Prof. Dr. U. Aßmann
Chair for Software Engineering
Faculty of Computer Science
Dresden University of Technology
WS 18/19, November 1, 2018

Lecturer: Dr. Sebastian Götz

5. Architectural Glue Patterns

1) Mismatch Problems
2) Adapter Pattern
3) Facade
4) Mediator

Literature (To Be Read)

- D. Garlan, R. Allen, J. Ockerbloom. **Architectural mismatch – or why it is so hard to build systems out of existing parts**. In Proceedings of the 17th International Conference on Software Engineering, ACM, 1995, pp. 179-185 https://dl.acm.org/citation.cfm?doid=225014.225031
- D. Garlan, R. Allen, J. Ockerbloom. **Architectural Mismatch: Why Reuse is Still So Hard.** IEEE Software 26:4, July/August 2009, pp. 66-69. https://ieeexplore.ieee.org/abstract/document/5076461
- GOF – Adapter, Mediator, Facade

Goal

- Understand architectural mismatch
- Understand design patterns that bridge architectural mismatch

5.1 Architectural Mismatch

Roots of Architectural Mismatch

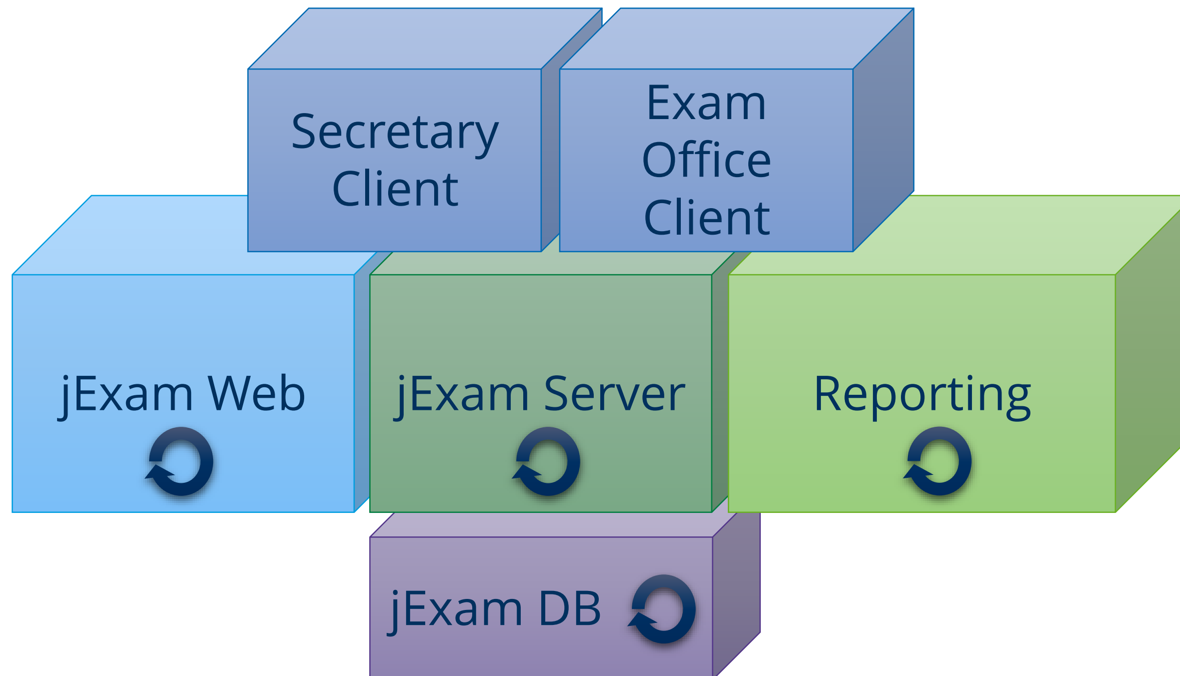
- Different Assumptions about the *component model*
  - Infrastructure
  - Control model
  - Data model
- Different assumptions about the *connectors*
  - Protocols
  - Data models
- Different assumptions about the *global architectural structure*
- Different assumptions about the *construction process*

Different Assumptions about the Component Model

- A component model assembles information and constraints about the nature of components
  - Nature of interfaces
  - Substitutability of components
- Components assume they have a certain **infrastructure**, but it might not be available
  - For example, one component assumes an Windows infrastructure, while another assumes Linux
- More in “Component-Based Software Engineering”, summer semester

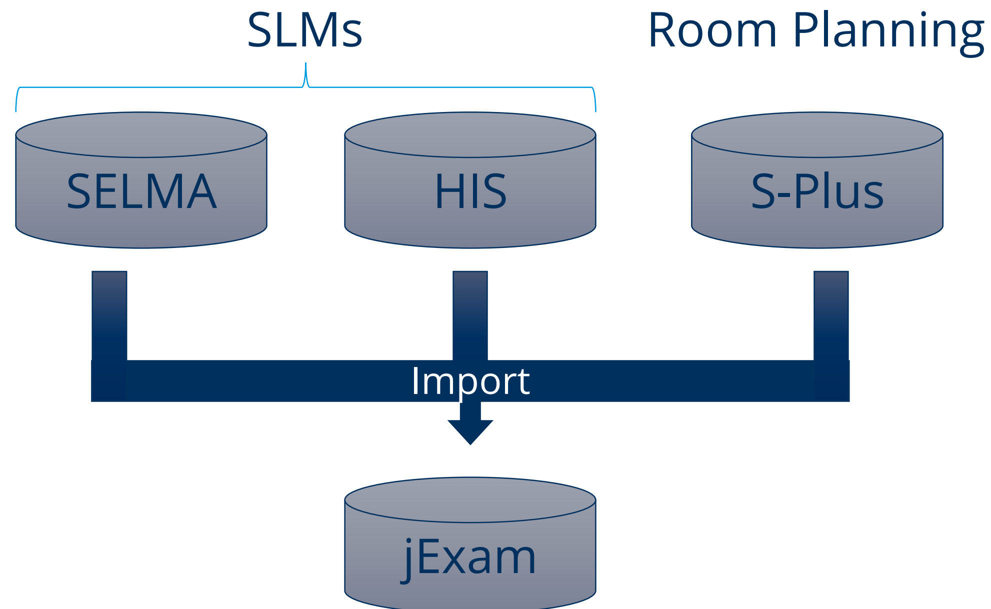
Assumptions on Control Model

- Components think differently in which components have the main control
  - Multiple components might each have an ever-running event loop inside



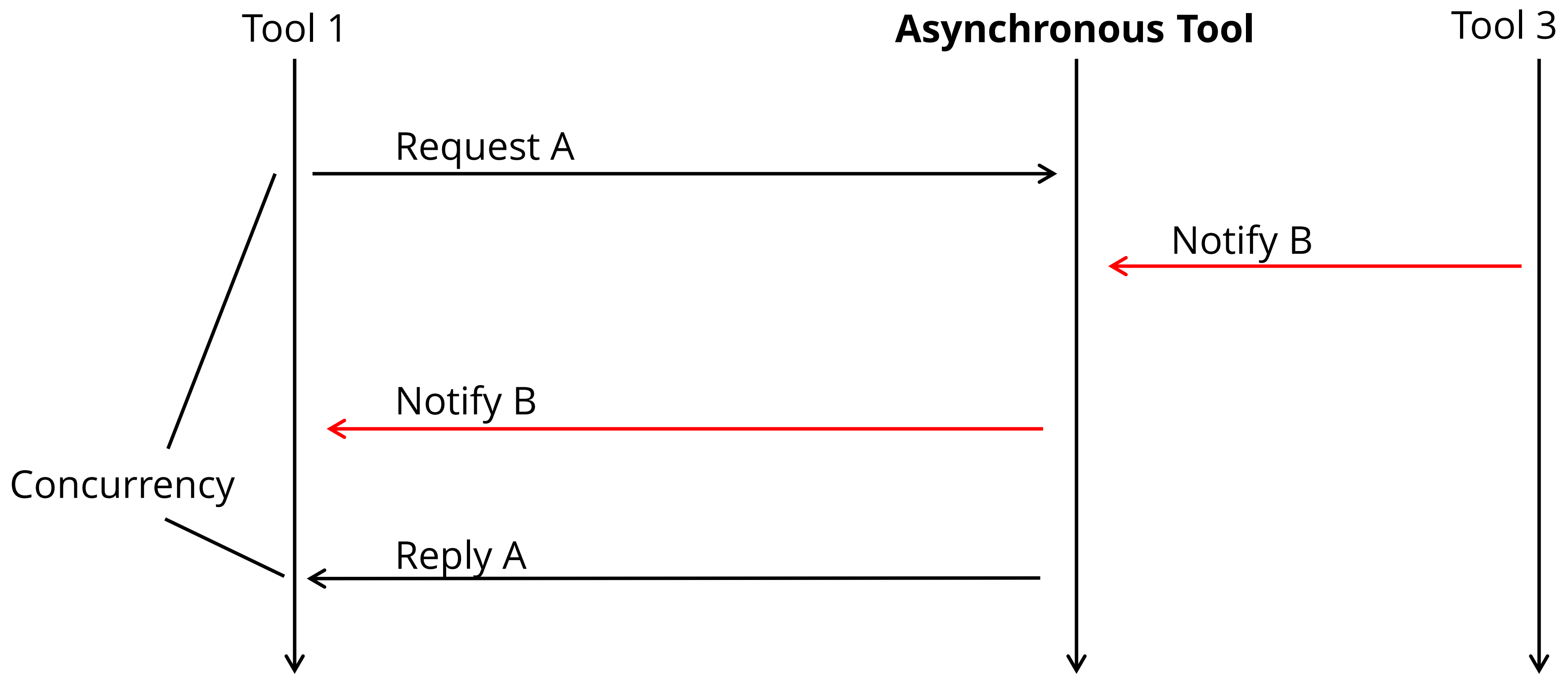
Assumptions on Data Model

- Different assumptions about the data



Connectors: Protocol Mismatch

Some tools work asynchronously; which superimposes concurrency to tools, when messages of different tools are crossing



Data Format Mismatch

- Components also have different assumptions what comes over a channel (a connection).
  - Strings
  - C data
  - C++ data
- Requires translation components
  - This can easily become a performance bottleneck

Assumptions about the Global Architecture

- For example,
  - a **database-centered** architecture (repository style) versus
  - A **shared-nothing** architecture

Assumptions about the Building Process

- Assumptions about the library infrastructure
- Assumptions about a generic language (C++)
- Assumptions about a tool specific language
- Combination is fatal:
  - Some component A may have other expectations on the generated code of another component B as B itself
  - Then, the developer has to patch the generated code of A with patch scripts (another translation component)

Proposed Solutions of [Garlan et al. 1995]

- Make *all* architectural assumptions explicit
  - Problem: how to document or specify them?
  - Many of the aforementioned problems are not formalized
  - Implicit assumptions are a violation of the information hiding principle, and hamper variability
- Make components more independent of each other
- Provide bridging technology
  - For building language translation components (compiler construction, compiler generators, XML technology)
- Distinguish architectural styles (architectural patterns) explicitly
  - Distinguish connectors explicitly
- Solution: design patterns serve all of these purposes

Usability of Extensibility Patterns

- All extensibility patterns can be used to treat architectural mismatch
- Behavior adaptation
  - **ChainOfResponsibility** as filter for objects, to adapt behavior
  - **Proxy** for translation between data formats
  - **Observer** for additional behavior extension, listening to the events of the subject
  - **Visitor** for extension of a data structure hierarchy with new algorithms
- Bridging data mismatch
  - **Decorator** for wrapping, to adapt behavior, and to bridge data mismatch, not for protocol mismatch
  - **Bridge** for factoring designs on different platforms (making abstraction and implementation components independent)

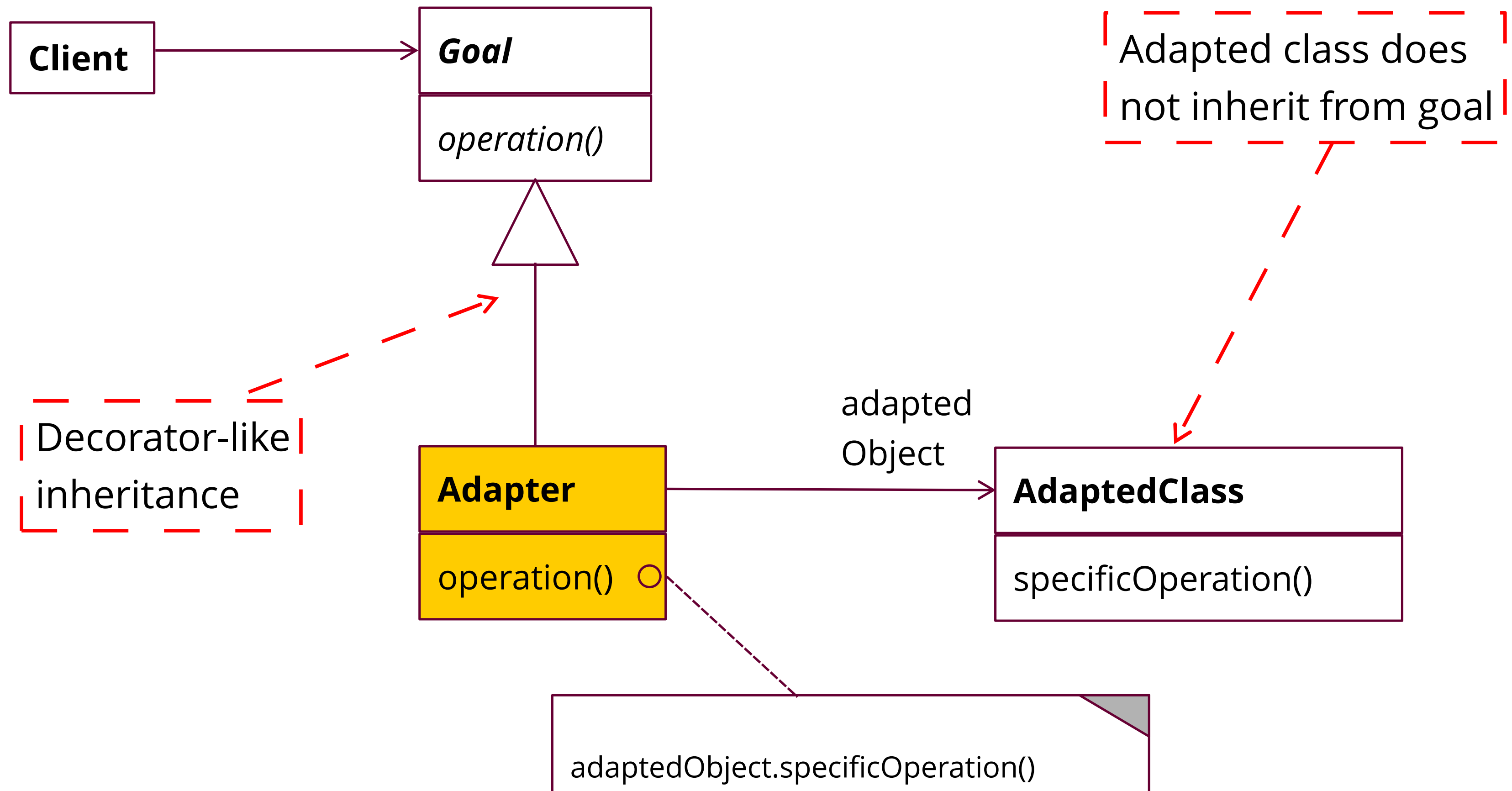
5.2 Adapter

Object Adapter

- An object adapter is a proxy that maps one interface to another
  - Or a protocol
  - Or a data format
- An adapter cannot easily map control flow to each other
  - Since it is passed *once* when entering the adapted class

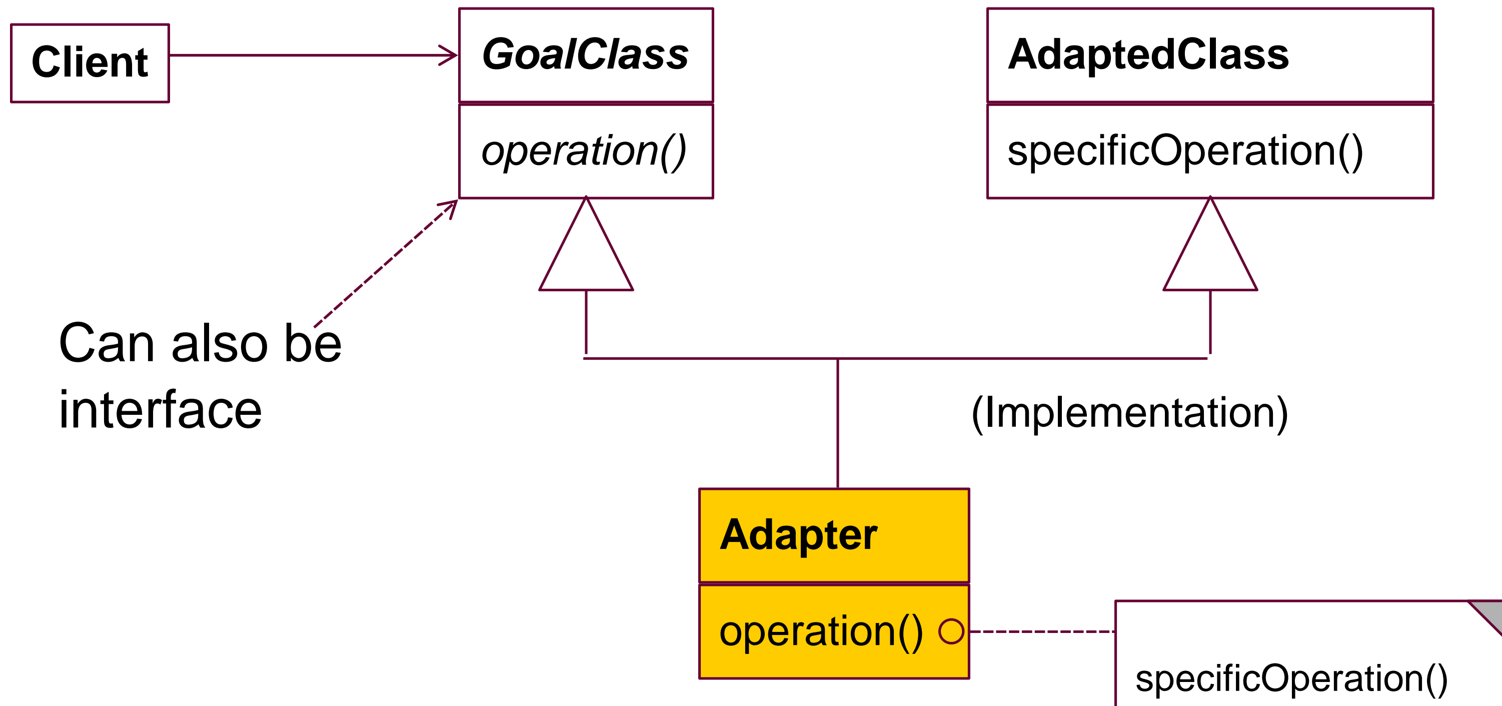
Object Adapter

Object adapters use delegation

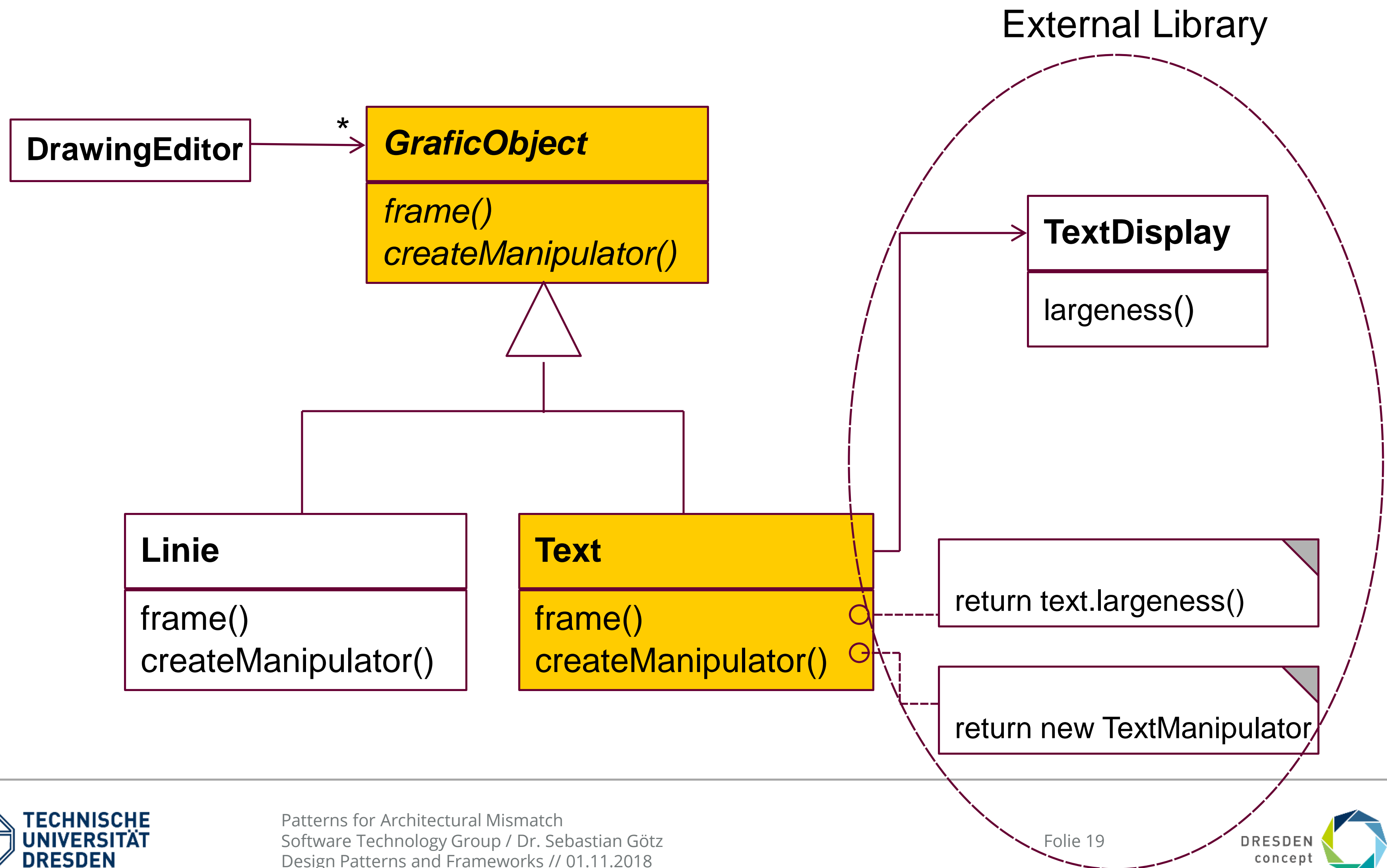


Class Adapter

Instead of delegation, class adapters use multiple inheritance

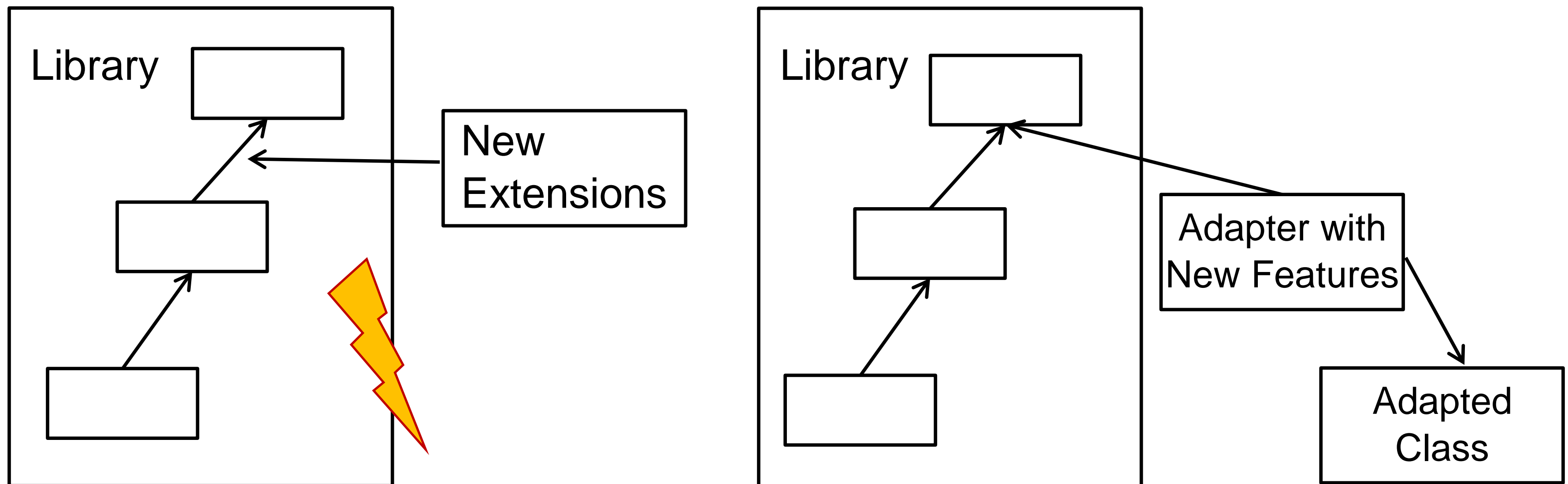


Example: Use of Legacy Systems: Using External Class Library For Texts



Adapters and Decorators

- Similar to a decorator, an adapter inherits its interface from the goal class
  - but adapts the interface
- Hence, adapters can be *inserted* into inheritance hierarchies later on

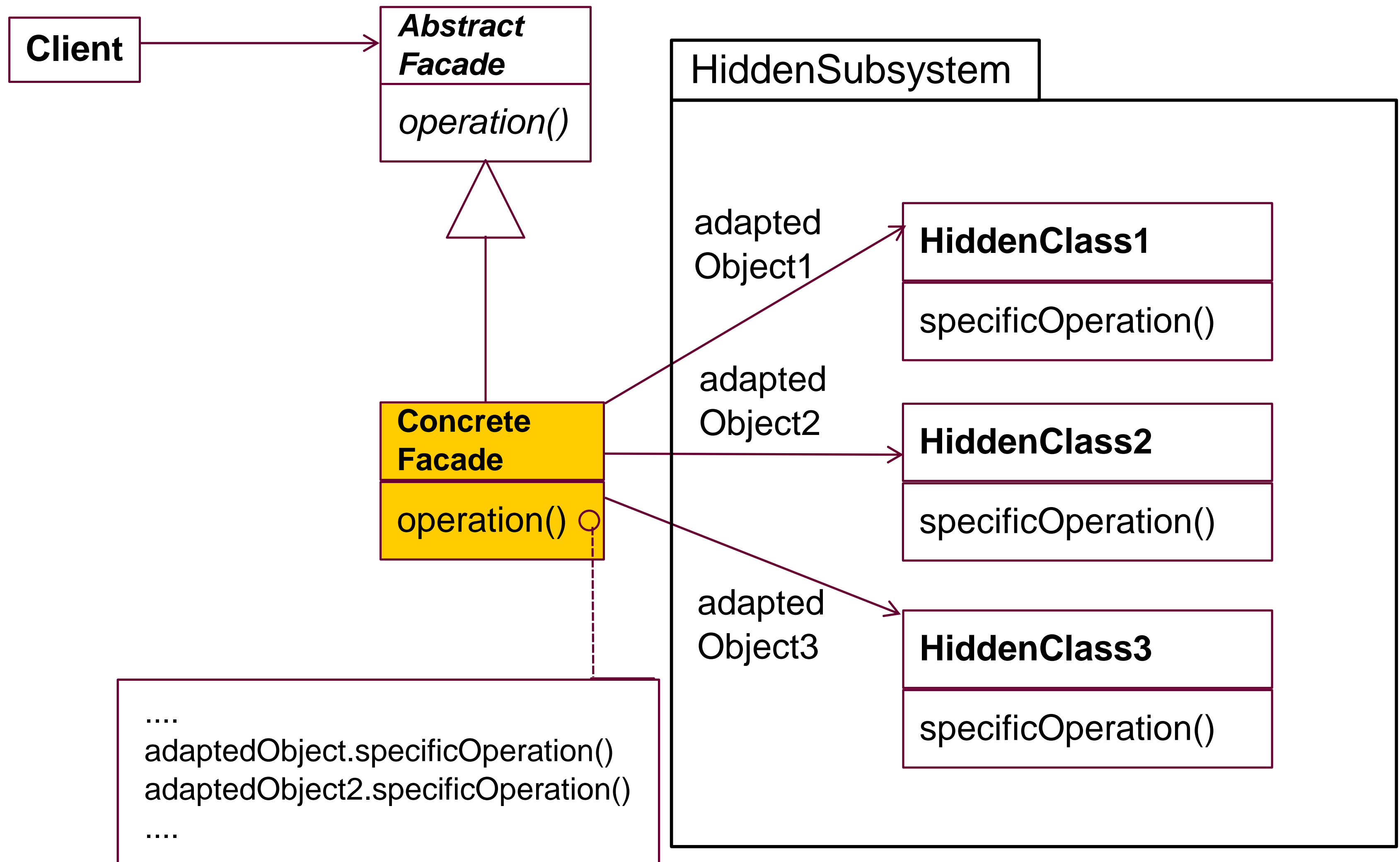


5.3 Facade

Facade

- A **facade** is an object adapter that hides a complete set of objects (subsystem)
- Or: a proxy that hides a subsystem
- The facade has to map its own interface to the interfaces of the hidden objects

Facade Hides a Subsystem

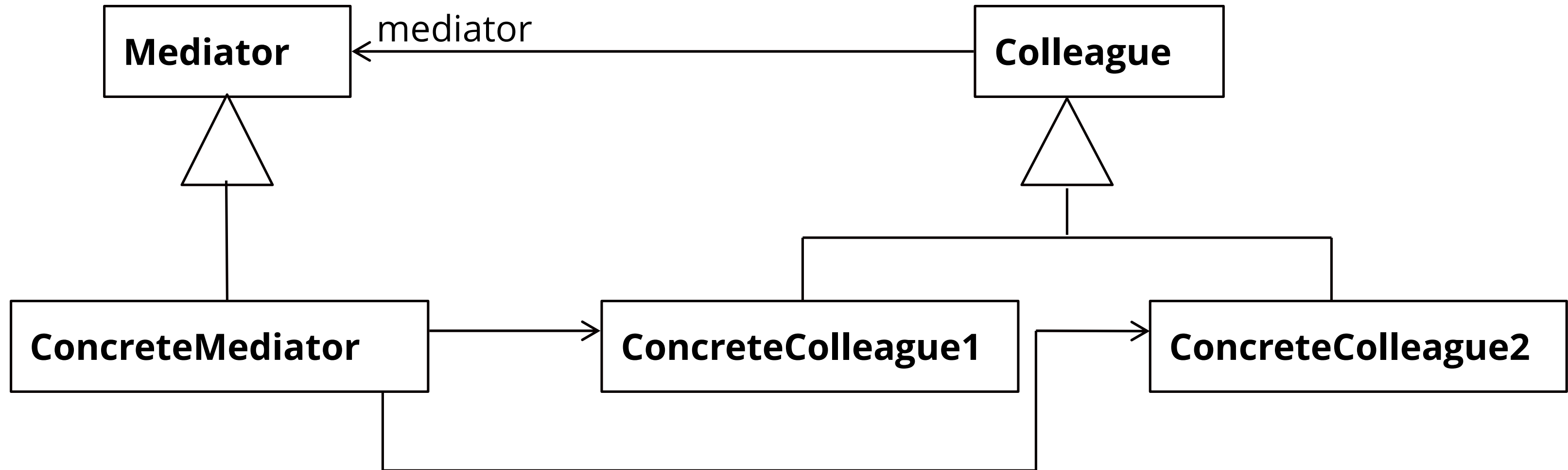


5.4 Mediator (Broker)

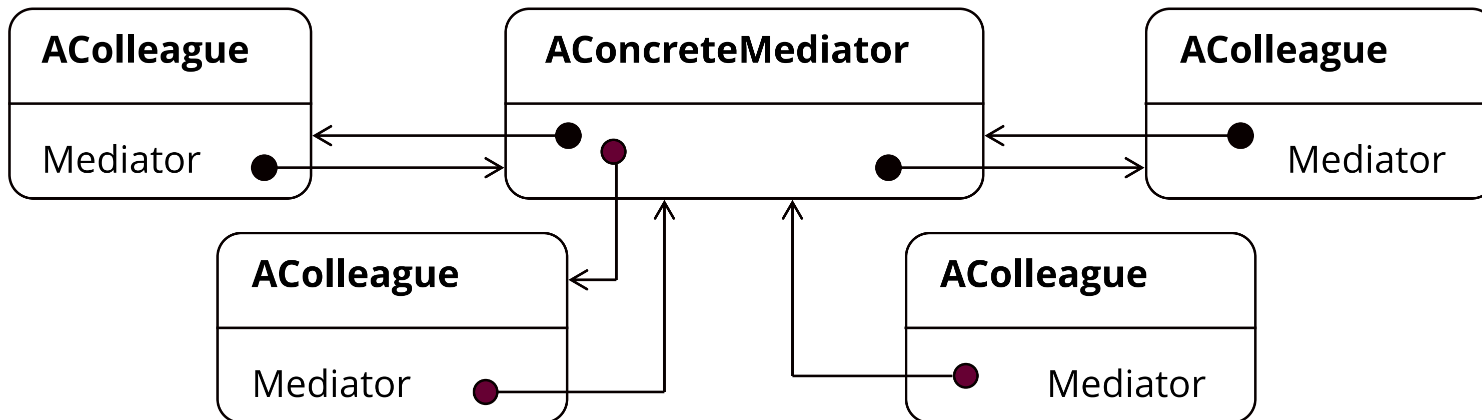
Mediator (Broker)

- A mediator is an n-way proxy for communication
  - Combined with a Bridge
- A mediator serves for
  - *Anonymous* communication
  - *Dynamic* communication nets

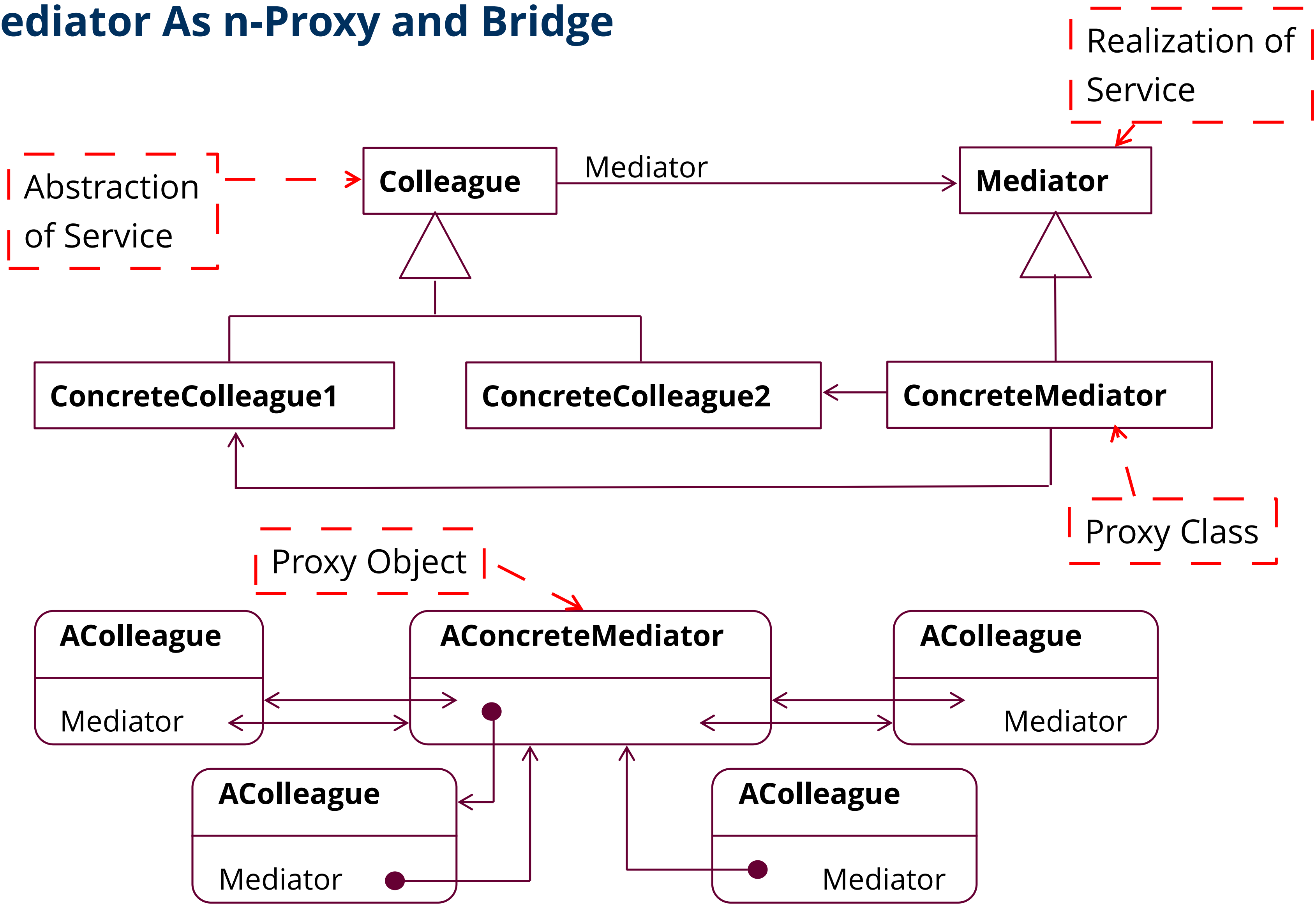
Mediator



Typical Object Structure:



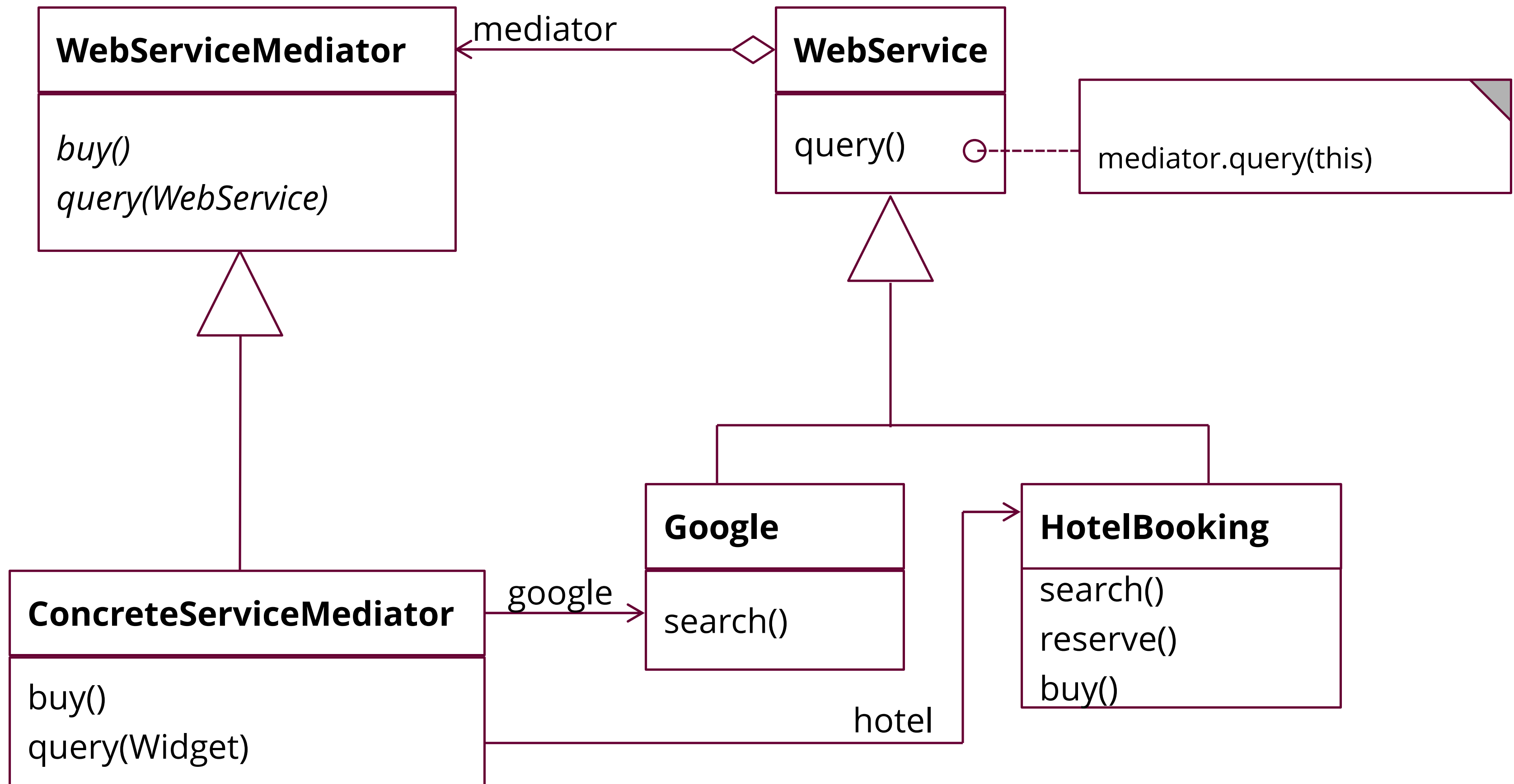
Mediator As n-Proxy and Bridge



Intent of Mediator

- Proxy object hides all communication partners
  - Every partner uses the mediator object as proxy
  - Clear: real partner is hidden
- Bridge links both communication partners
  - Both mediator and partner hierarchies can be varied
- ObserverWithChangeManager combines Observer with Mediator

Web Service Brokers



Summary

- Architectural mismatch between components and tools consists of different **assumptions** about *components, connections, architecture, and building procedure*
- Design patterns, such as extensibility patterns or communication patterns, can bridge architectural mismatches
  - Data mismatch
  - Interface mismatch
  - Protocol mismatch
- With Glue Patterns, reuse becomes much better

The End