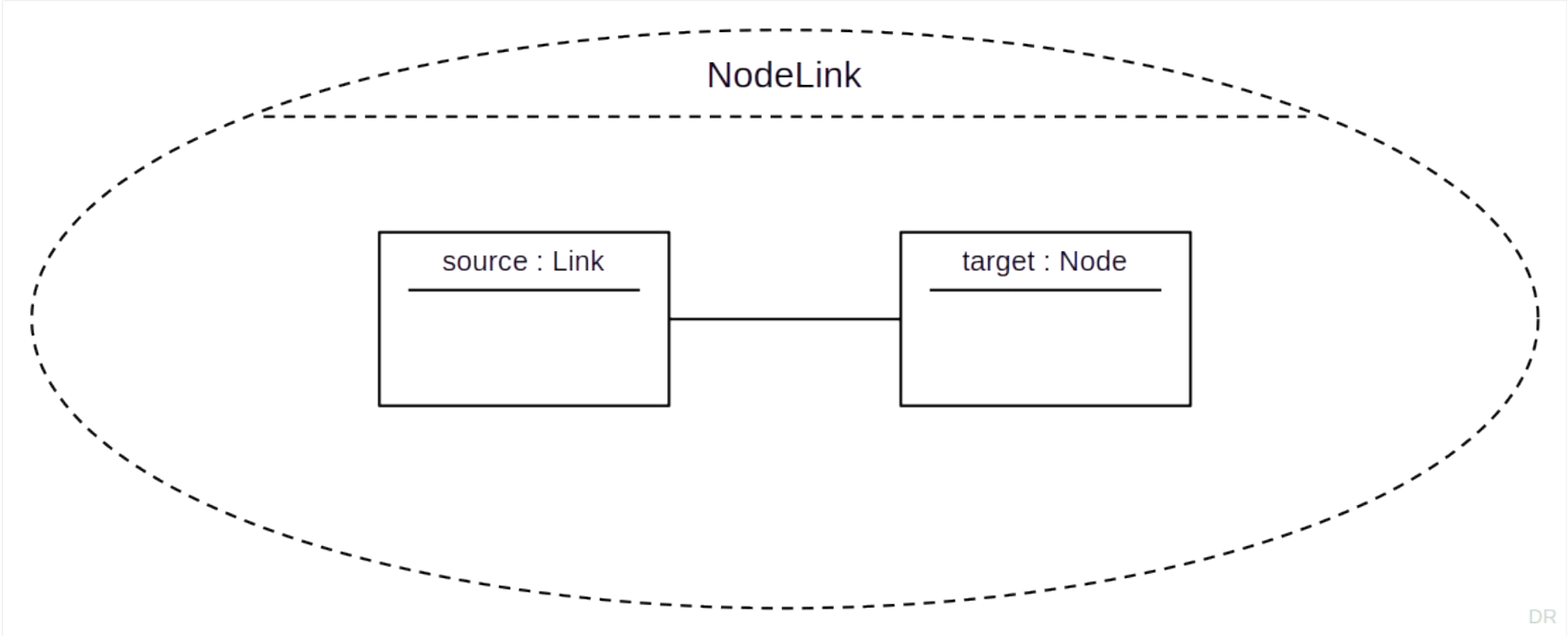
A **role specification** (UML: Role) is

- A model of the behavior of one object in a collaboration instance

Extended File System Example Design



Collaboration Example (UML)



Collaboration as Code (as Parts)

```
export class Link extends Node {

    protected targetNode: Node | null = null;

    constructor(bn: string, pn: Directory, tn?: Node) {
        super(bn, pn);

        if (tn != undefined) {
            this.targetNode = tn;
        }
    }

    public getTarget(): Node | null {
        return this.targetNode;
    }

    public setTarget(target: Node): void {
        this.targetNode = target;
    }

    ...
}
```

```
export class Node {
    ...
}
```

Collaboration Role Binding Example (UML)

Collaboration Roles as Code

```
export interface NodeLinkSource {
   getTarget(): NodeLinkTarget | null;
   setTarget(target: NodeLinkTarget): void;
}

export class Link extends Node implements NodeLinkSource {
    protected target: NodeLinkTarget | null = null;

    constructor(bn: string, pn: Directory, tn?: Node) {
        super(bn, pn);

        if (tn != undefined) {
            this.target = tn;
        }
    }

    public getTarget(): NodeLinkTarget | null {
        return this.target;
    }

    public setTarget(target: NodeLinkTarget): void {
        this.target = target;
    }

    ...
}
```

```
export interface NodeLinkTarget {
   // no methods

}

export class Node implements NodeLinkTarget {
    protected baseName: string = "";
    protected parentNode: Directory;

    constructor(bn: string, pn: Directory) {
        this.doSetBaseName(bn);
        this.parentNode = pn;
        this.initialize(pn);
    }

    protected initialize(pn: Directory): void {
        this.parentNode = pn;
        this.parentNode.addChild(this);
    }

    protected doSetBaseName(bn: string): void {
        this.baseName = bn;
    }

    ...
}
```

Collaboration as Code (as Simulated Language Construct)

```
export collaboration NodeLink {

   role Source {

       protected target: Target | null = null;

       constructor(tn?: Target) {
           if (tn != undefined) {
               this.target = tn;
           }
       }

       public getTarget(): Target | null {
           return this.target;
       }

       public setTarget(target: Target): void {
           this.target = target;
       }

   }

   role Target {
       // no methods
   }

}
```

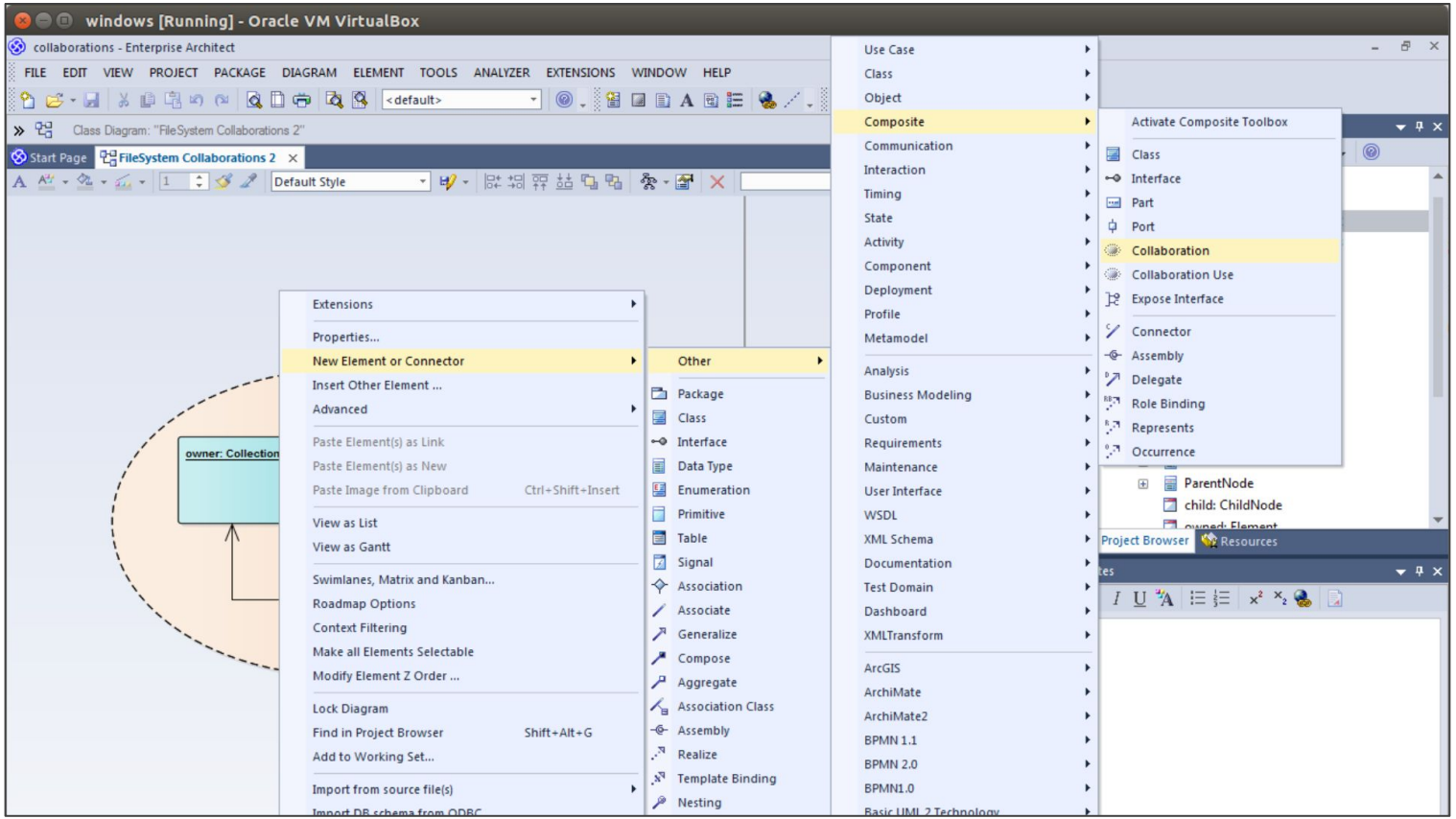
```
import { NodeLink.Target } from "./NodeLink";

export class Node implements NodeLink.Target {
    ...
}
```

```
import * as NodeLink from "./NodeLink";

export class Link extends Node implements NodeLink.Source {

    protected target: NodeLink.Target | null = null;

    ...

    public getTarget(): NodeLink.Target | null {
        return this.target;
    }

    public setTarget(target: NodeLink.Target): void {
        this.target = target;
    }

    ...
}
```

Benefits of Collaboration-Based Design

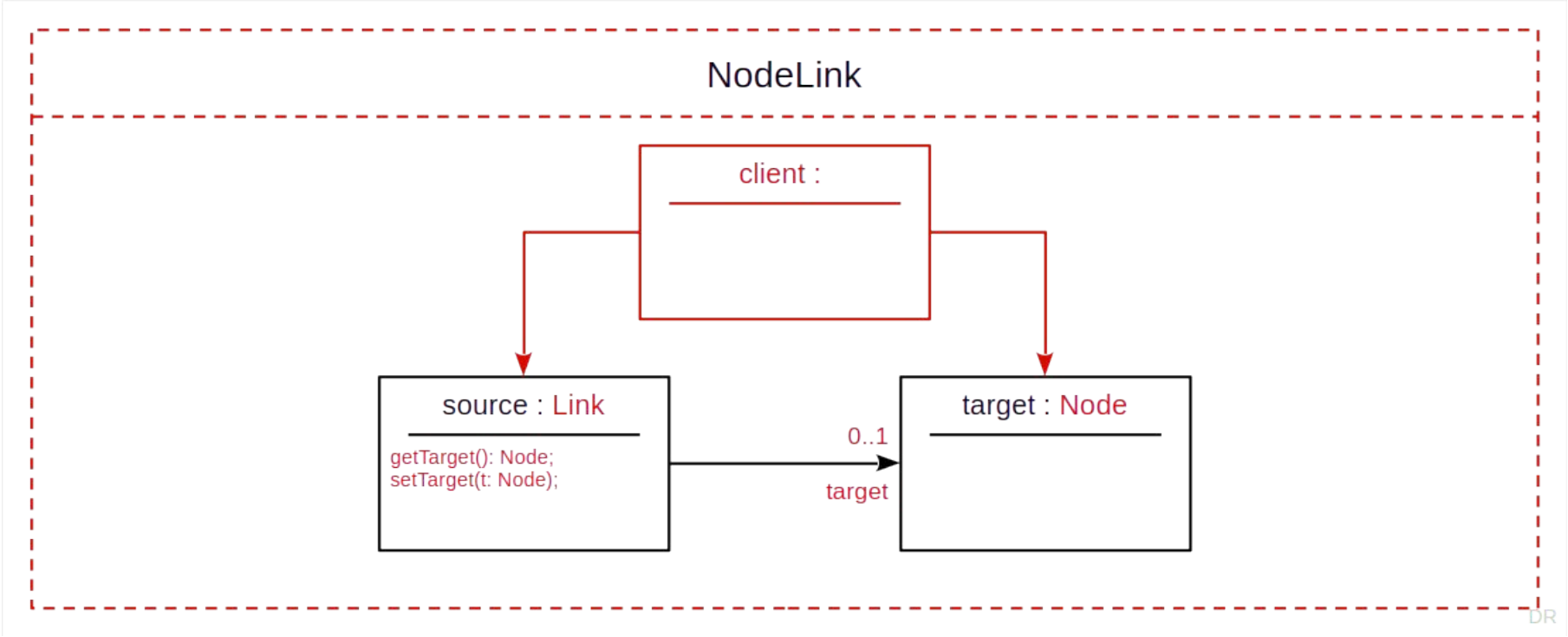
1. Separation of concerns
2. Better reusable models (potentially)

Reality Check: UML Collaboration in Sparx EA



2. File System Collaborations

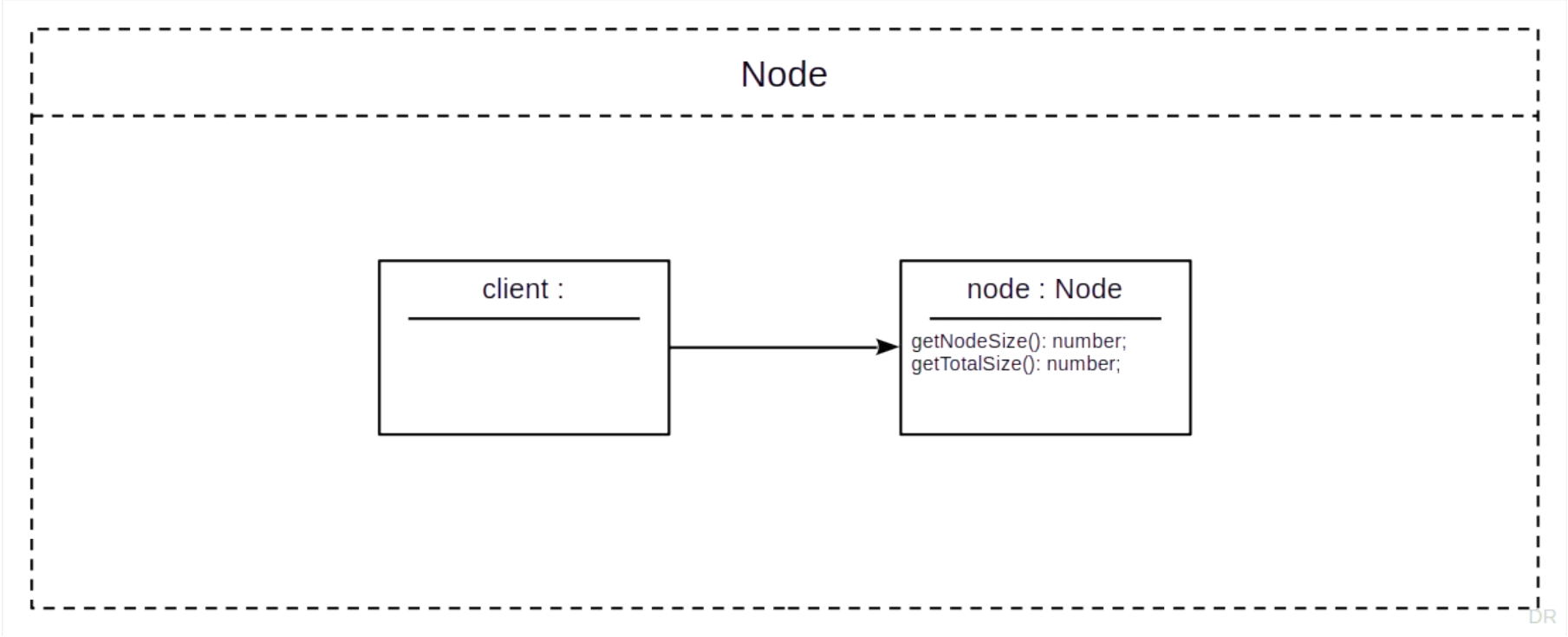
Node Link Collaboration (Riehle's UML Extension)



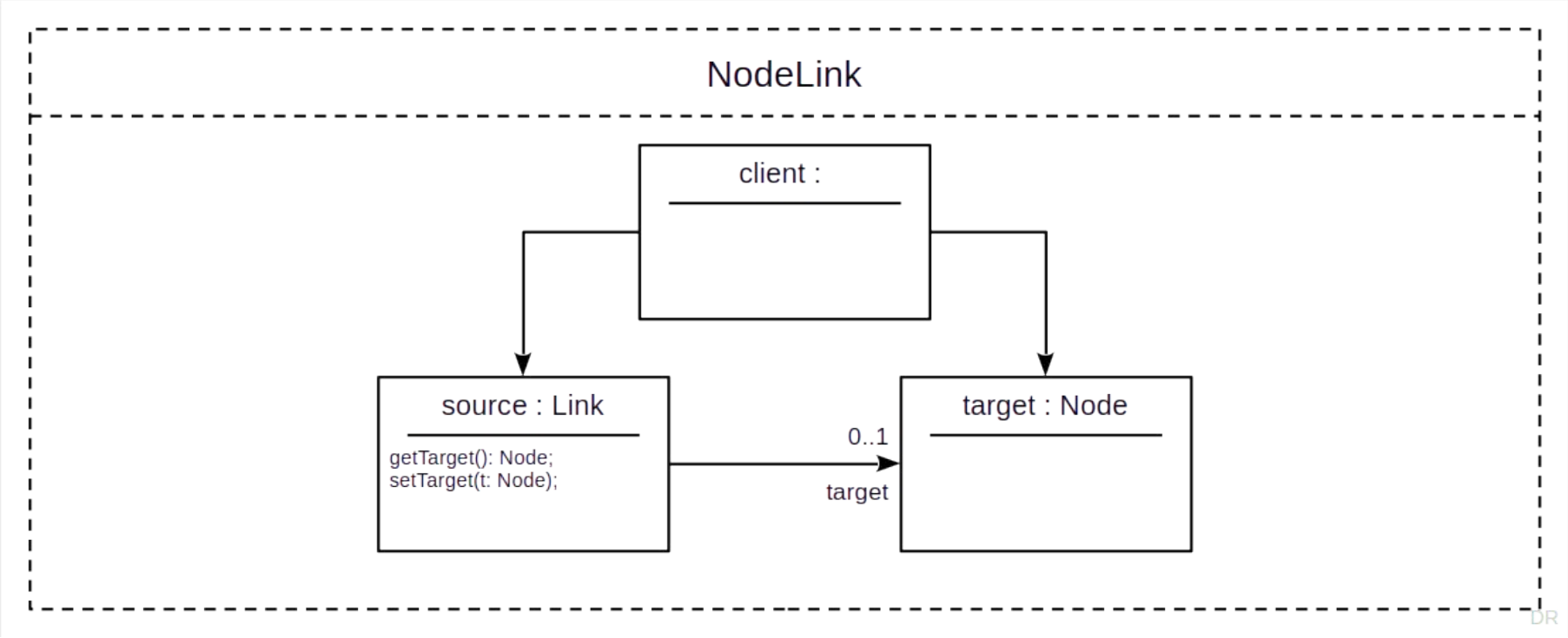
Collaborations in File System Example

1. Node (with client) collaboration
2. NodeLink collaboration
3. File (with client) collaboration
4. ParentChild (with client) collaboration
5. RootNode collaboration

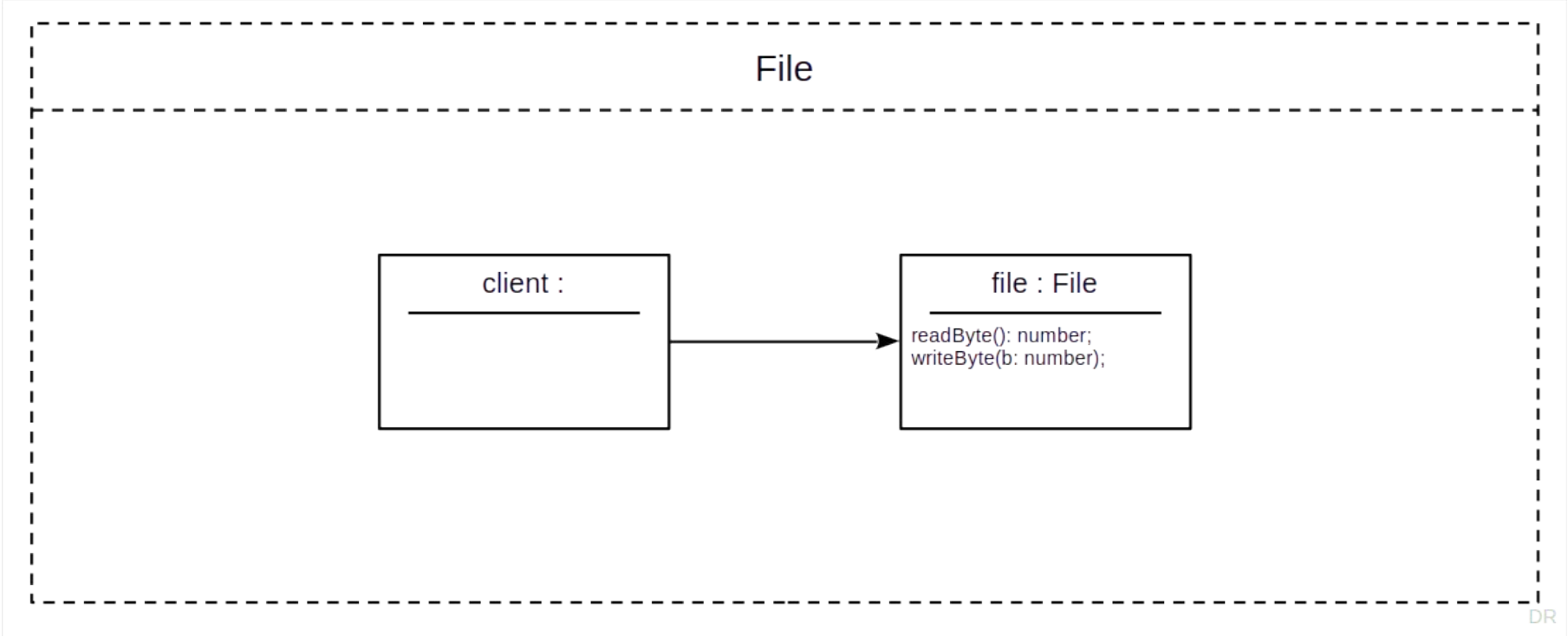
Node Collaboration



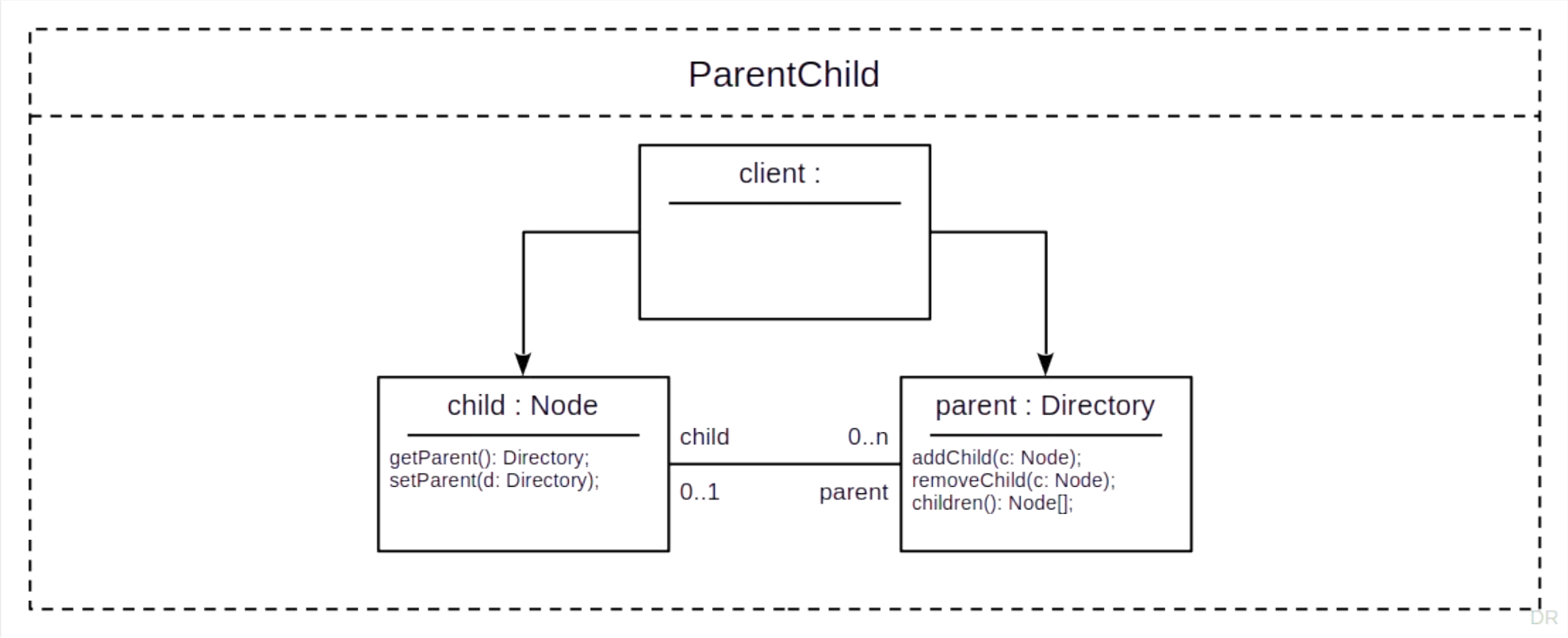
NodeLink Collaboration



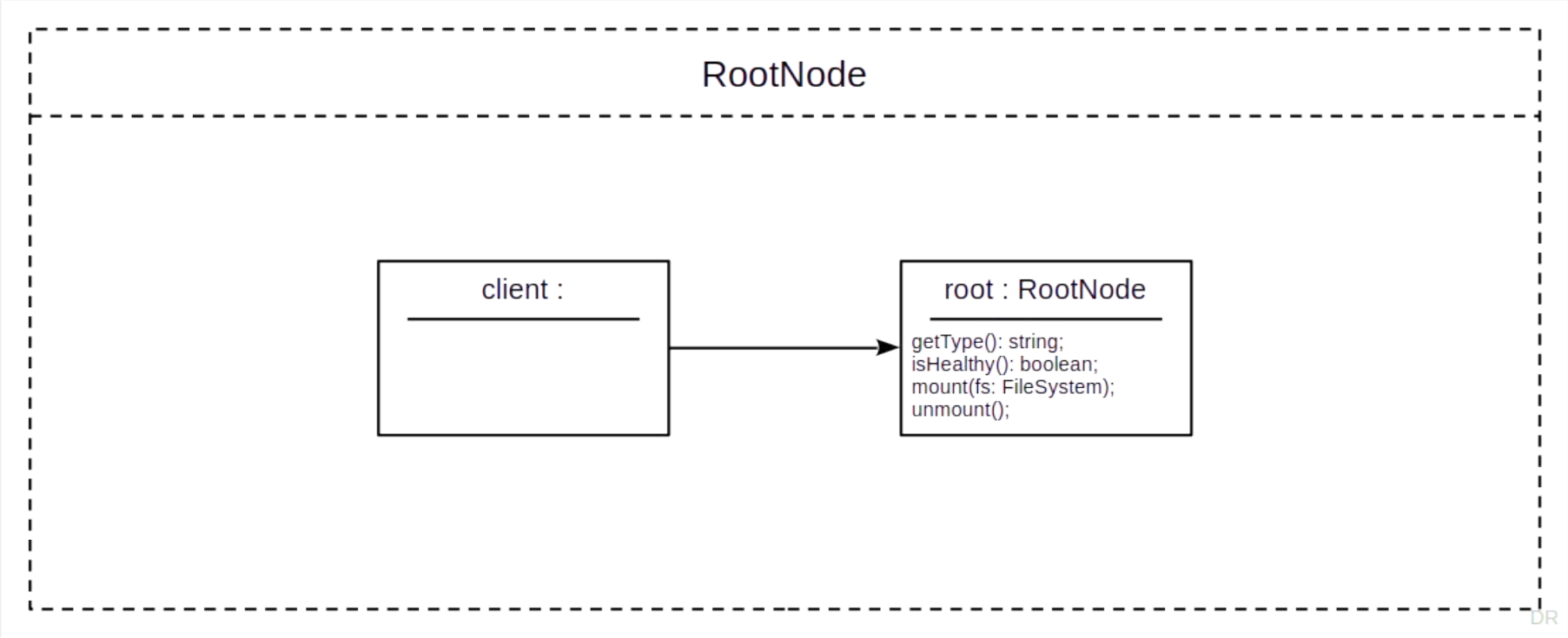
File Collaboration



Parent / Child Collaboration



RootNode Collaboration



Types of Collaborations

Primary service collaborations

- Typically simple client / service collaborations
- The client role often has no methods
- Are visible to the outside (use-clients)

Maintenance collaborations

- Maintain the domain logic with in the model
- Often follow design pattern logic
- Usually not visible to the outside (no use-clients)

3. Collaboration and Class Models

Collaborations / Roles vs. Class Models / Classes

Collaborations

- Focus on behavior

Roles

- Are part of collaborations
- Compose classes from parts
- Use use-client interface

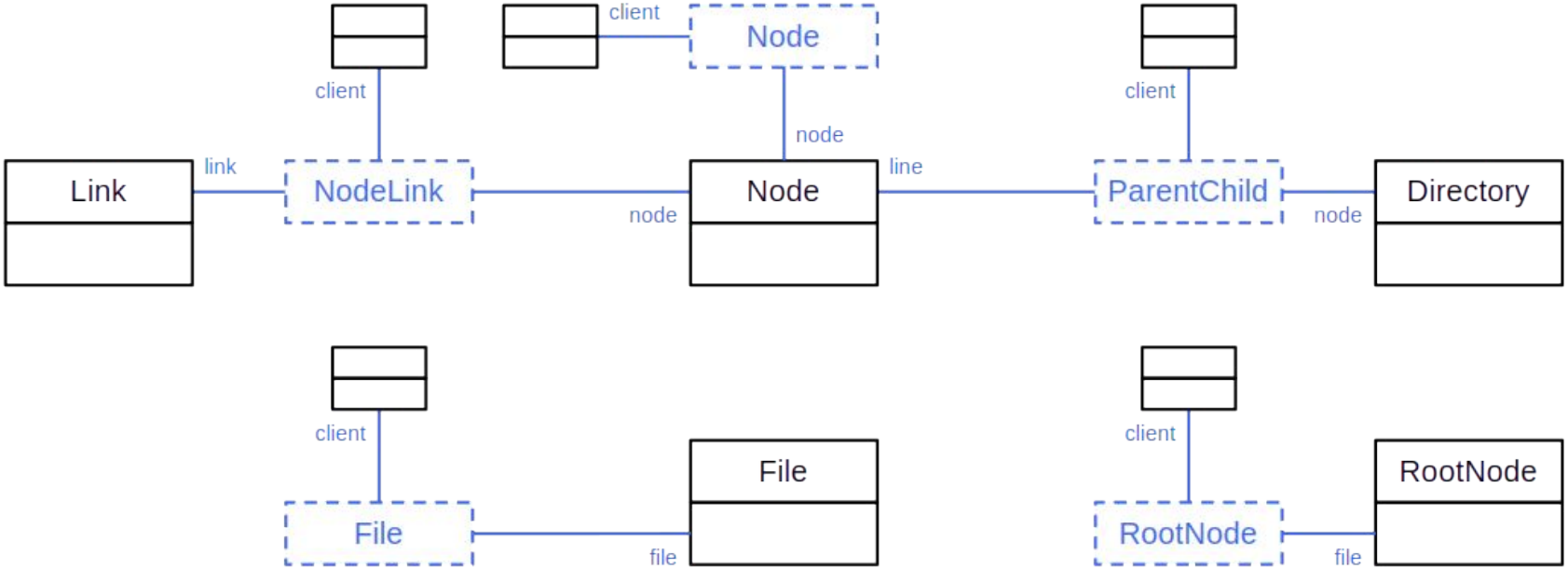
Class models

- Focus on structure

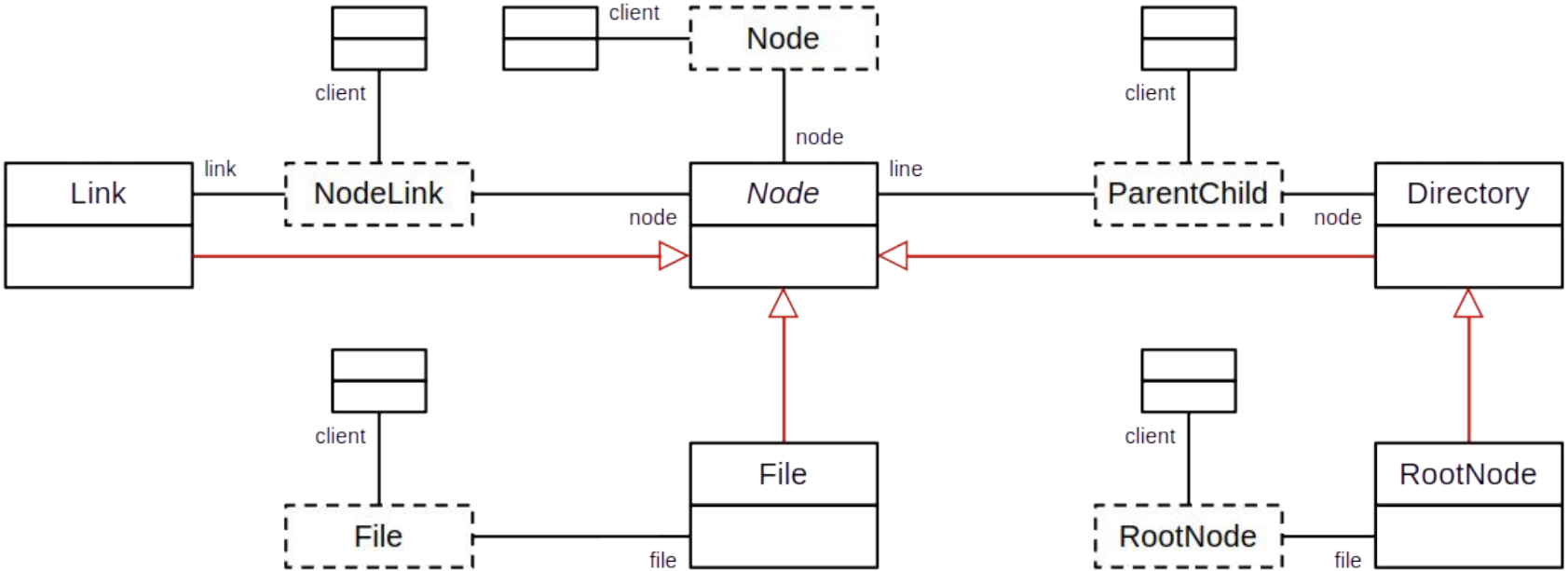
Classes

- Are part of class hierarchies
- Compose classes by inheritance
- Use inheritance interface

Collaboration Role to Class Assignments



Collaborations with Class Hierarchy



Role and Class Duality

A role ties a class into a collaboration context

- Control flow enters and leaves through a role

A class composes behavior in multiple contexts into one unit

- Role implementation is mapped onto implementation state

Levels of Abstraction

Design patterns

- No modeling support

Design templates

- UML Collaboration

Class models

- UML Class Model

4. History and Related Concepts

History of Collaboration Modeling Methods

1. Trygve Reenskaug, Role Modeling [1]
2. Dirk Riehle, Framework Design [2]
3. OMG, UML 2.0 Specification [3]

[1] Reenskaug, T., Wold, P., & Lehne, O. A. (1996). Working with Objects. Manning.
[2] Riehle, D. (2000). Framework Design: A Role Modeling Approach. Doctoral Dissertation, no. 13509. Swiss Federal Institute of Technology at Zurich (ETH Zurich), Zurich, Switzerland.
[3] OMG (2005). Unified Modeling Language 2.0. OMG.

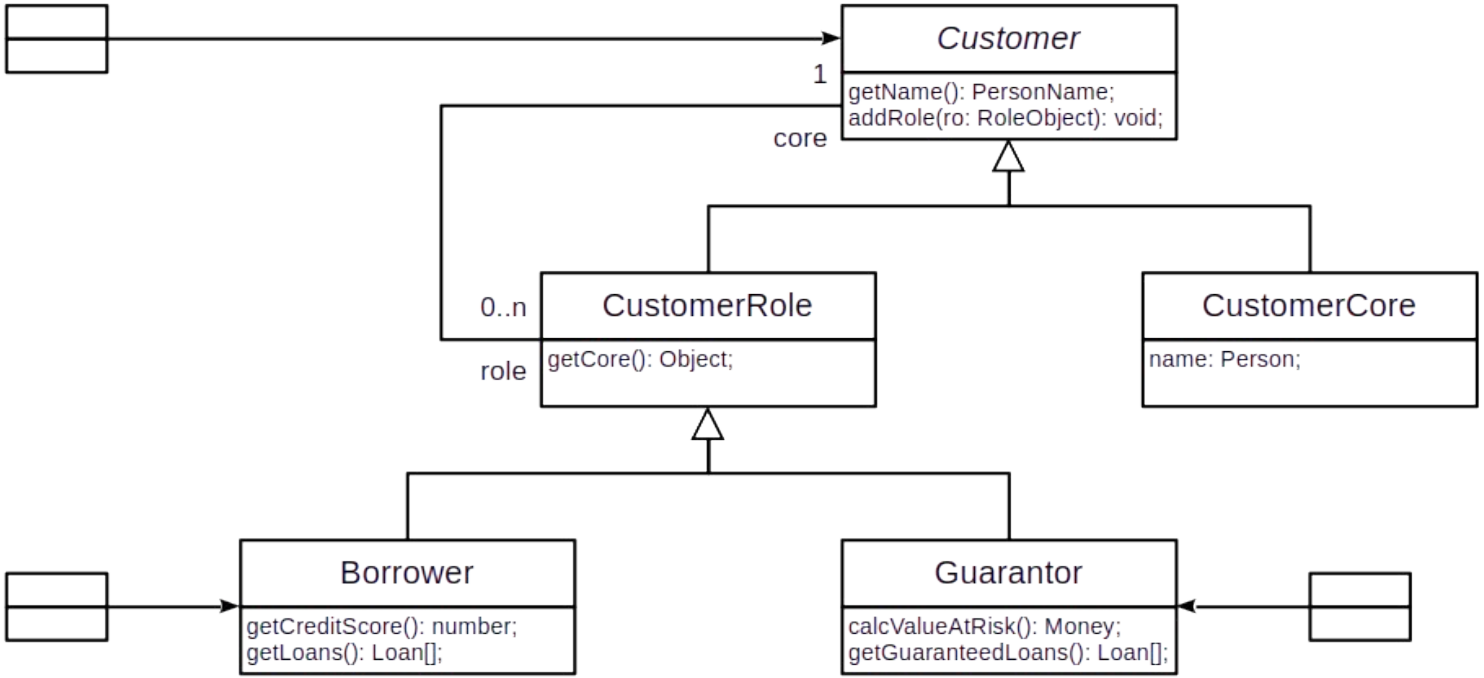
Related Concepts from Programming Language History

Past attempts

- Interfaces
- Protocols
- Mix-ins
- Traits

Common to all is that they ignore the collaboration, focus only on roles

Role Object Example



[1] Bäumer, D., Riehle, D., Siberski, W. & Wulf, M. (2000). Role Object. In Pattern Languages of Program Design 4 (PLoPD 4), pp. 15-32. Addison-Wesley.

Role Object Pattern

Context

- A large modular system with core domain concepts and varied applications

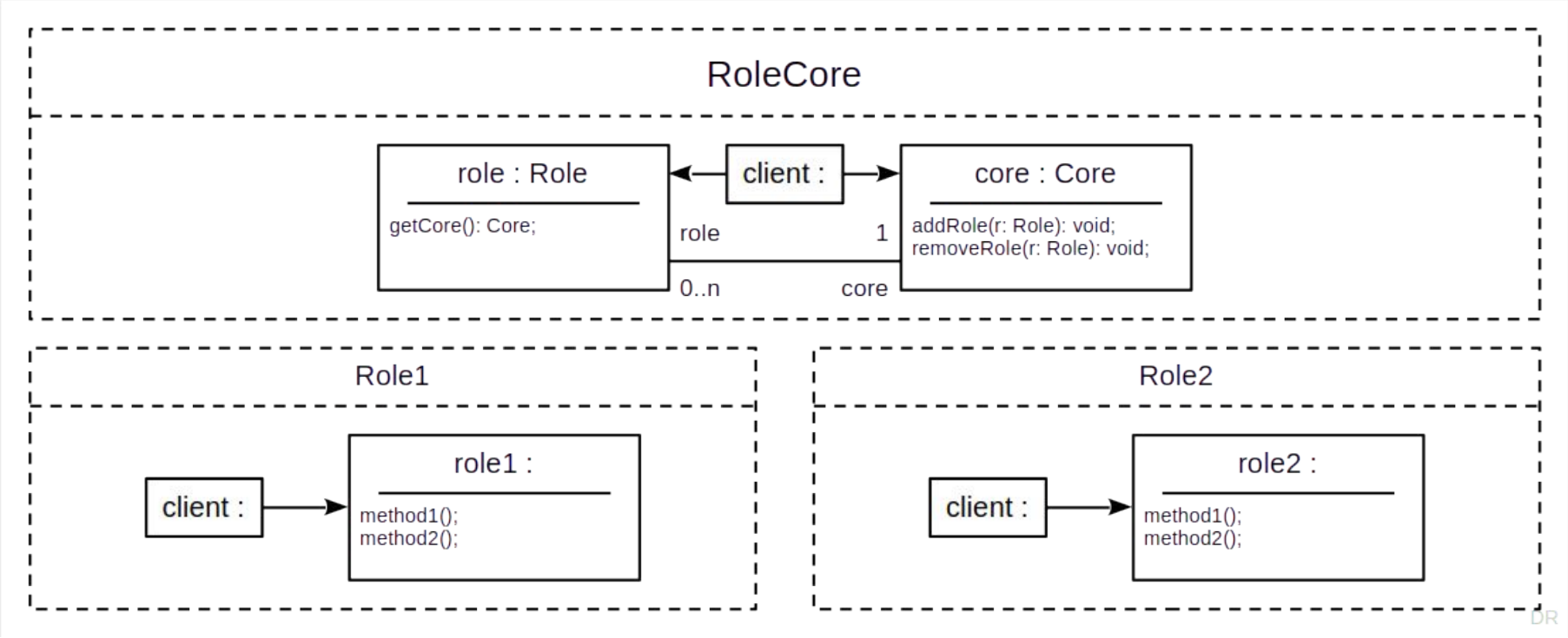
Problem

- A core concept needs to show a different face in each application

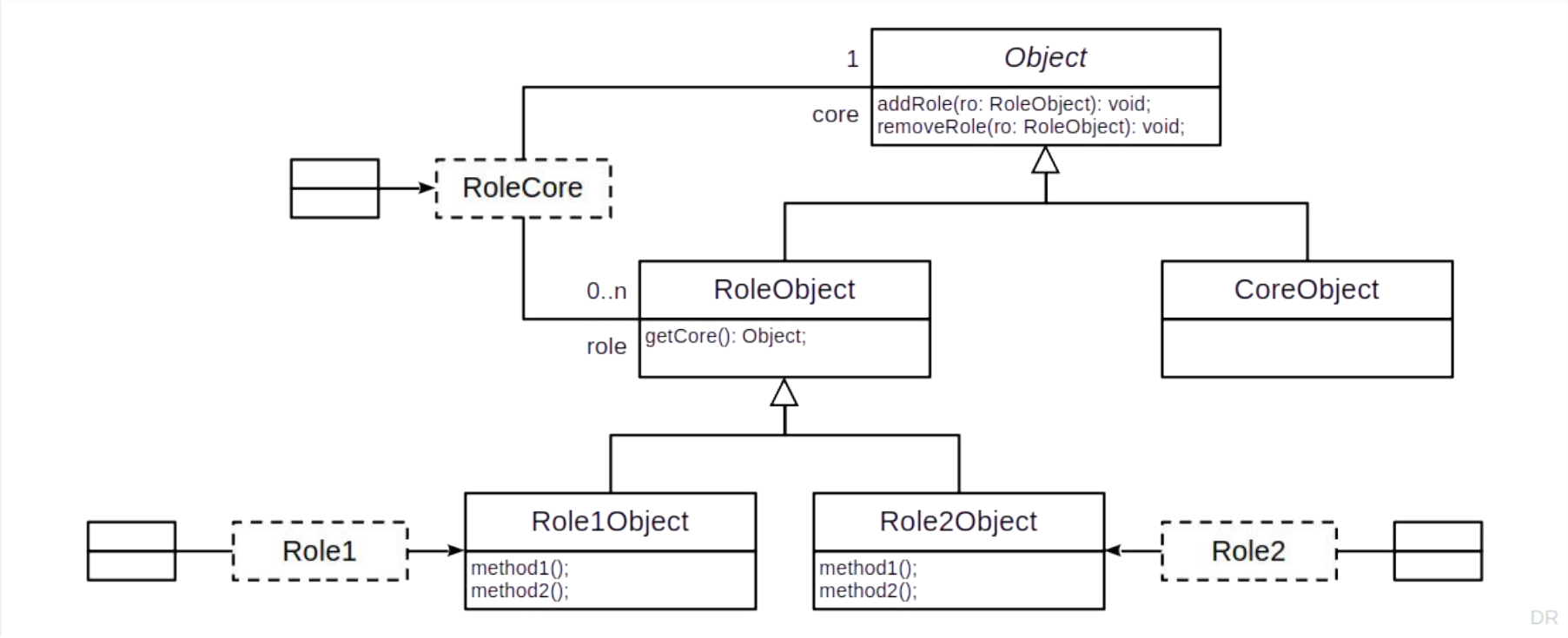
Solution

- Extend the core concept with separate role objects facing the application

Role Object Collaborations



Role Object Structure Diagram



Summary

1. Collaboration in UML
2. File system collaborations
3. Collaboration and class models
4. History and related concepts

Thank you! Any questions?

dirk.riehle@fau.de – https://oss.cs.fau.de

dirk@riehle.org – https://dirkriehle.com – @dirkriehle

Legal Notices

License

- Licensed under the CC BY 4.0 International license

Copyright

- © 2012, 2018, 2024 Dirk Riehle, some rights reserved