Object Oriented Frameworks

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

ADAP C12

Components

- A component is some entity with a defined boundary; it should have
  - High internal cohesion
  - Low external coupling
- Components can be composed from smaller components
  - Atomic components may be files (code) or functions (run-time)
- There are two types of components
  - Code components (source code in directories, compiled binaries)
  - Run-time component (may or may not map on code components)
- Practically, you always take either about code or run-time components
  - Only modeling language designers may care about the more general term
  - Why? Because you are either designing a code architecture or run-time architecture

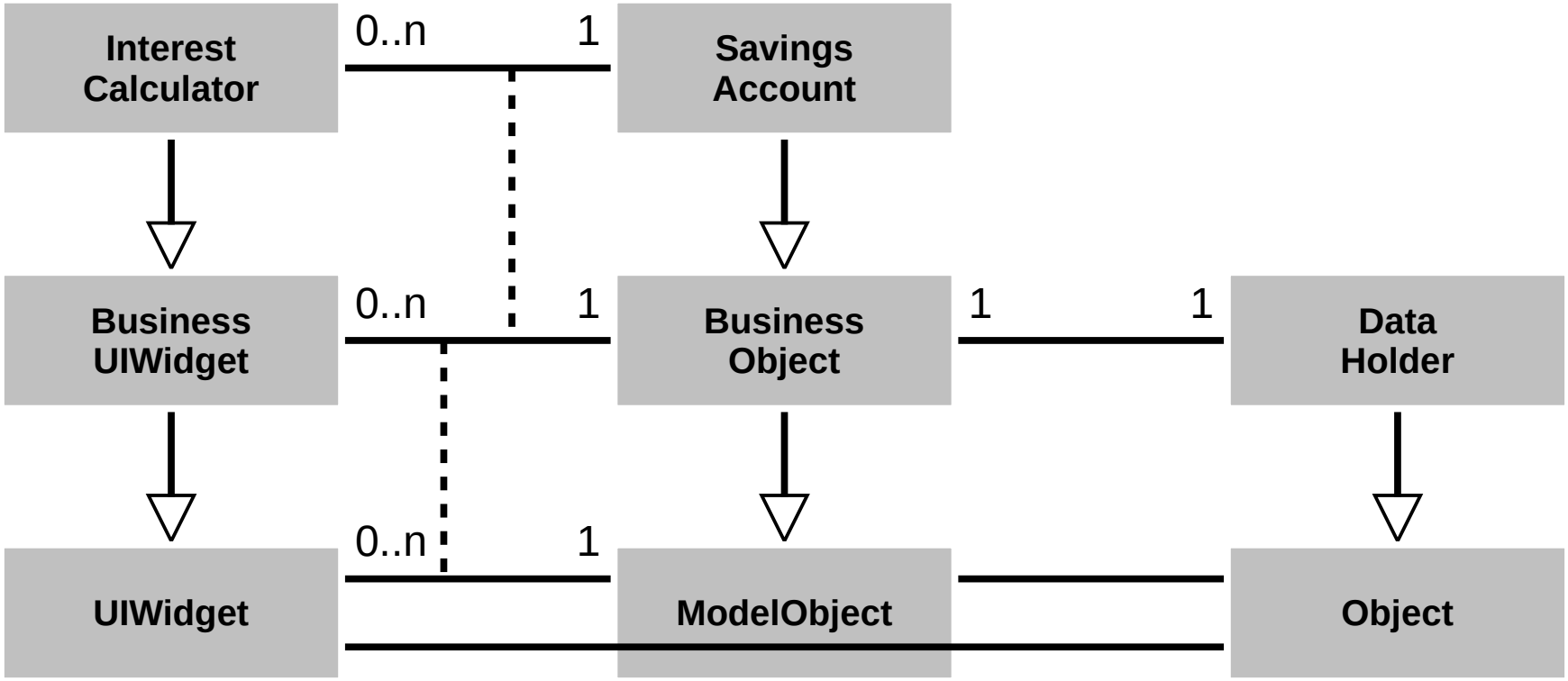
Code Components

- A set of source code files, compiled into a binary or related delivery format
  - With high cohesion and low coupling
- Example delivery formats for code components
  - Java: .class files, jar-files
  - C: .o files, shared libraries
  - Web servers: war files and more
- Source code is usually compiled into one binary, not reused as source code
  - Only (re-used) as the binary as part of a code component architecture
- Code components can be aggregated into larger code components
  - Used to be done mainly for binaries, not source code; is changing

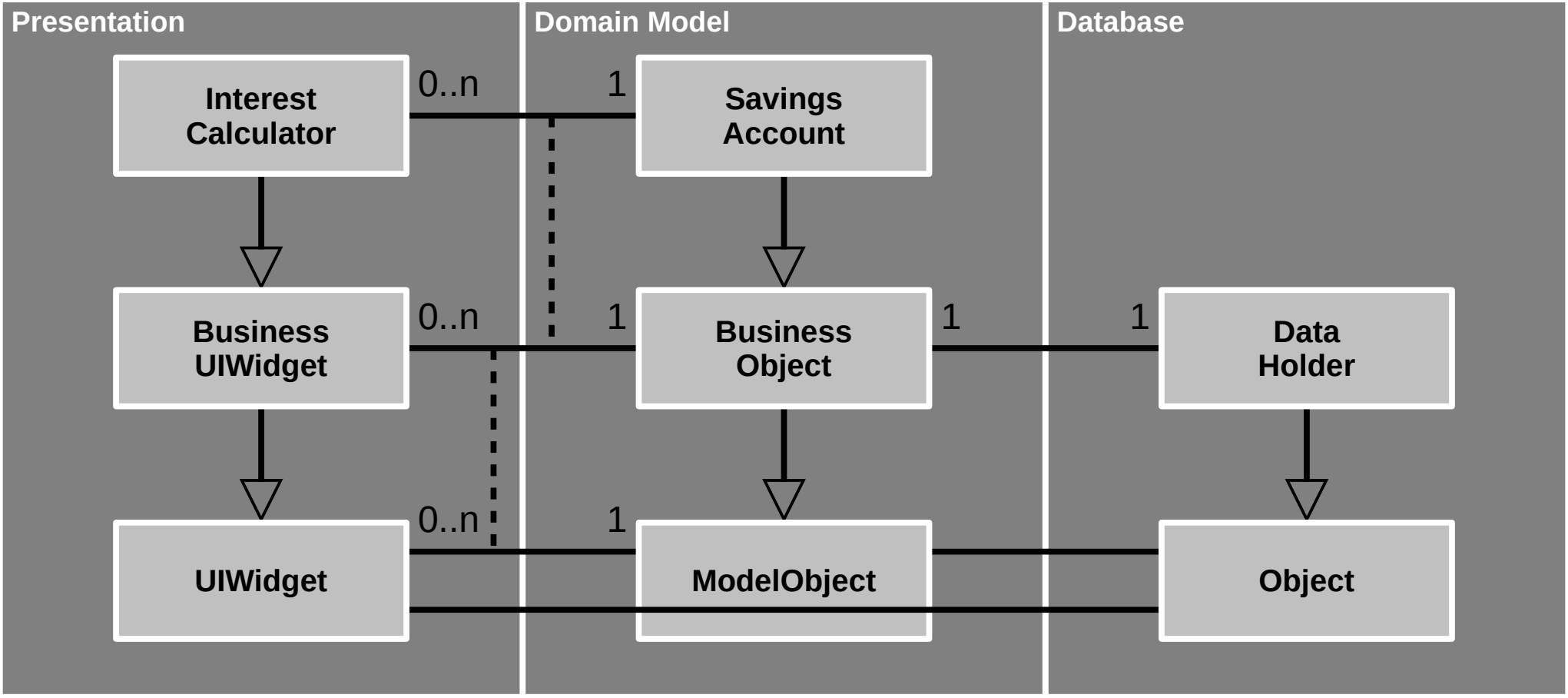
Run-time Components

- One or more run-time entities (objects, data + functions) grouped into an entity
  - With high cohesion and low coupling
- The boundary around the entities often only exists only in an architect's mind
  - May be captured as part of a system model, but gets resolved at run-time
- The boundary around the entities can be made more explicit though
  - Closures
  - Threads or agents
  - Processes
  - Containers
- Run-time components can be composed into larger run-time components

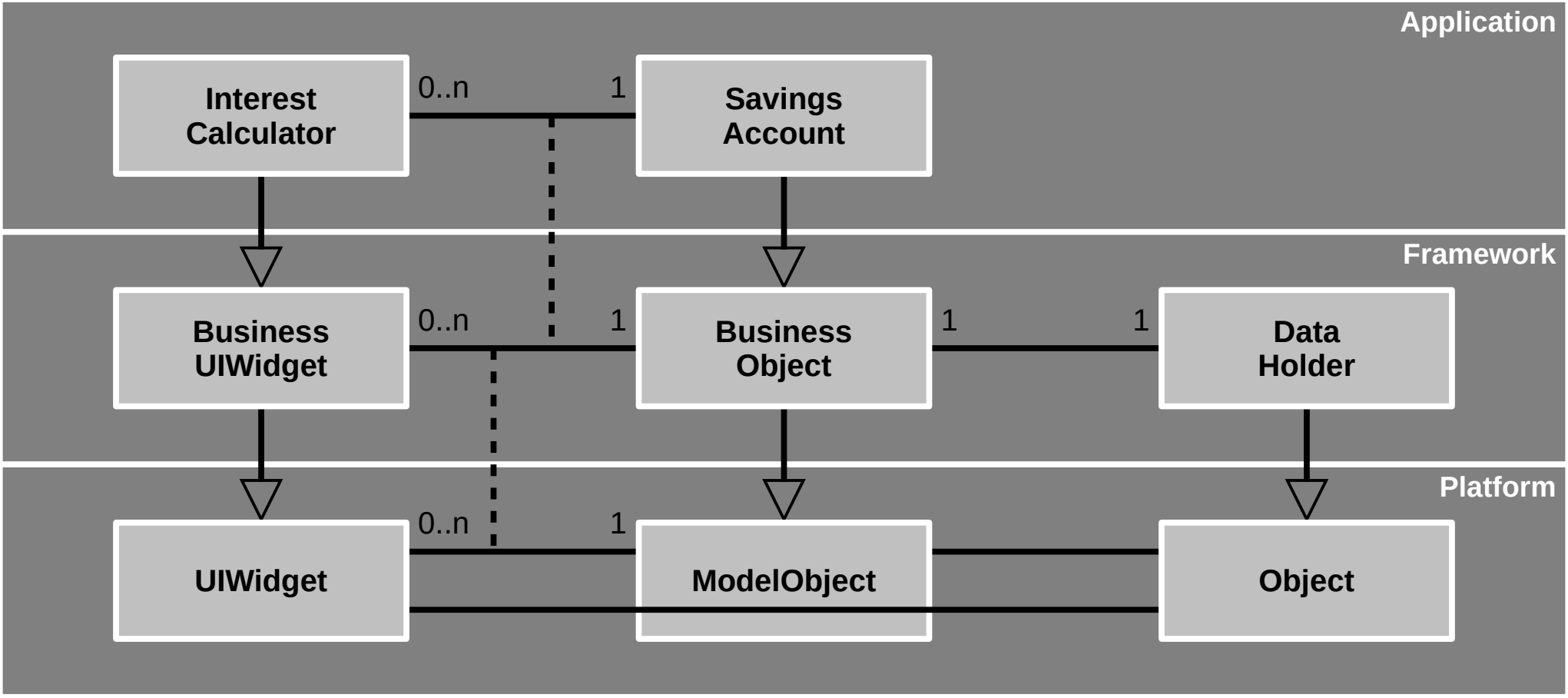
Component Example



Run-time Components



Code Components



Types of Code Components

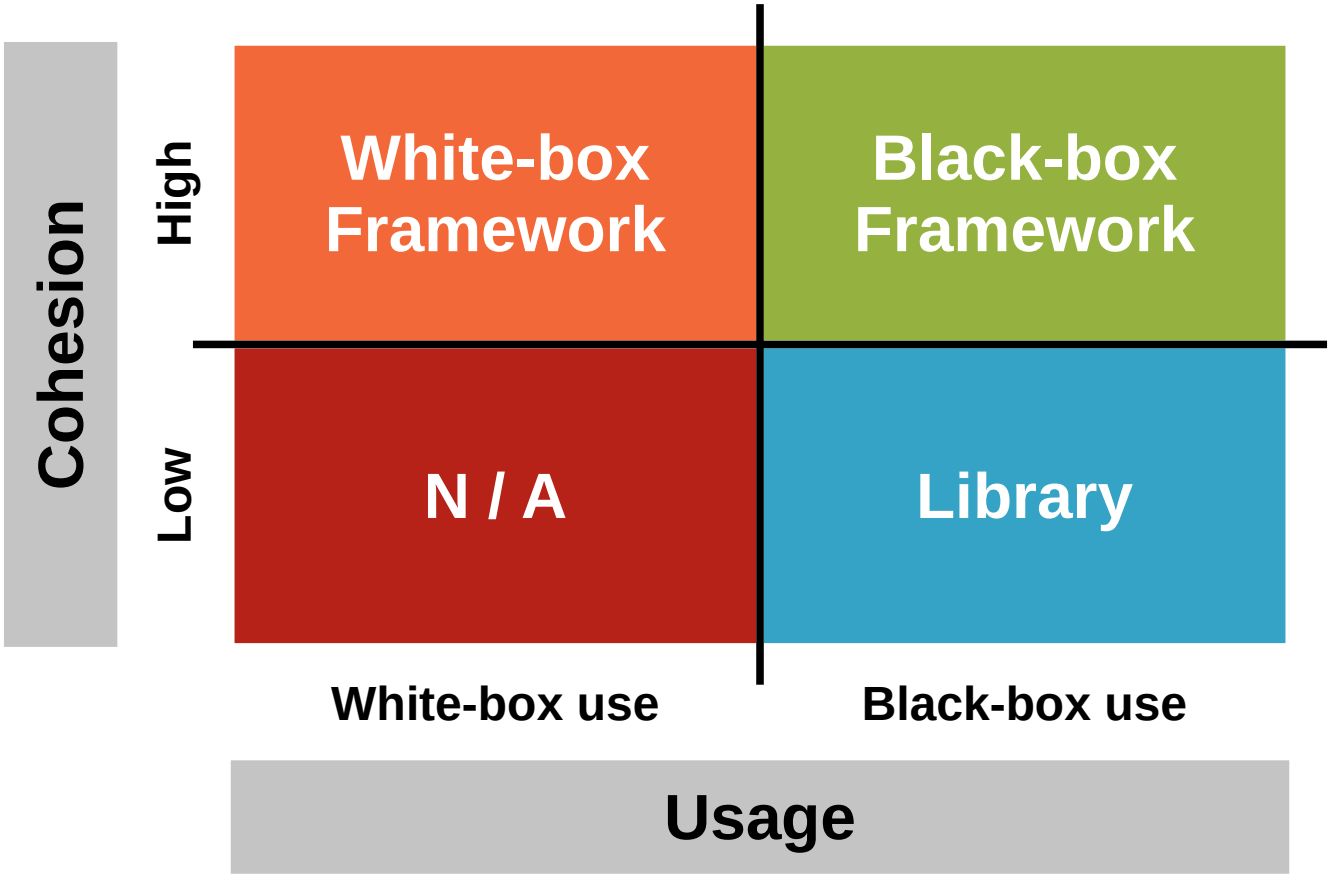
- **Libraries [1]**
- **Platforms**
- **Frameworks**

[1] A.k.a. toolkits

Object-Oriented Framework

- Definition of object-oriented framework
  - Is an abstract object-oriented design that can be reused
  - Has default implementation classes that can be used
  - Typically covers one particular technical domain
- White-box framework
  - An object-oriented framework mostly used by implementing subclasses
  - Requires user to understand internal workings of framework
  - Typically a framework in its early stages
- Black-box framework
  - An object-oriented framework mostly used by composing instances
  - Easier to use but may be less flexible than white-box framework
  - Typically a framework in its mature stages

Frameworks vs. Libraries 1 / 2



Frameworks vs. Libraries 2 / 2

- Frameworks
  - Provides abstract design
    - High cohesion of classes
    - Inheritance and delegation
      - Inheritance interface
    - More difficult to use than library
  - Examples
    - Java Object framework
    - Wahlzeit domain model
- Libraries
  - Provides no abstract design
    - Mostly loose class relationships
    - No or little use of inheritance
      - Only use-relationship
    - Easier to use than framework
  - Examples
    - Java utility classes
    - Wahlzeit utility classes

Framework Interfaces

1. **Use-client interface**
2. **Inheritance interface**
3. **Meta-object Protocol**

Use-Client Interface

- The use-client interface is the traditional interface
  - Invoked using method calls by client objects on framework objects
- Best practices of defining use-client interfaces
  - An abstract object-oriented design that reflects the domain
    - Using interfaces, abstract classes, and implementation classes
    - Using collaborations spelling out roles and their responsibilities
    - Using exceptions properly to document behavior in case of failure
  - With clear idea of types of objects, for example, value objects
  - With clear idea of patterns employed to structure the design

Inheritance Interface

- The inheritance interface uses polymorphism
  - Subclasses extend the design while conforming to it
  - Leads to inverted control-flow, a.k.a. “Hollywood principle”
- Best practices of defining inheritance interfaces
  - An abstract object-oriented design that reflects the domain
    - Using the abstract superclass rule
    - Using the narrow inheritance interface principle
  - With clear idea of patterns employed to structure the interface, e.g.
    - Primitive and composed methods
    - Factory method, template, method, etc.
      -
  - Document extension points

Inheritance Interfaces of the Wahlzeit Framework

- Main (startup and shutdown protocol)
- Model (photo, user, and case handling)
- Handlers (user functions and workflows)
- Agents (threaded non-user functions)
- …

Wahlzeit Framework with Flowers Extension

Framework

Extension

Main Inheritance Interface

```
public abstract class AbstractMain {
  protected void startUp(String rootDir) throws Exception { ... }
  protected void shutDown() throws Exception { ... }
  ...
}

public abstract class ModelMain extends AbstractMain {
  protected void startUp(String rootDir) throws Exception { ... }
  protected void shutDown() throws Exception { ... }
  ...
}

public class ServiceMain extends ModelMain {
  public void startUp(boolean inProduction, String rootDir) throws Exception { ... }
  public void shutDown() throws Exception { ... }
  ...
}

public abstract class ScriptMain extends ModelMain {
  public void run() { ... }
  ...
}
```

Meta-object Protocol

- Java annotations

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