5. Architectural Glue Patterns

Prof. Dr. U. Aßmann
Chair for Software Engineering
Faculty of Computer Science
Dresden University of Technology

WS 17/18, November 13, 2017

Lecturer: Dr. Sebastian Götz

1) Mismatch Problems
2) Adapter Pattern
3) Facade
4) Some variants of Adapter
5) Adapter Layers
6) Mediator
7) Repository Connector

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.** Int. Conf. on Software Engineering (ICSE'95) http://repository.cmu.edu/cgi/viewcontent.cgi?article=1714&context=compsci
- D. Garlan, R. Allen, J. Ockerbloom. **Architectural Mismatch: Why Reuse is Still So Hard.** IEEE Software 26:4, July/August 2009, pp. 66-69.
- GOF – Adapter, Mediator, Facade

References

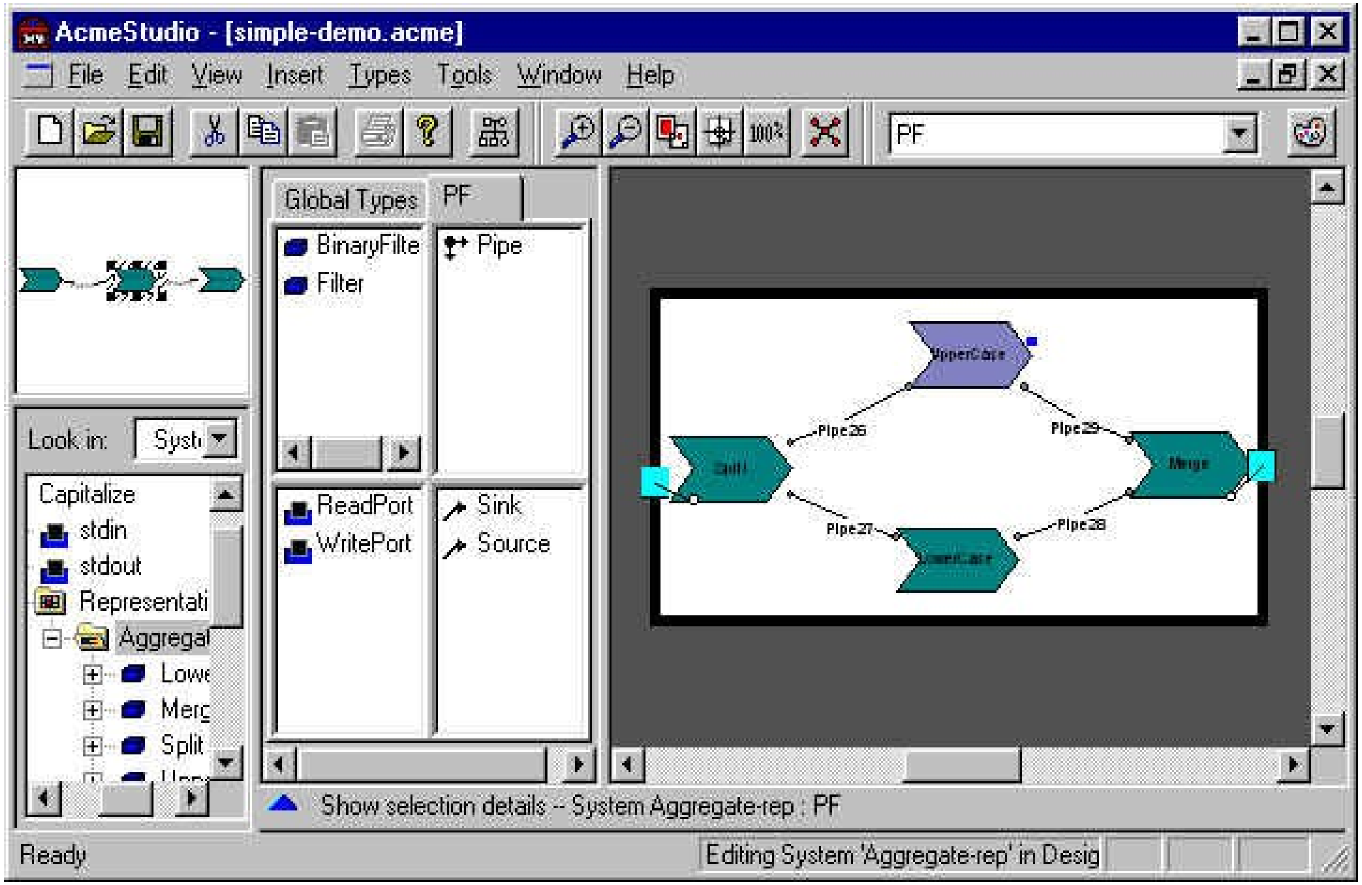
- The C++ main memory database OBST from Karlsruhe
  - **OBST Tutorial** http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.4966&rep=rep1&type=pdf
  - **OBST Overview** http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.2746&rep=rep1&type=pdf

Goal

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

Architectural Mismatch

- Case study of Garlan, Allen, Ockerbloom 1995
- Building the architectural system Aesop



AcmeStudio - [simple-demo.acme]

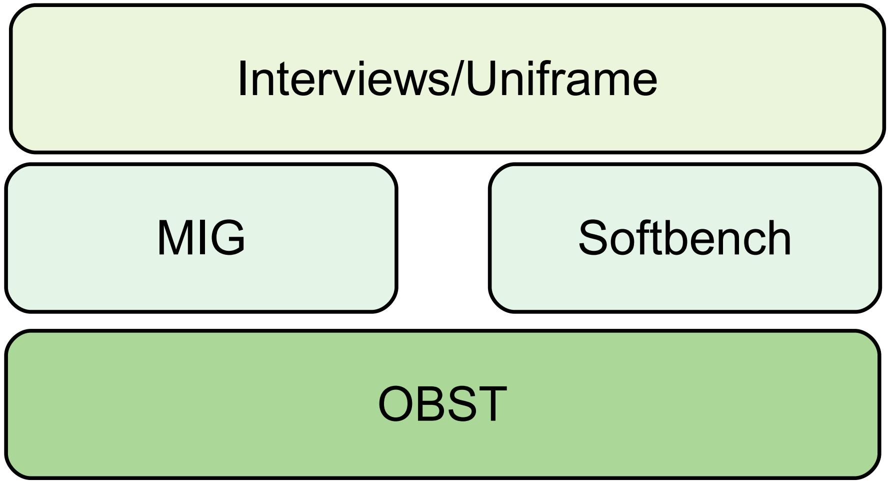
Show selection details -- System Aggregate-rep : PF

Ready

Editing System 'Aggregate-rep' in Desig

Architectural Mismatch

- Aesop was built out of 4 off-the-shelf components
  - OBST: an object-oriented C++ database
  - Interviews and Uniframe, a windowing toolkit
  - Softbench, an event bus (event-based mediator)
  - RPC interface generator of Mach (MIG)
- All subsystems written in C++ or C
- First version took 5 person years, and was still sluggish, very large
- Problems can be characterized in terms of components and connections



Interviews/Uniframe

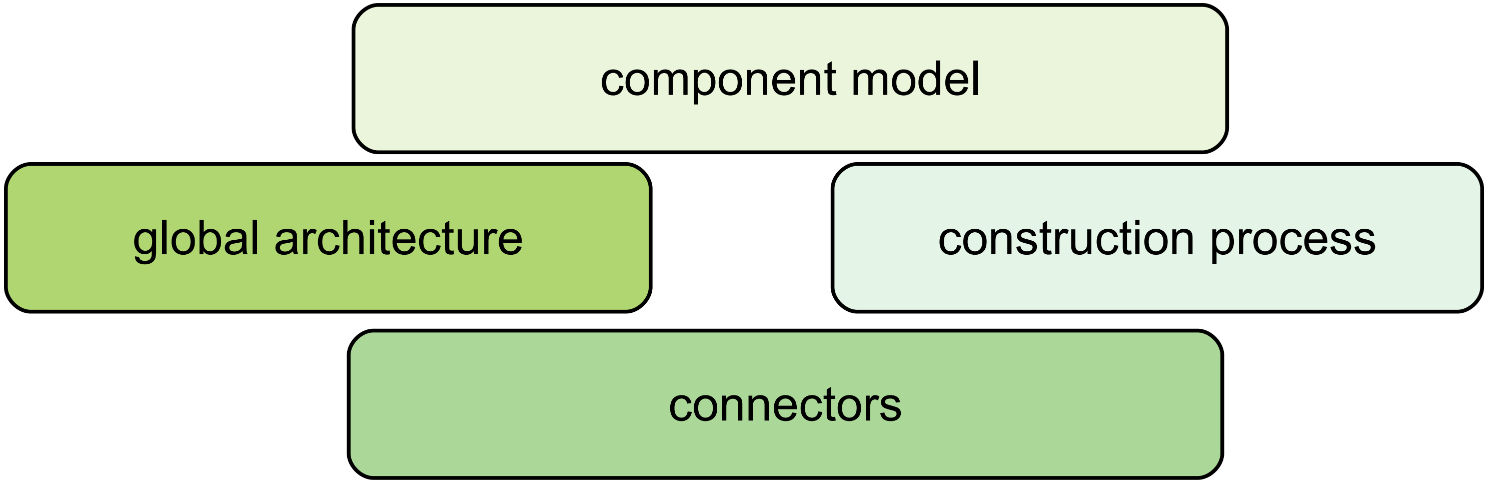
MIG

Softbench

OBST

Classification of Different Assumptions of the COTS

- 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*



component model

global architecture

construction process

connectors

Different Assumptions about the Component Model

- A component model assembles information and constraints about the nature of components
  - Nature of interfaces
  - Substitutability of components
- Here: **Component Infrastructure, Control model, Data model**
- Different Assumptions about the Component Infrastructure:
  - Components assume that they should provide a certain infrastructure, which the application does not need
  - OBST provides many library functions for application classes; Aesop needed only a fraction of those
- Components assume they have a certain infrastructure, but it is not available
  - Softbench assumed that all other components have access to an X window server (for communication)
- More in “Component-Based Software Engineering”, summer semester

Assumptions on Control Model

- COTS think differently in which components have the main control
  - Softbench, Interviews, and MIG have an ever-running event loop inside
  - They call applications with callbacks (observer pattern)
- However, they use different event loops:
  - Softbench uses X window event loop
  - MIG and Interviews have their own ones
  - The event loops had to be reengineered, to fit to each other

Assumptions on Data Model

- Different assumptions about the data
  - Uniframe: hierarchical data model
  - Manipulations only on a parent, never on a child
  - However, the application needed that
  - Decision: rebuild the data model from scratch, is cheaper than modification

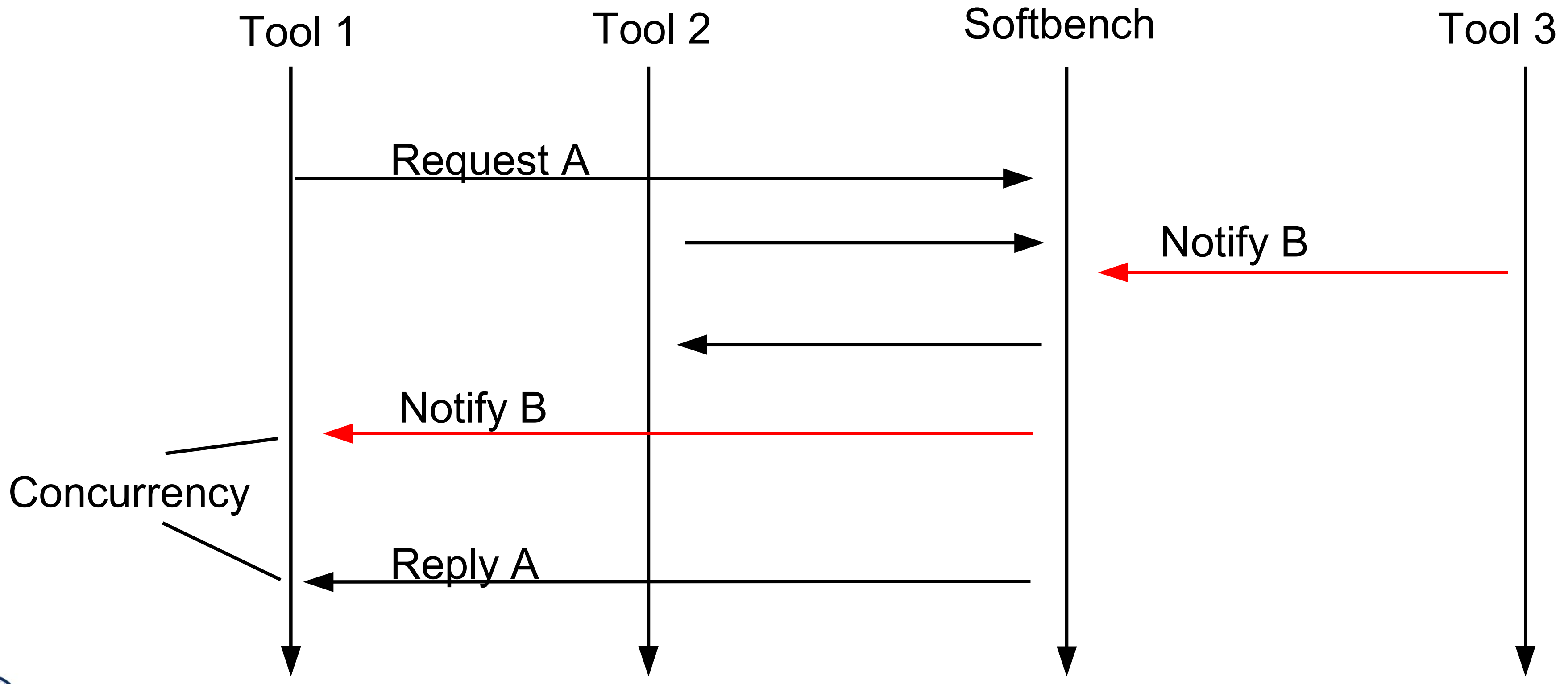
Assumptions about the Connectors

Protocol Mismatch

- Softbench works asynchronously; which superimposes concurrency to tools
  - Softbench is a mediator between tools
- 2 kinds of interaction protocols
  - Request/Reply (callback, observer): tool requests a service, registers a callback routine, is called back by Softbench
  - Notify via Softbench

Protocol Mismatch

- Softbench works 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).
  - Softbench: Strings
  - MIG: C data
  - OBST: C++ data
- Requires translation components
  - When accessing OBST, data must be translated all the time
  - This became a performance bottleneck

Assumptions about the Global Architecture

- OBST
  - Assumes a database-centered architecture (Repository Style)
  - Assumes independence of client tools
  - And provides a transaction protocol per single tool, not per combination of tools
  - Doesn't help when tools have interactions

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]

- 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)

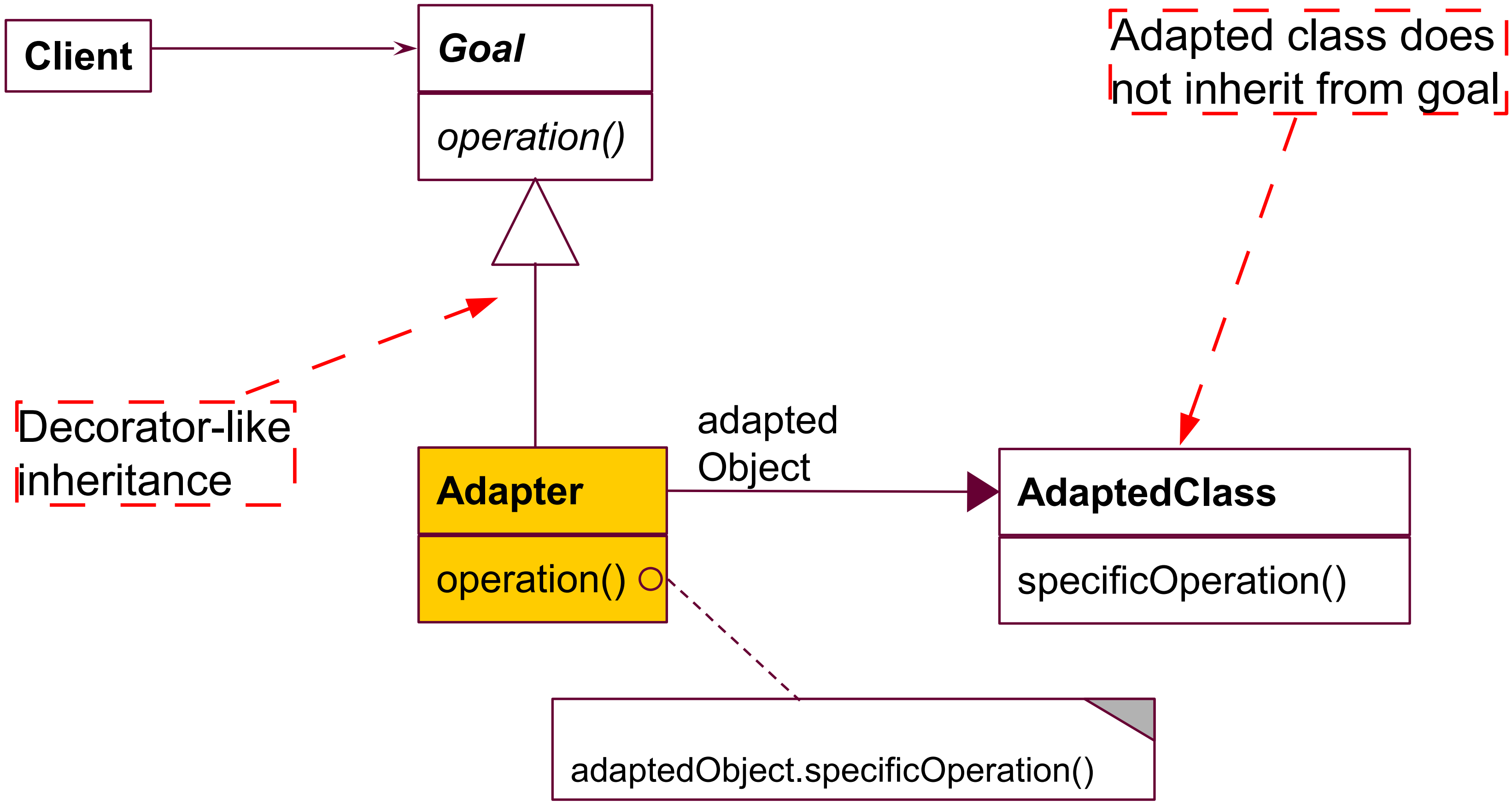
5.2 Adapter

Object Adapter

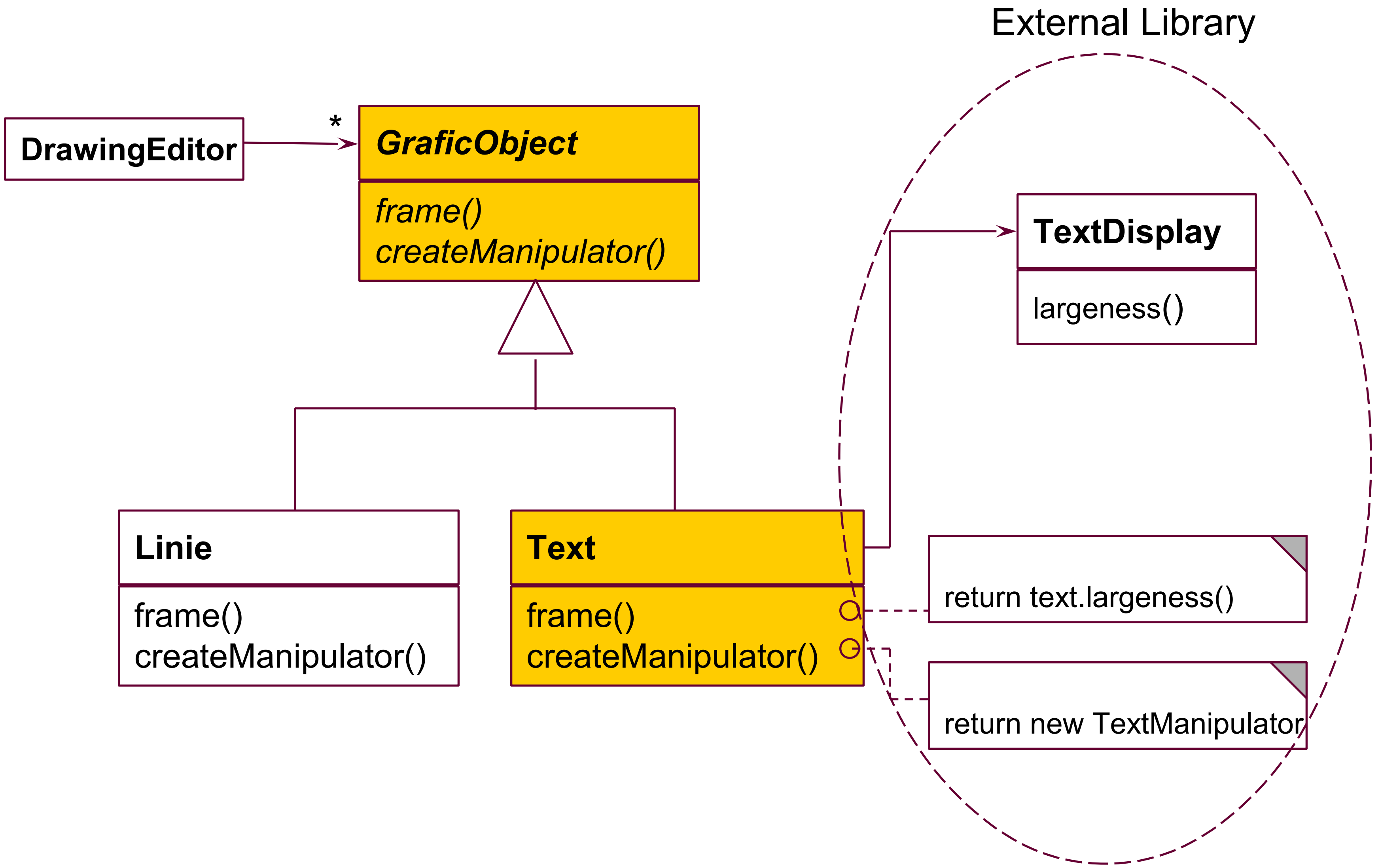
- An object adapter is a proxy that maps one interface to another
  - Or a protocol
  - Or a data format
- An adapter cannot easily map control flow to each other
  - Since it is passed *once* when entering the adapted class

Object Adapter

- Object adapters use delegation

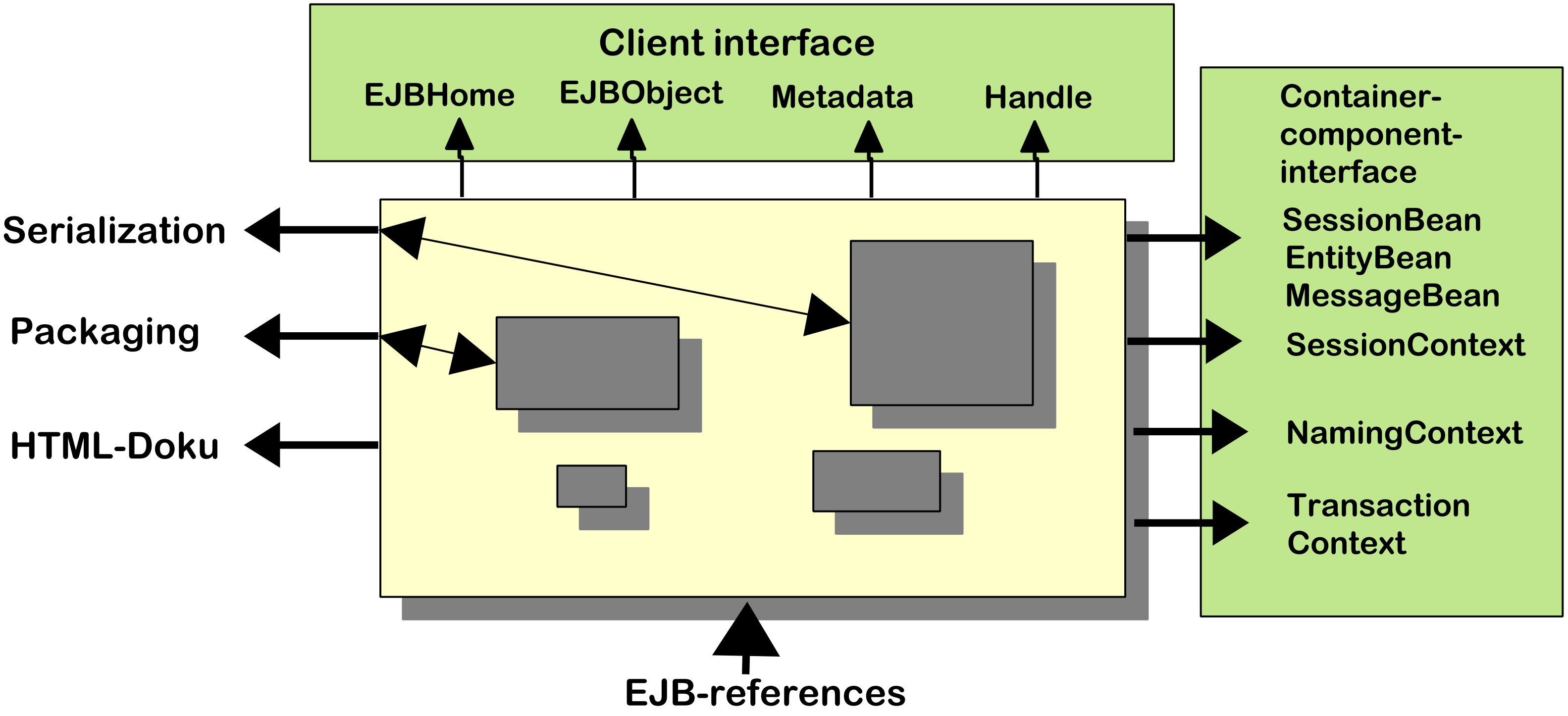

Client
Goal
operation()
Adapted class does not inherit from goal
Decorator-like inheritance
Adapter
operation()
adapted Object
AdaptedClass
specificOperation()
adaptedObject.specificOperation()

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

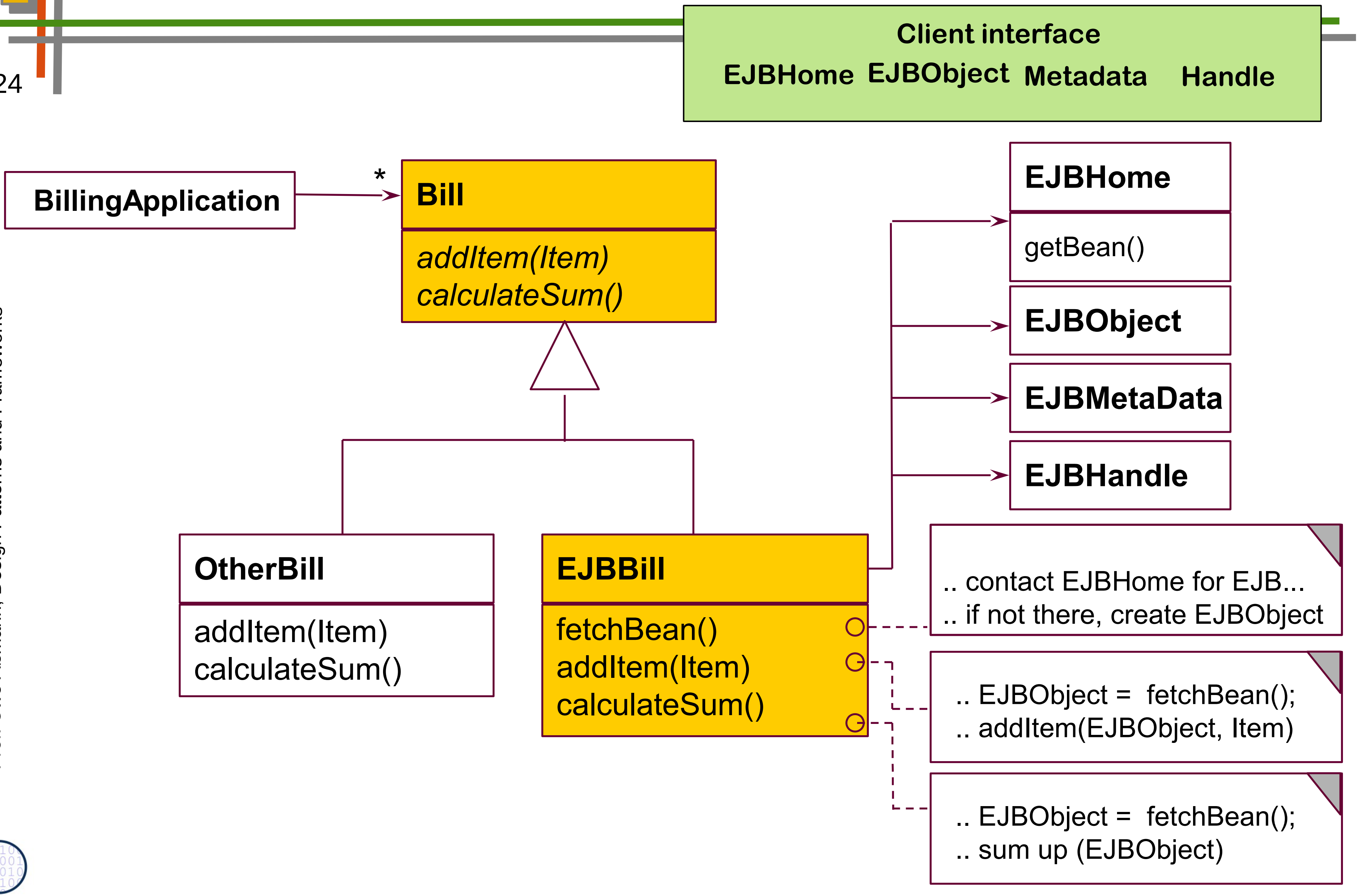


Adapters for COTS

- Adapters are often used to adapt components-off-the-shelf (COTS) to applications
- For instance, an EJB-adapter allows for reuse of an Enterprise Java Bean in an application



EJB Adapter

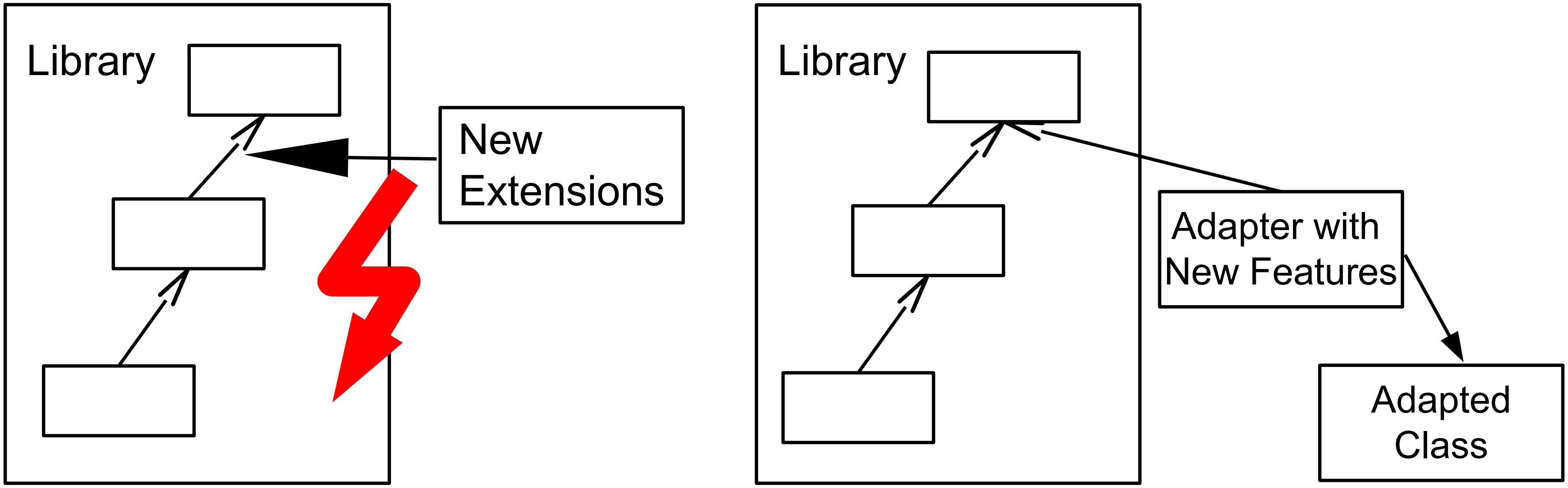

Client interface
EJBHome EJBObject Metadata Handle
BillingApplication
*
Bill
addItem(Item)
calculateSum()
OtherBill
addItem(Item)
calculateSum()
EJBBill
fetchBean()
addItem(Item)
calculateSum()
EJBHome
getBean()
EJBObject
EJBMetaData
EJBHandle
.. contact EJBHome for EJB...
.. if not there, create EJBObject
.. EJBObject = fetchBean();
.. addItem(EJBObject, Item)
.. EJBObject = fetchBean();
.. sum up (EJBObject)

A Remark to Adapters in Component Systems

- Component models define *standard, unspecific* interfaces
  - E.g., EJBHome / EJBObject
- Classes usually define *application-specific* interfaces
- To increase reuse of classes, the Adapter pattern(s) can be used to map the application-specific class interfaces to the unspecific component interfaces
- Example:
  - In the UNIX shell, all components obey to the pipe-filter interfaces *stdin, stdout, stderr* (untyped channels or streams of bytes)
  - The functional parts of the components have to be *mapped* by some adapter to the unspecific component interfaces.

Adapters and Decorators

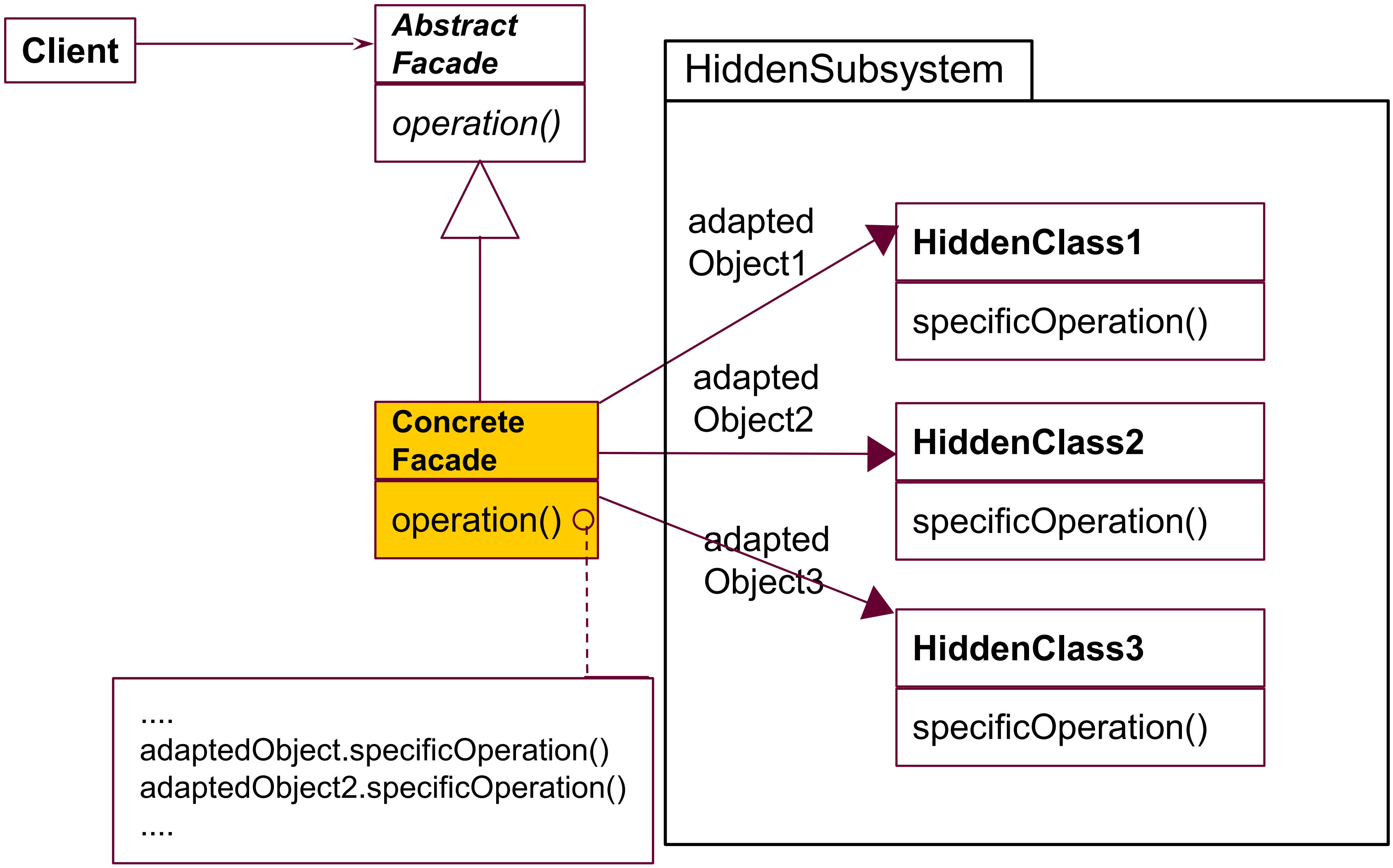
- Similar to a decorator, an adapter inherits its interface from the goal class
  - but adapts the interface
- Hence, adapters can be *inserted* into inheritance hierarchies later on



5.3 Facade

- 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

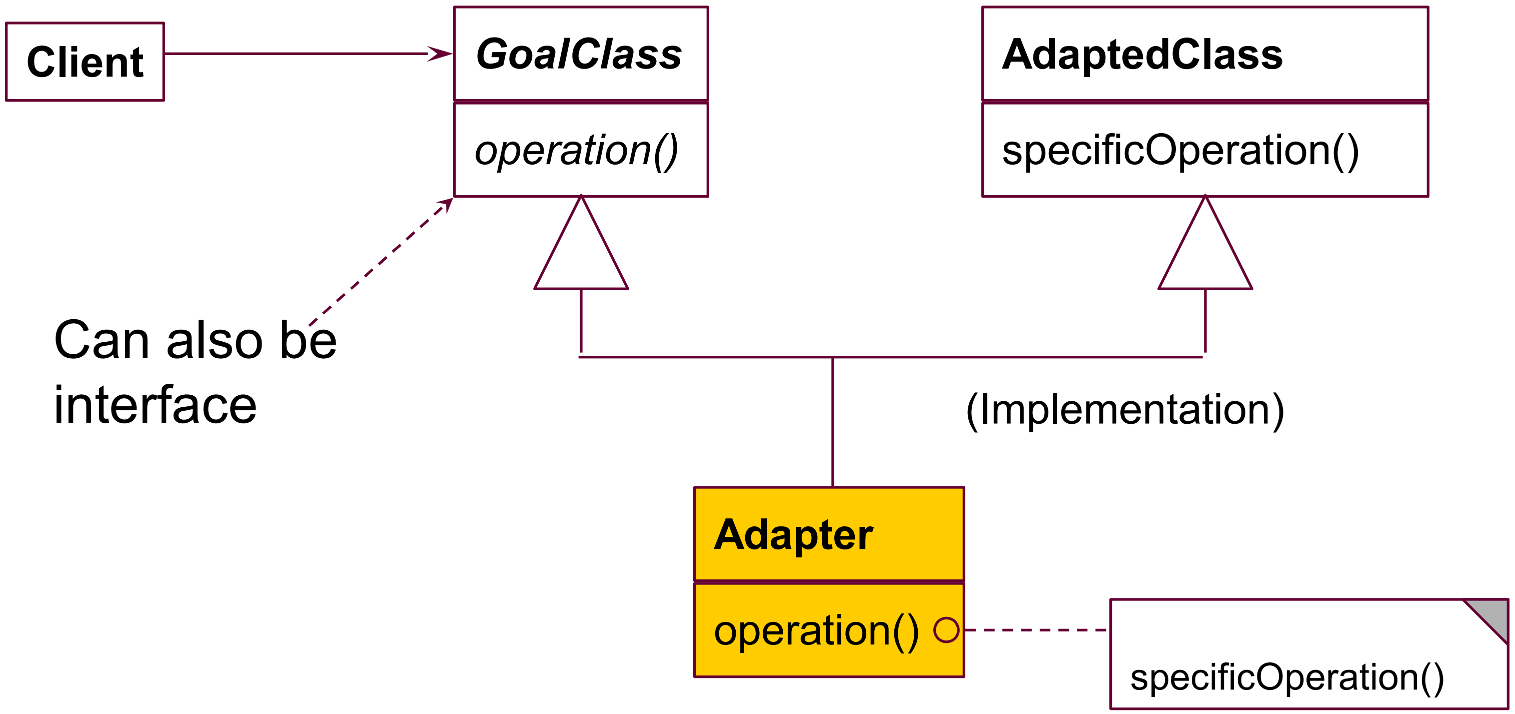
Facade Hides a Subsystem



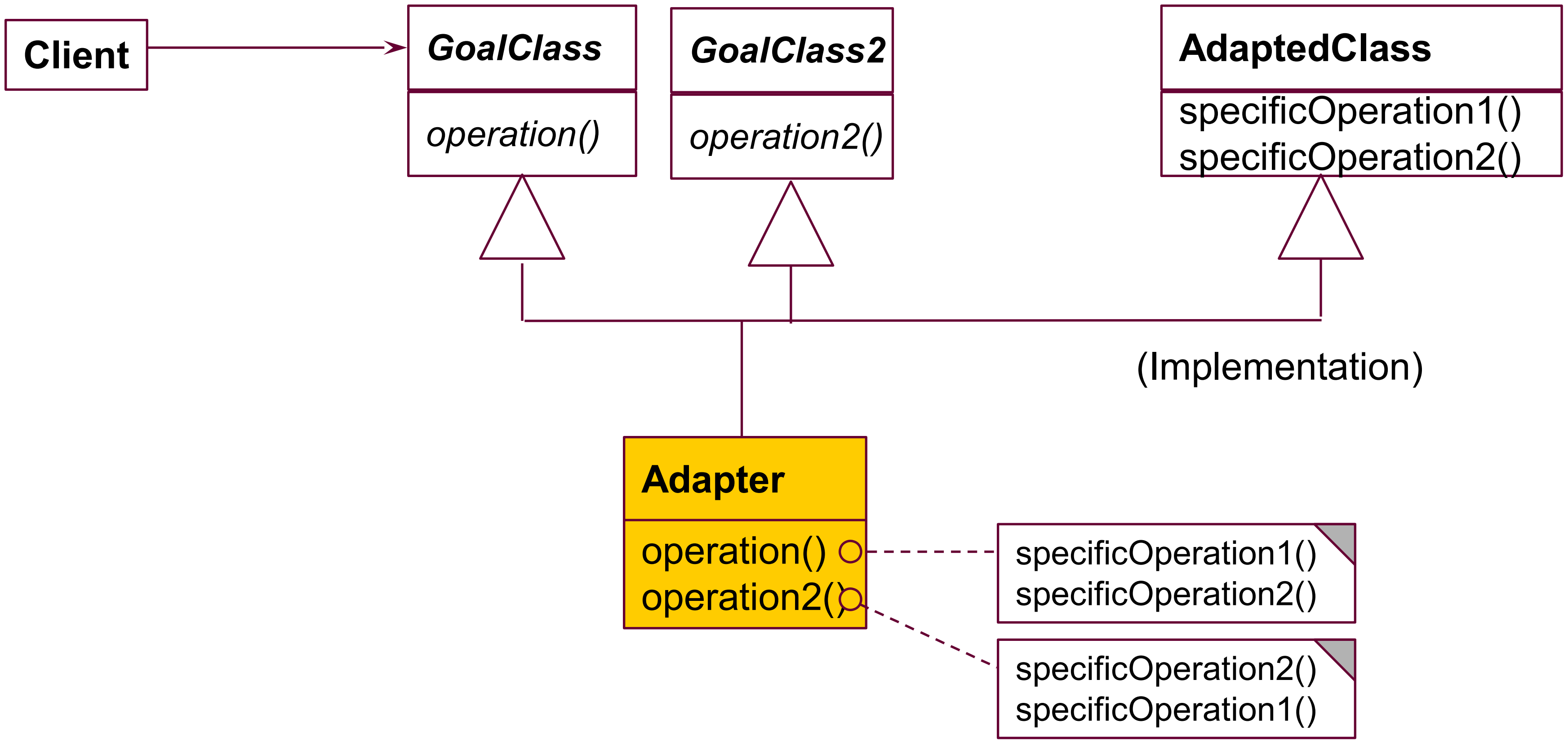
```
....
adaptedObject.specificOperation()
adaptedObject2.specificOperation()
....
```

5.4 Class Adapter (Integrated Adapter)

- Instead of delegation, class adapters use multiple inheritance



2-Way Class Adapter (Role Mediator)

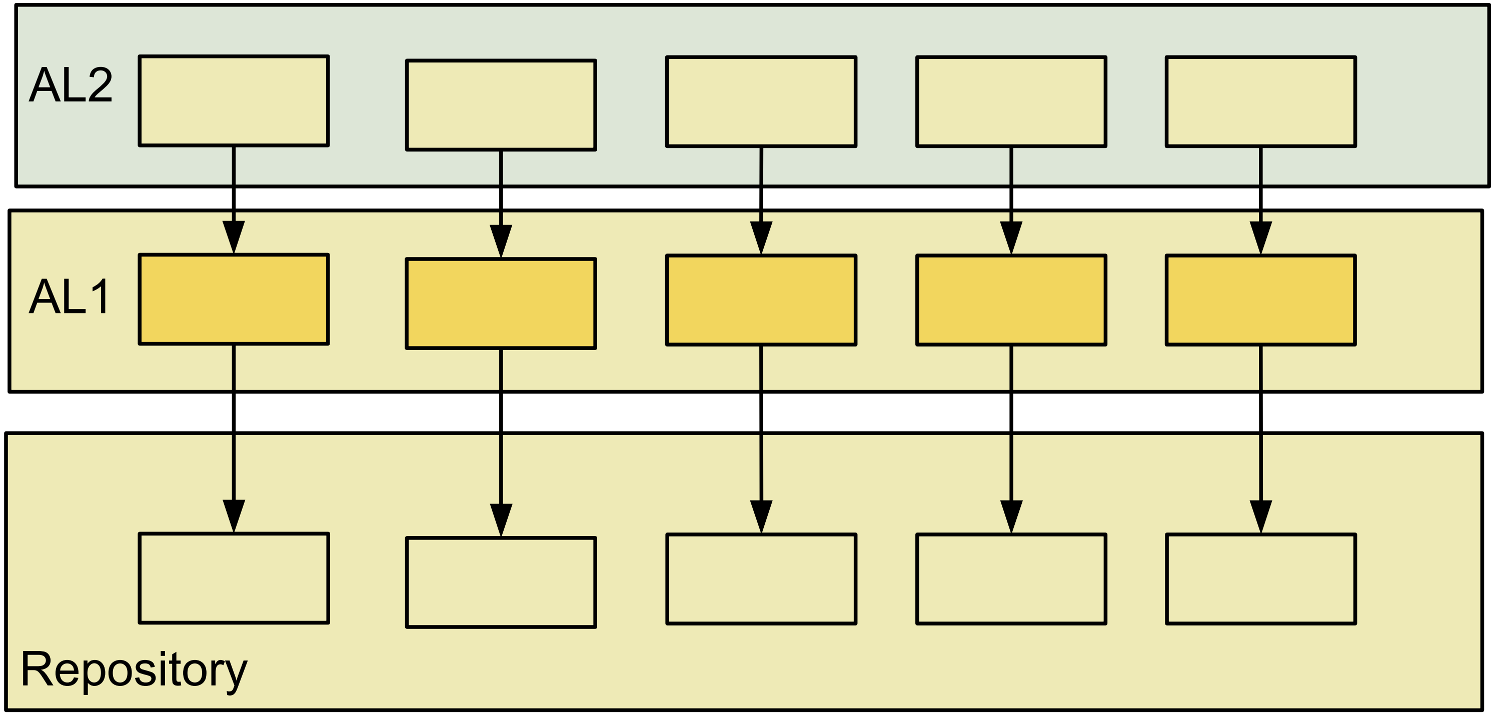


More than one goal class may exist.
Every goal class plays a *role* of the concrete object (see later).

5.5 Adapter Layers

Adapter Layer

- An **Adapter Layer** is a set of adapters hiding a sublayer
  - Every layer has different interfaces (services) that are mapped
- Similar to *Decorator Layer*, but with different interfaces or protocols on each layer

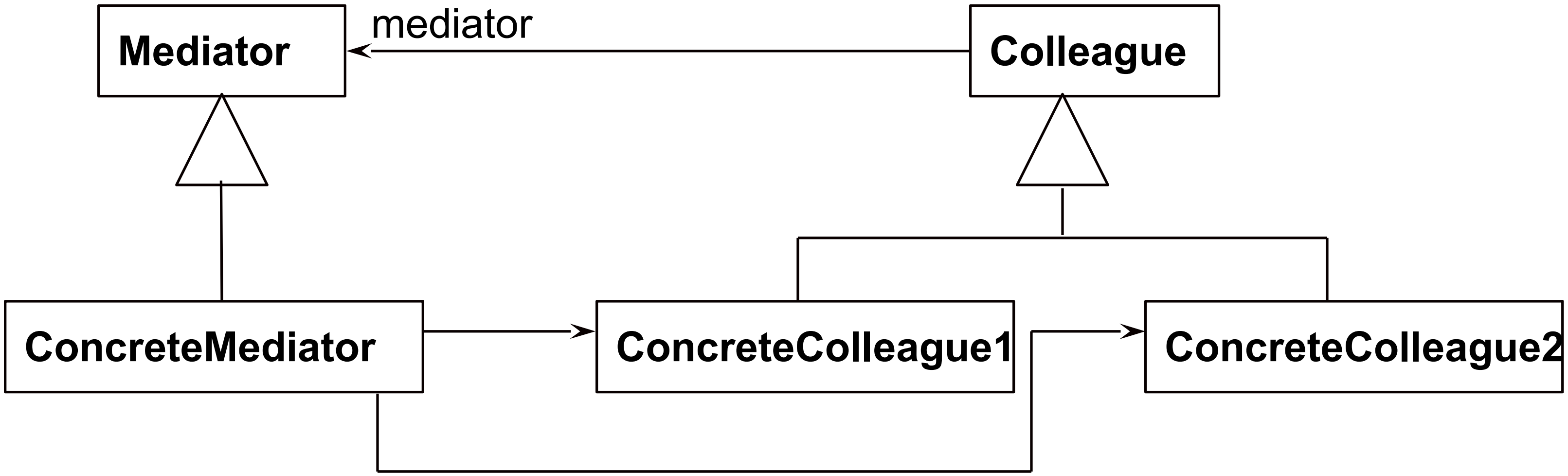


5.6 Mediator (Broker)

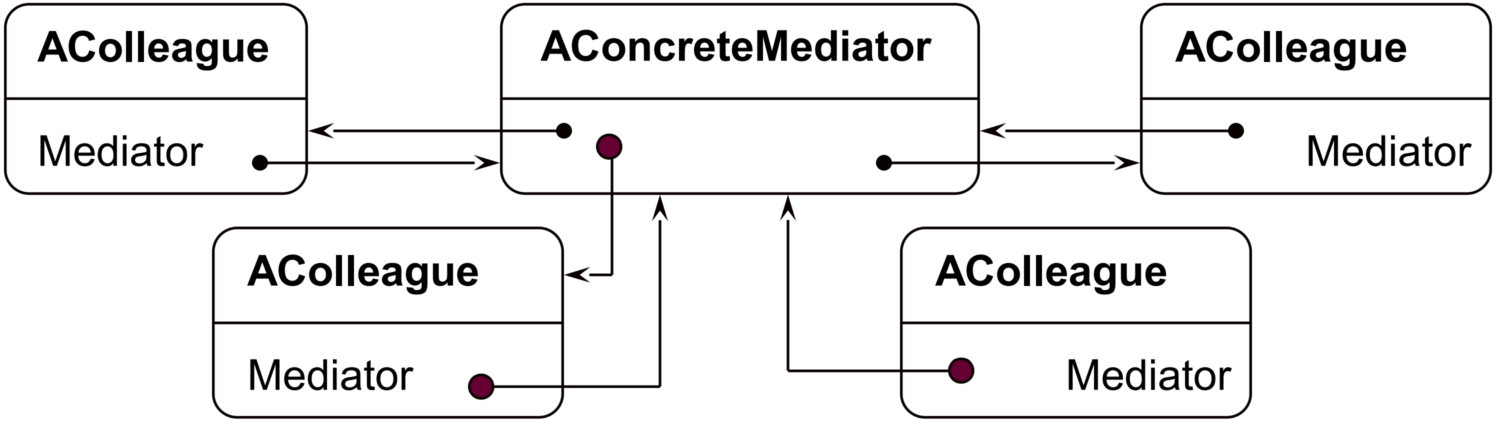
Mediator (Broker)

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

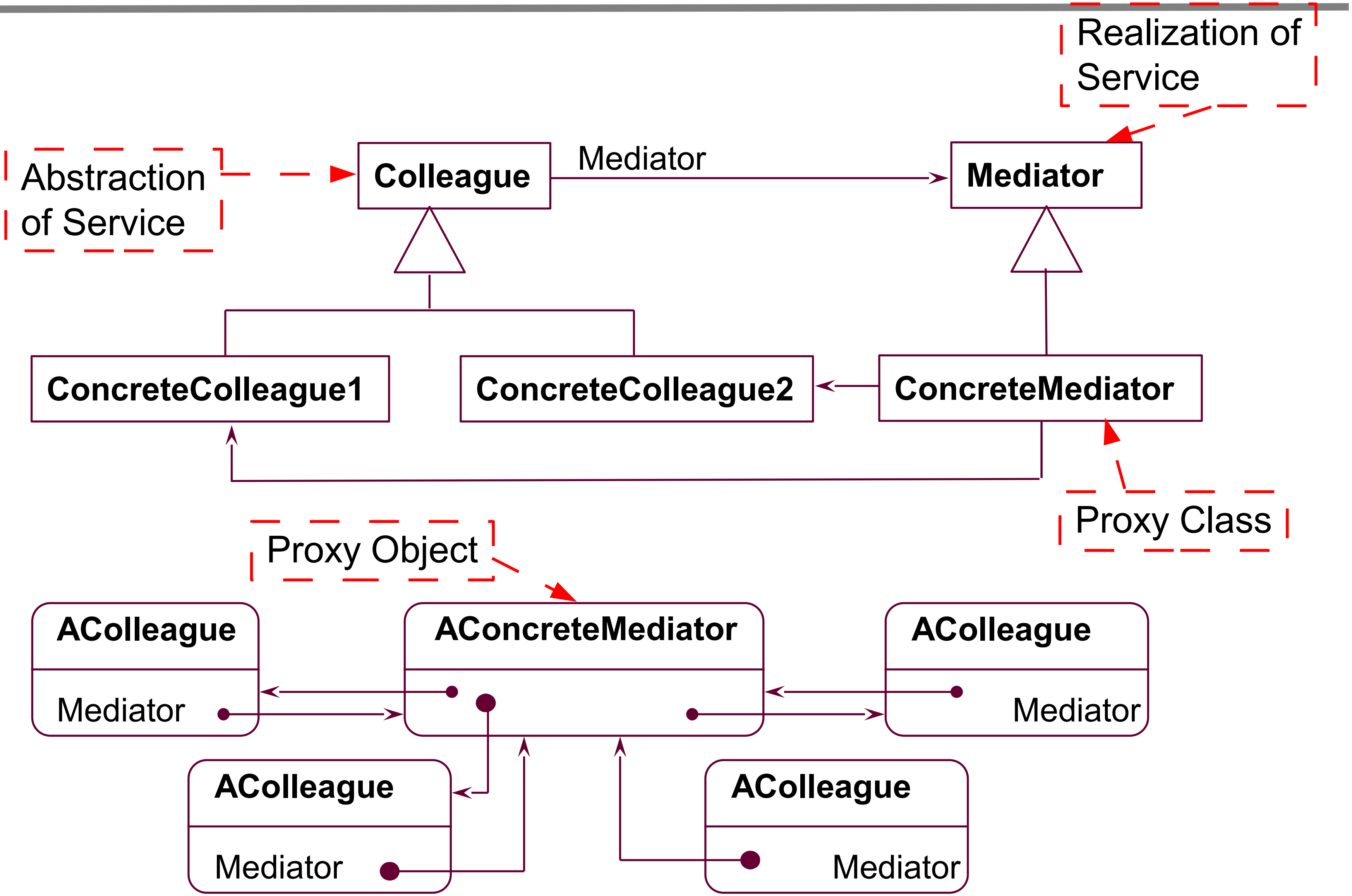
Mediator



Typical Object Structure:



Mediator As n-Proxy and Bridge

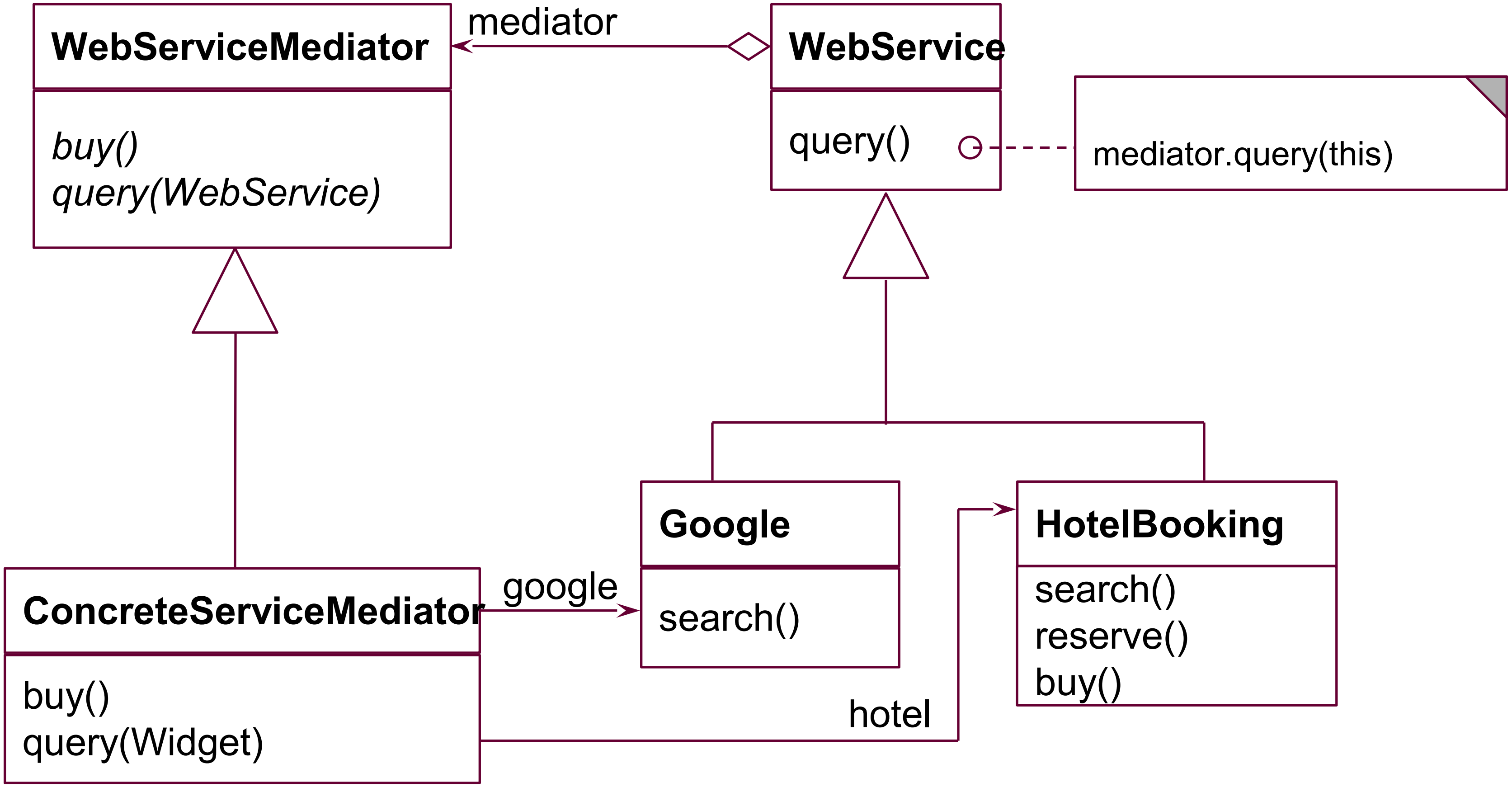


Intent of Mediator

- 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

- Communication between Web services can be mediated via a broker object (aka object request broker, ORB)



5.7 Coupling Tools with the Repository Connector Pattern

Coupling of Tools via Repositories

- How can two tools collaborate that did not know of each other?

- Answer: by coupling their repositories
  - Choose a master and a slave tool
  - Choose a master repository
  - Shadow the master repository in the slave repository
- Consequence: all data lies in slave repository, and can be worked on by slave *and* master

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
- Coupling two tools that had not been foreseen for each other is possible with lazy indirection proxies (RepositoryConnector)
- With Glue Patterns, reuse of COTS becomes much better

The End