Collaboration-Based Design

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

ADAP C11

Collaboration-based Design

- **Collaboration-based design**
  - An approach to modeling and implementation using collaborations
- **Collaboration (specification / description)** a.k.a. role model
  - A model of objects collaborating for one particular purpose
- **Role (type / specification / description)**
  - A model of the behavior of one object within a collaboration
- **Collaboration (instance)**
  - A set of specific objects collaborating according to a collaboration specification
- **Object**
  - The representation of a phenomenon playing roles in collaborations

Benefits of Collaboration-based Design

1. **Separation of Concerns**
2. **Better Reusable Models**

File System Example Revisited

Primary Service Collaborations

Primary Service Roles as Code

```
public class Node {
  // Client-Node-Collaboration
  public String getName();
  public void setName(String name);
  ...
}
```

```
public class Link extends Node {
  // Client-Link-Collaboration
  public Node getTarget();
  ...
}
```

```
public class File extends Node {
  // Client-File-Collaboration
  public void write(byte[] data);
  ...
}
```

Hierarchy Collaboration

Secondary Service Roles as Code

```
public class Node {
  // Hierarchy-Collaboration
  public Node getOwner();
  public void setOwner(Node n);

  // Other collaborations
  ...
}
```

```
public class Directory extends Node {
  // Hierarchy-Collaboration
  public void addOwned(Node n);
  public void removeOwned(Node n);
  public Iterater getIterator();

  // Other collaborations
  ...
}
```

Parent Child Collaboration

Maintenance Roles as Code

```
public class Node {
  // Parent-Child-Collaboration
  protected Node getParent();
  protected void setParent(Node n);

  // Other collaborations
  ...
}
```

```
public class Directory extends Node {
  // Parent-Child-Collaboration
  // No methods

  // Other collaborations
  ...
}
```

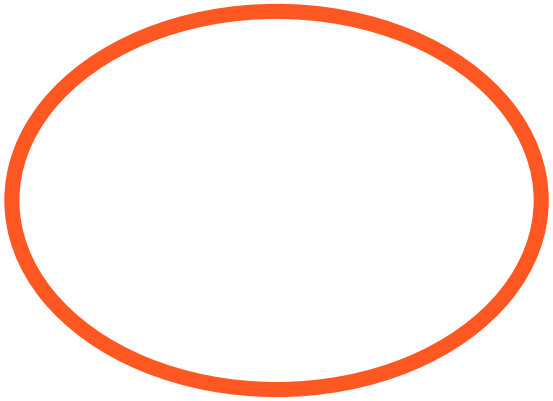
Types of Collaborations

- Primary service collaborations
  - Typically, client-service-collaborations
  - The client role often has no methods
  - Visible to the outside (of the model)
- Secondary service collaborations
  - Client-service-collaborations used for technical purposes
  - Often follow design patterns to realize logic
  - Visible to the outside (of the model)
- Maintenance collaborations
  - Collaborations that maintain the domain logic within the model
  - Often follow design patterns to realize logic
  - Usually not visible to the outside

Collaboration / Class Duality

- A collaboration focuses on
  - the interaction of objects for one purpose
- A class focuses on
  - the integration the roles an object plays in multiple collaborations

Collaborations and Role Binding to Classes



Collaboration-based Design and Reuse

- For a collaboration to be used in multiple contexts
  - It needs to be independent of those contexts
    - Naming cannot be context-specific → Node becomes Owned
  - It must be possible to apply it to those contexts
    - Naming should be adjustable (method renaming) → Owned becomes Node
  - Role composition (in one class) needs to be made explicit
    - Composition has important domain analysis meaning
- Reusable models need programming language support

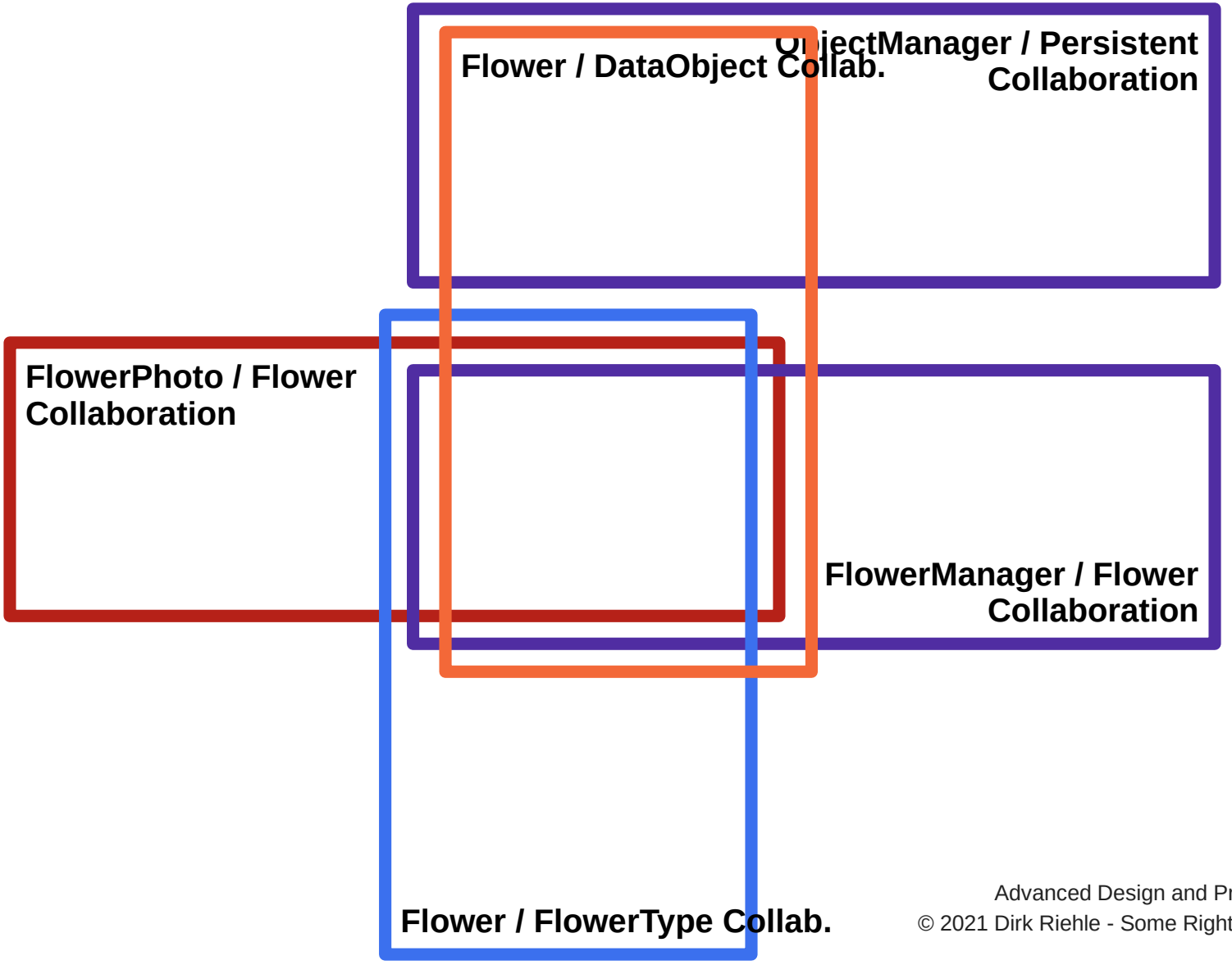
Levels of Abstraction

	Design Pattern	Design Template	Class Model
Level	Design illustration	Design template	Specific model
Language	No formal language available	UML-Collaboration	UML-Class-Model, UML-Collaboration-Use
Use in CBD	N/A	Collaboration	Role binding
Example	N/A	Hierarchy = { Client, Owner, Owned } ParentChild = { Parent, Child }	Hierarchy.Owner → Directory ParentChild.Parent → Directory

UML and Collaboration-based Design [1]

[1] From the UML 2.5 specification (May 2011)

Concrete Syntax for Collaborations



Flower Collaborations

- FlowerPhoto / Flower Collaboration
  - Purpose: Provide main domain functionality
  - Role types: FlowerPhoto (Client), Flower (Service)
- Flower / FlowerType Collaboration (Type Object)
  - Purpose: Provide information common to all instances of a type
  - Role types: Client, Flower (Base Object), FlowerType (Type Object)
- FlowerManager / Flower Collaboration (Manager)
  - Purpose: Centralize object management in one place
  - Role types: Client, FlowerManager (Manager), Flower (Element)

Lessons from the Flower Collaborations

- Collaborations
  - Often have an implicit Client role
  - Almost always overlap when applied to a class model
  - There should be inheritance between collaborations
  - Many collaborations are design pattern applications

Collaborations in Programming [R00]

```
public collaboration ParentChild {
  public role Parent {
    public void addChild(Child c);
    public void removeChild(Child c);
    public Iterator<Child> getIterator();
  }

  public role Child { ... }
  ...
}
```

```
public class Node binds ParentChild.Child {
  ...
}
```

```
public class Directory extends Node binds ParentChild.Parent {
  ...
}
```

Roles as Code Templates [V97]

```
public interface Owner<C> {
  public void addOwned(C c);
  public void removeOwned(C c);
  public Iterator<C> getIterator();
}
```

```
public class Node {
  protected Node parent = null;
  protected Node getParent() { ... }
  protected void setParent(Node n) { ... }
  ...
}
```

```
public class Directory extends Node, implements Owner<Node> {
  public void addOwned(Node n);
  public void removeOwned(Node n);
  public Iterator<Node> getIterator();
  ...
}
```

Client-side Role Specifcations

```
public collaboration File {
  public role Client {
    // no methods, but specification of
    // behavioral constraints, e.g.
    // no read or write before open or after close
  }

  public role File {
    public void open();
    public byte[] read(int);
    public void write(byte[]);
    public void close();
    ...
  }
}
```

Similar / Related to Roles

- **Interfaces**
- **Protocols**
- **Mix-ins**
- **Traits**

Composite Pattern Revisited [G+95]

Participants Section of Composite Pattern

- Component (Graphic)
  - declares the interface for objects in the composition.
  - implements default behavior for the interface common to all classes, as appropriate.
  - declares an interface for accessing and managing its child components.
  - (optional) defines an interface for accessing a component's parent in the recursive structure, and implements it if that's appropriate.
- Leaf (Rectangle, Line, Text, etc.)
  - represents leaf objects in the composition. A leaf has no children.
  - defines behavior for primitive objects in the composition.
- Composite (Picture)
  - defines behavior for components having children.
  - stores child components.
  - implements child-related operations in the Component interface.
- Client
  - manipulates objects in the composition through the Component interface.

Composite Pattern as Role Model [R97]

Design Pattern Composition [R11]

Review / Summary of Session

- Collaboration-based design
  - Definitions, constituents
  - In design and programming
- Design patterns, templates, models
- Composition of models

Thank you! Questions?

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

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

Credits and License

- Original version
  - © 2012-2021 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - None yet