Object-Oriented Frameworks

Dirk Riehle, FAU Erlangen

ADAP D05

Agenda

1. Code and runtime components
2. Object oriented frameworks
3. Use-client interface
4. Inheritance interface
5. Framework extensions
6. Meta-object protocol
7. Layers vs. tiers revisited

1. Components

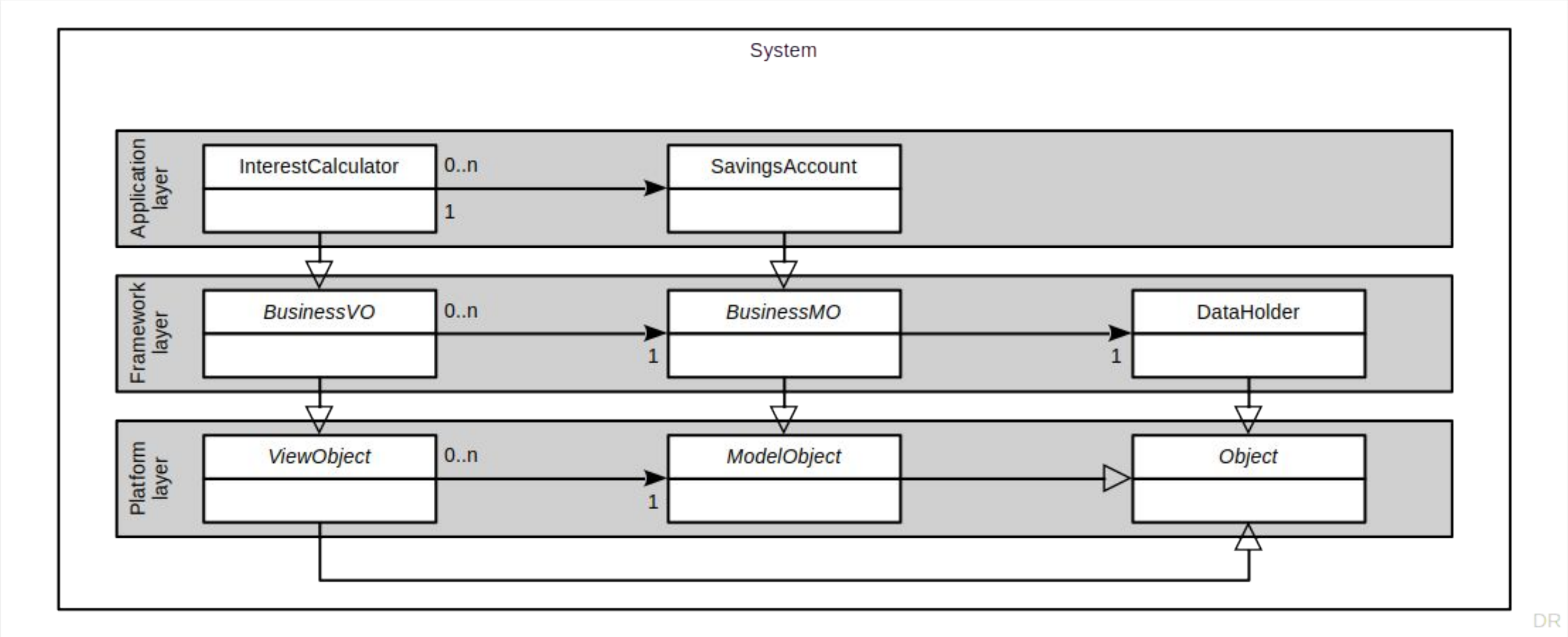
Components

- A component is some entity with a defined boundary; it should have
  - High internal cohesion and low external coupling
- Components can be composed from smaller components
  - Atomic components may be files (code) or functions (runtime)
- Code components are code
  - Source code in directories, compiled binaries, etc.
- Runtime component are data (with associated code)
  - May or may not map on code components
- You always either talk about code or runtime components

Code Components (Classes)

- 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, etc.
- Source code is usually compiled into one binary
  - Not reused as source code (not even generics/templates)
- Code components can be aggregated
  - Used to be done mainly for binaries, not source code; is changing

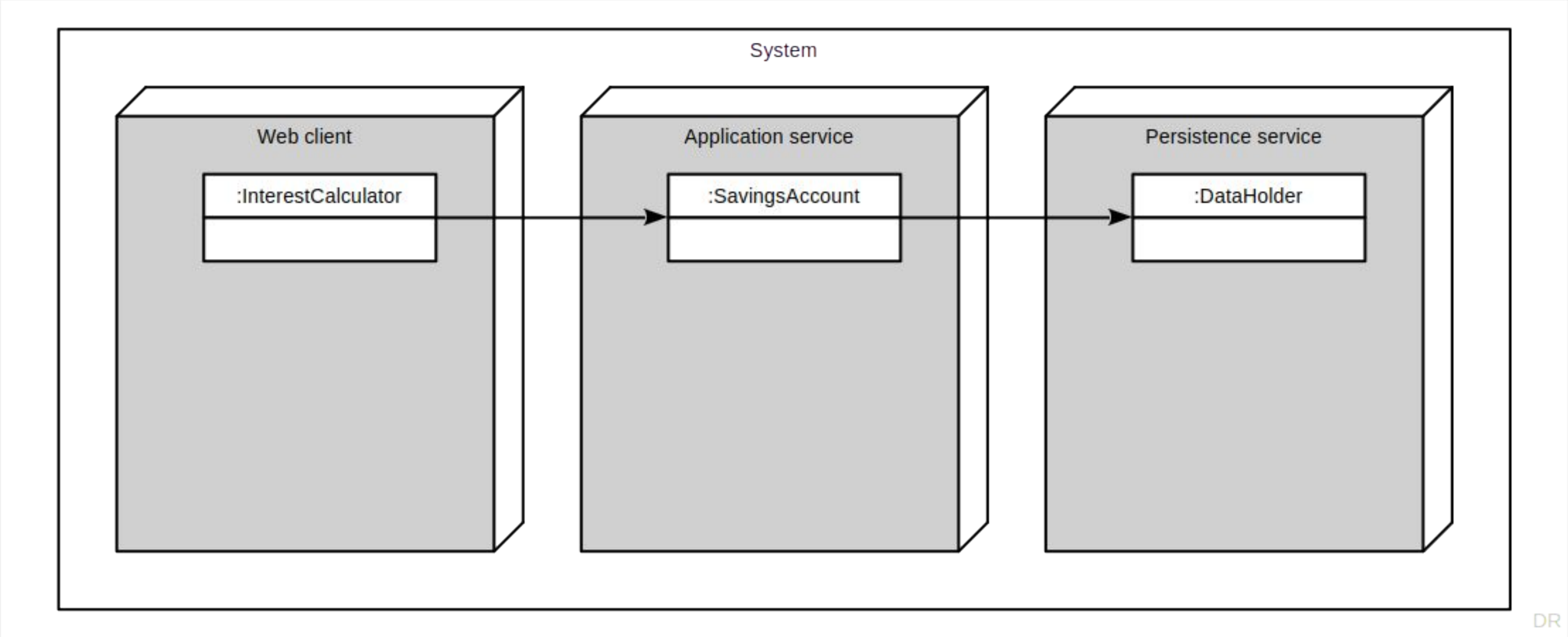
Code Components and Layers (Recap)



Runtime Components (Objects)

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

Runtime Components and Tiers (Recap)



Types of Code Components

Small code components

1. Functions
2. Classes

Larger code components

3. Libraries (also toolkits)
4. Frameworks
5. Platforms

2. Object-Oriented Frameworks

Object-Oriented Framework

An 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 application domain

White-Box Framework

A white-box framework is an object-oriented framework that

- Is mostly used (applied) by implementing subclasses
- Requires the users to understand how it works internally
- Is often still young and immature i.e. developing rapidly

Black-Box Framework

A black-box framework is an object-oriented framework that

- Is mostly applied (used) by composing instances
- Is typically easier to use than a white-box framework
- Is typically in its mature stages (all code is there)

Frameworks vs. Libraries / Toolkits 1 / 2

A framework has high cohesion, a library has low cohesion

A framework may be both white-box or black-box, a library only black-box

Frameworks vs. Libraries 2 / 2

Frameworks

- Provide an abstract design
  - High cohesion of classes
  - Inheritance and delegation
  - Inheritance interfaces
- More difficult to use

Libraries

- Provide no abstract design
  - Mostly individual classes
  - No or little use of inheritance
  - Only use relationships
- Easier to use

Framework Interfaces

1. Use-client interface
2. Inheritance interface
3. Meta-object protocol

3. Use-Client Interface

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 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

4. Inheritance Interface

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 how to structure the interface
    - Primitive and composed methods
    - Factory method, template, method, etc.
  - Document extension points

Happy Go Lucky Inheritance Interfaces

To be done

5. Framework Extensions

Framework Extension

A (framework) extension point (class) is

- A framework class and its inheritance interface intended to be subclassed

A framework extension is

- A set of classes created to apply (use) the framework in its domain
- Always a white-box use of the framework

Framework and framework extension are separate code components

Happy Go Lucky Framework Extensions

To be done

6. Meta-Object Protocols

Meta-Object Protocol

A meta object is

- An object describing another object (the base object)

A meta-object protocol (MOP) is

- A use-client interface to the meta objects of a system

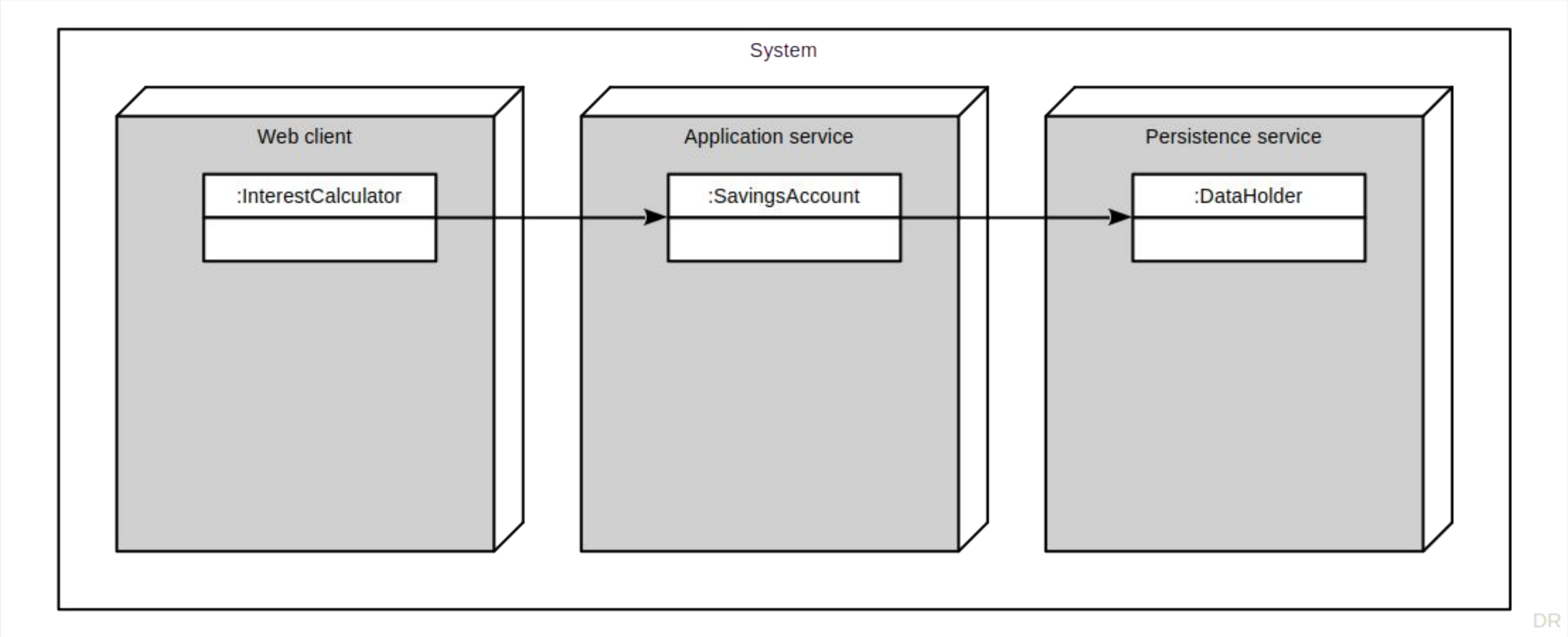
The most common MOP is a language’s reflection API

Happy Go Lucky Meta-Object Protocol

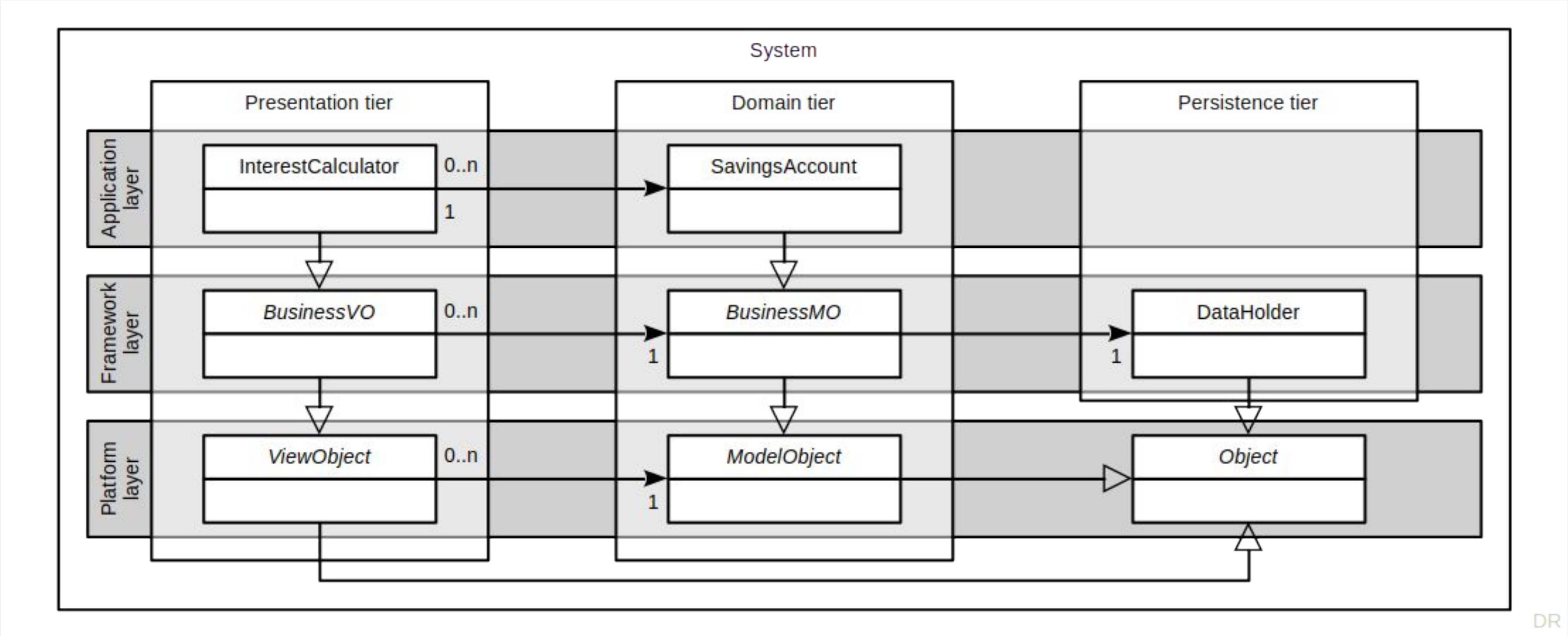
To be done

7. Layers vs. Tiers

Runtime Objects in a Three-Tier System



The Code Structure of a Three-Tier System



Layers as Stacking Mechanism

Dependencies go from higher to lower code layer

- Explicit method calls go from higher to lower layer
- Control flow either returns normally or by way of callbacks
- Strict stacking allows only for dependencies on next lower level

Code component dependencies form a directed acyclic graph (DAG)

Tiers as Stacking Mechanism

A runtime component (i.e. an object)

- Always only exists within one tier
- Is aggregated from data across the code layers

Summary

1. Code and runtime components
2. Object oriented frameworks
3. Use-client interface
4. Inheritance interface
5. Framework extensions
6. Meta-object protocol
7. Layers vs. tiers revisited

Thank you! Any questions?

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

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

Legal Notices

License

Copyright