
Patterns for Decoupling

Ingolf H. Krueger

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA



California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA



Overview

- Recap: Patterns for Architecture and Design
- Adapter
- Decorator
- Proxy
- Observer
- Model-View-Controller
- Recursive Control

What are Patterns?

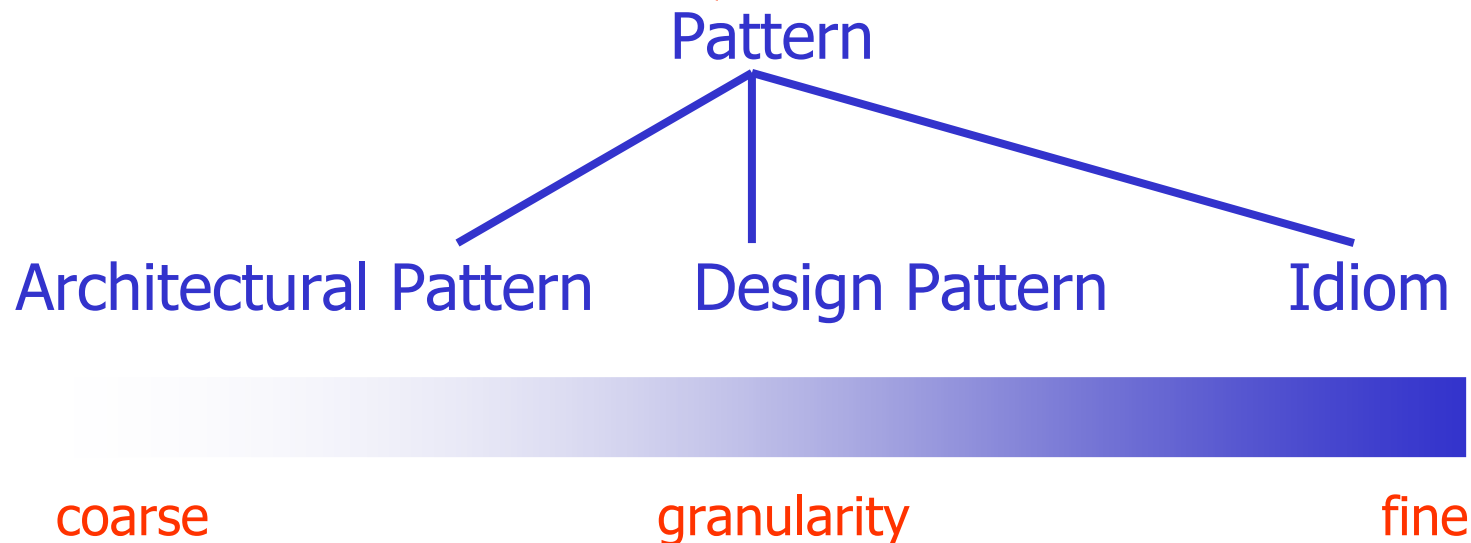
A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate.

[POSA96]

What are Patterns?

adapted from [POSA96]

- Describes *one* proven solution for a recurrent design problem
- Defines the context for the solution's applicability



Design Patterns for Decoupling and Distribution

Key steps during the design of components and their interfaces:

- Decoupling of strongly interdependent components
- Distribution of functions/data to other components

⇒ Application of design patterns

Overview

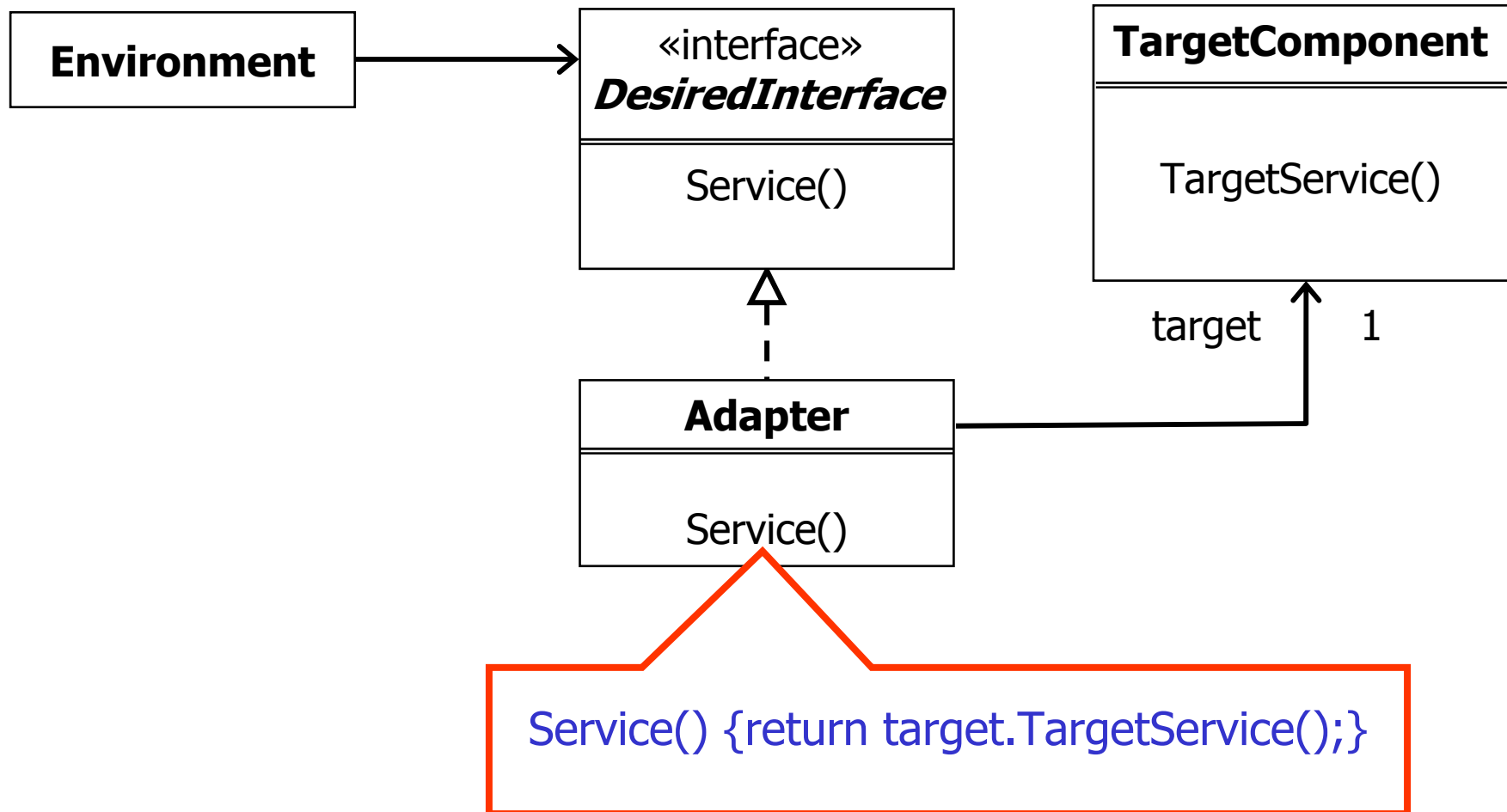
- Recap: Patterns for Architecture and Design
- Adapter
- Decorator
- Proxy
- Observer
- Model-View-Controller
- Recursive Control

Adapter

- Context & Problem
 - A given component (*target*) offers the desired functionality
 - The target's environment expects a different interface
- Solution:
 - Define an adapter component that offers the required interface
 - Associate the environment with the adapter instead of with the target
 - Let the adapter relay calls from the environment to the target; adjust the relayed calls towards the interface of the target
 - Let the adapter provide functions missing in the target

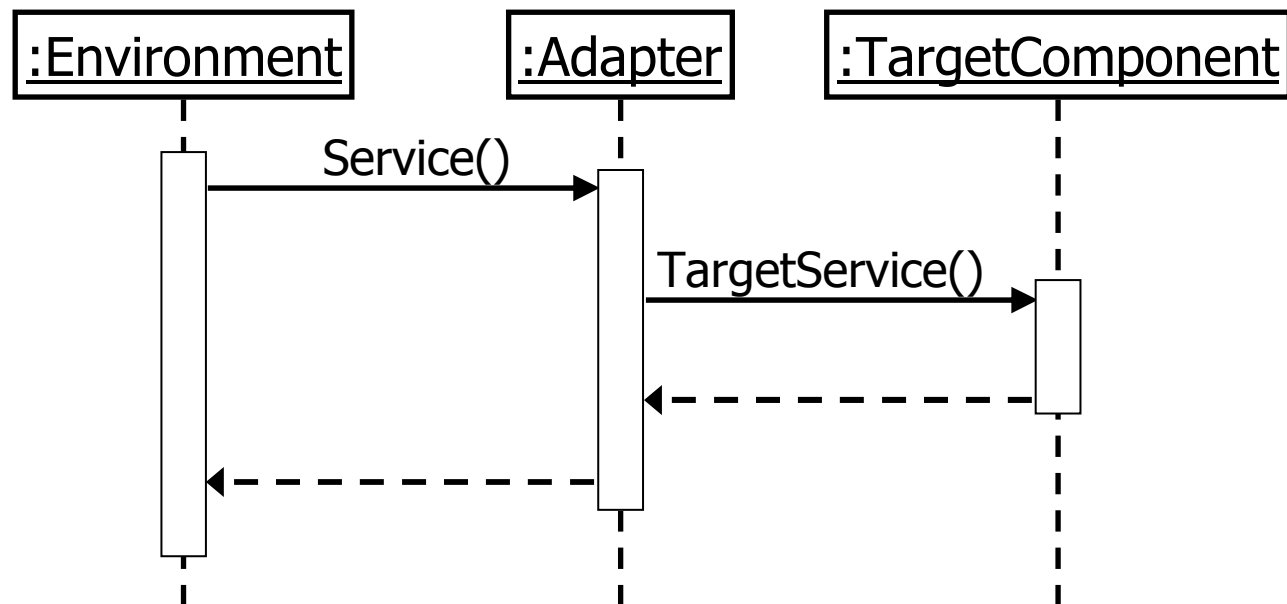
Adapter

- Structure:



Adapter

- Behavior:



Adapter

- Consequences:
 - Enables reuse of target in different context
 - Extension of functionality via the adapter potentially reduces the design's transparency
 - If target becomes subcomponent of adapter: adjusting of all clients of target required to accommodate the adapter's interface

Overview

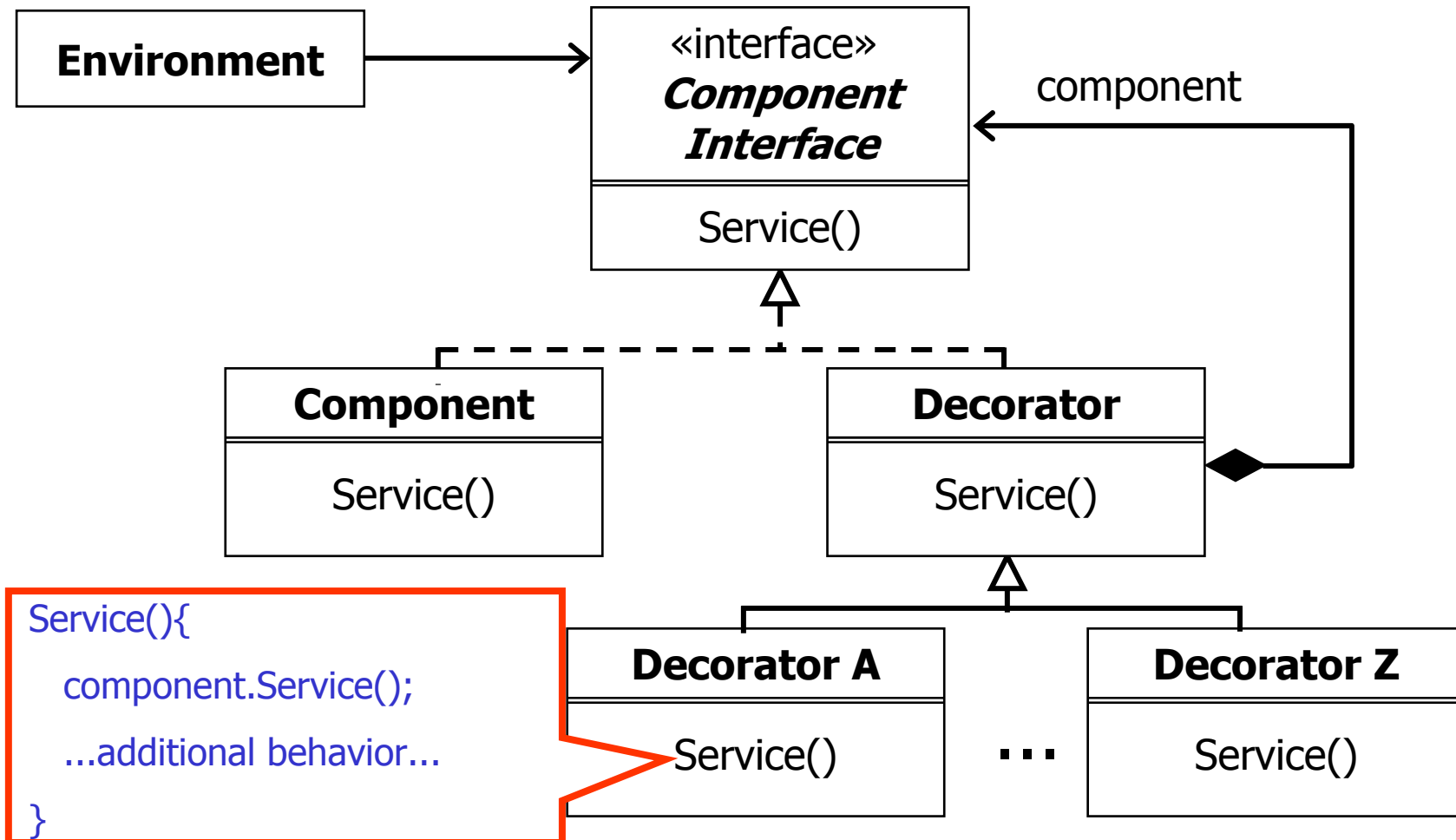
- Recap: Patterns for Architecture and Design
- Adapter
- Decorator
- Proxy
- Observer
- Model-View-Controller
- Recursive Control

Decorator

- Context & Problem
 - Modification of a target component's *behavior* is required
 - The component's *interface* has to remain unchanged
- Solution:
 - Define a decorator component that has the target's interface
 - Store a reference to the target in the decorator
 - Let the decorator relay calls to the target
 - Let the decorator adjust the result obtained from the target to match the required modifications, and relay the adjusted result to the original caller

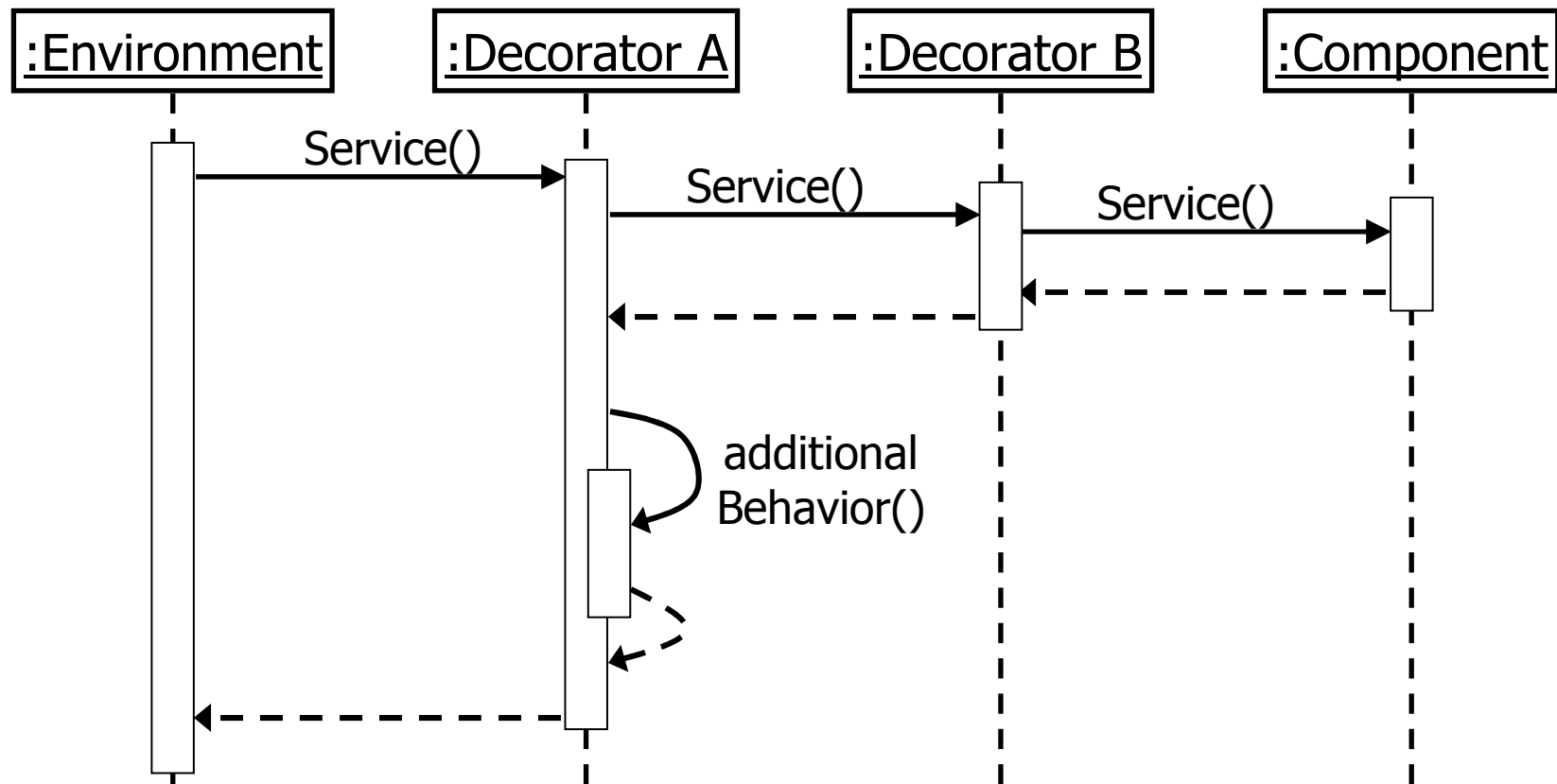
Decorator

- Structure:



Decorator

- Behavior:



Decorator

- Consequences:
 - Flexible addition to/modification of component behavior
 - Alternative to overloaded class hierarchies
 - Communication overhead
(potential performance loss)

Overview

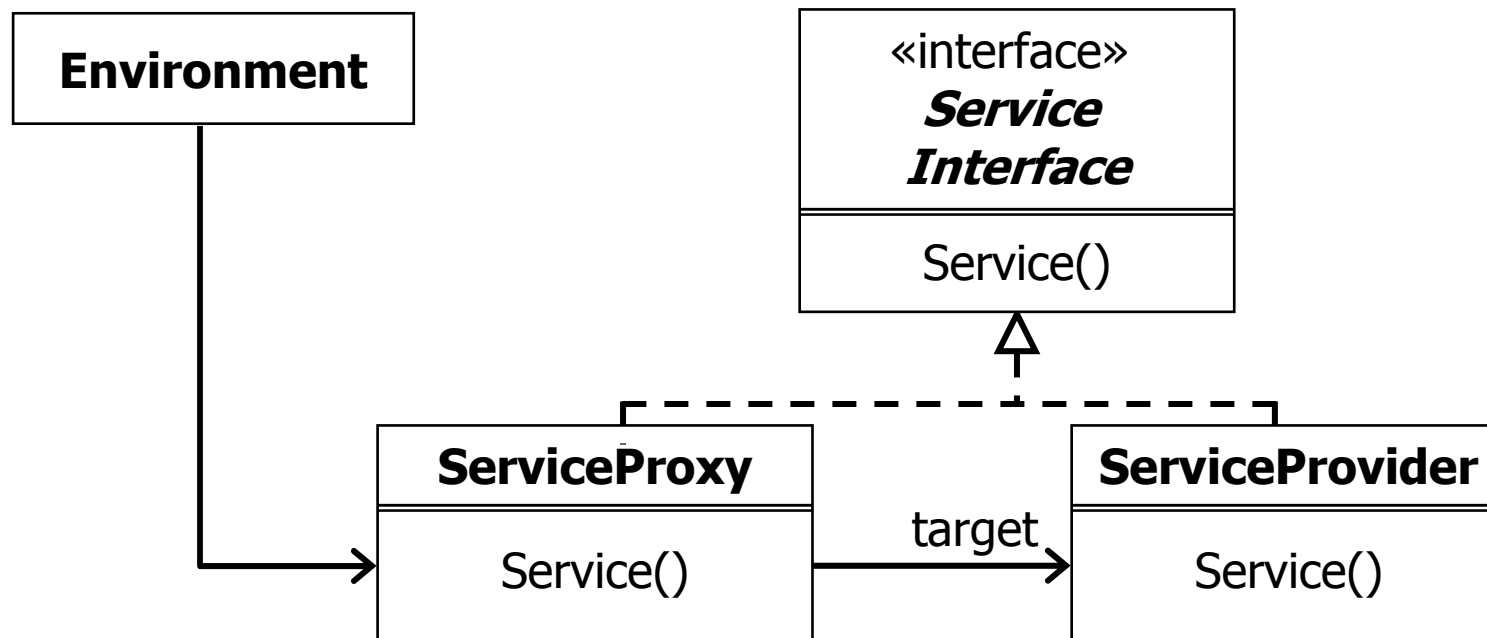
- Recap: Patterns for Architecture and Design
- Adapter
- Decorator
- Proxy
- Observer
- Model-View-Controller
- Recursive Control

Proxy

- Context & Problem
 - Access to target component from several other components required
 - Direct access to the target is impossible due to security, efficiency, or distribution requirements
- Solution:
 - Define a placeholder (a proxy) that offers the same interface as the target component
 - Make only the proxy known to the environment
 - Let the proxy access the target, observing security, efficiency and distribution requirements

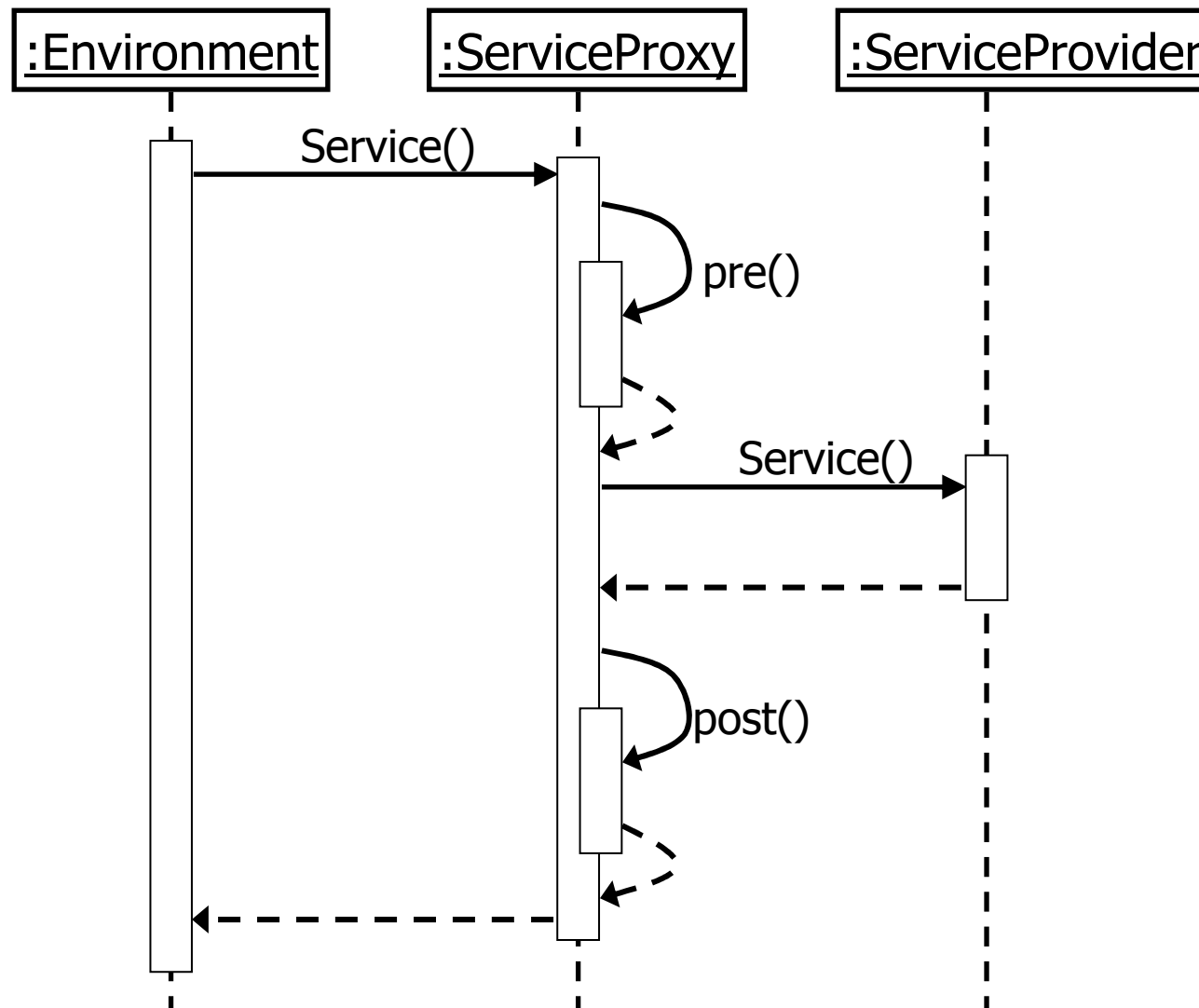
Proxy

- Structure:



Proxy

- Behavior:



Proxy

- Consequences:
 - Decoupling of service provider and service users
 - Location of service provider is transparent to service users
 - Proxy handles communication with service provider
 - Applicable for decoupling of address spaces
 - Increased communications overhead can reduce performance

Overview

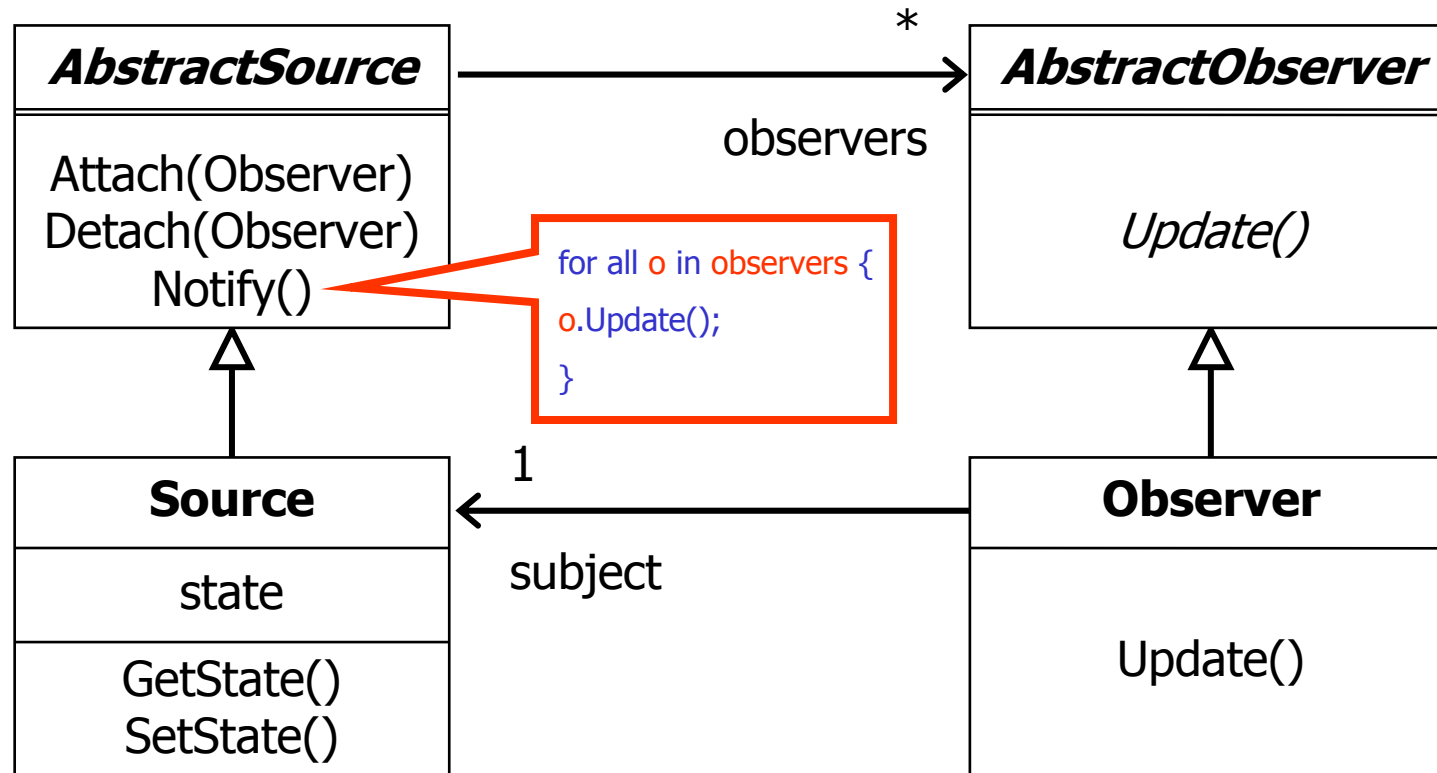
- Recap: Patterns for Architecture and Design
- Adapter
- Decorator
- Proxy
- **Observer**
- Model-View-Controller
- Recursive Control

Observer

- Context & Problem
 - Several components depend on a target component's state
 - All dependent components shall be informed about state changes of the target
 - Loose coupling between dependent and target component target required
- Solution:
 - Let all dependent components attach/register with the source
 - Let components no longer interested in the target's state changes detach/unregister from the target
 - In the event of a state change let the target send a notification message to all registered components
 - Let the registered components inquire about the new state after they have been notified

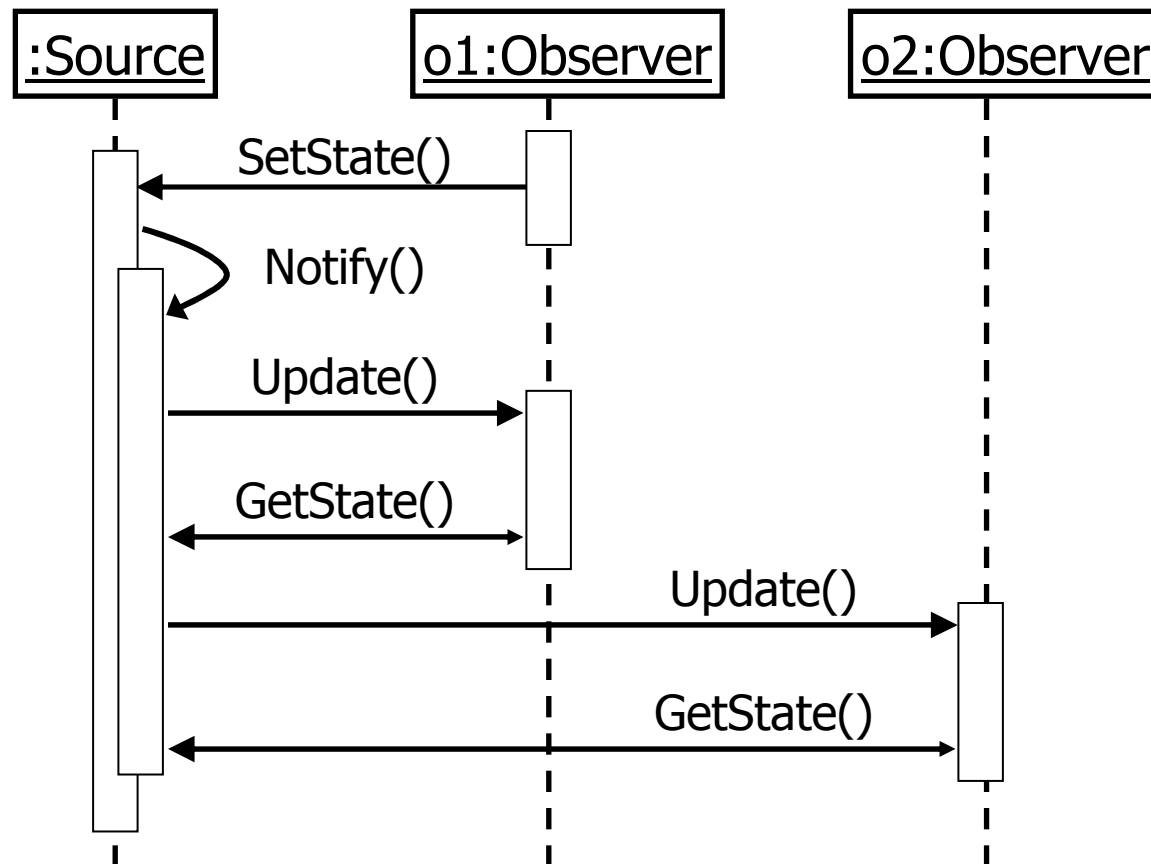
Observer

- Structure:



Observer

- Behavior:



Observer

- Consequences:
 - Loose coupling between target and dependent components
 - Support for event-based systems (multithreading)
 - Design strategy for multicast-systems
 - Problematic: Containment of update frequency
Solution: apply multiple update strategies (strategy pattern)

Overview

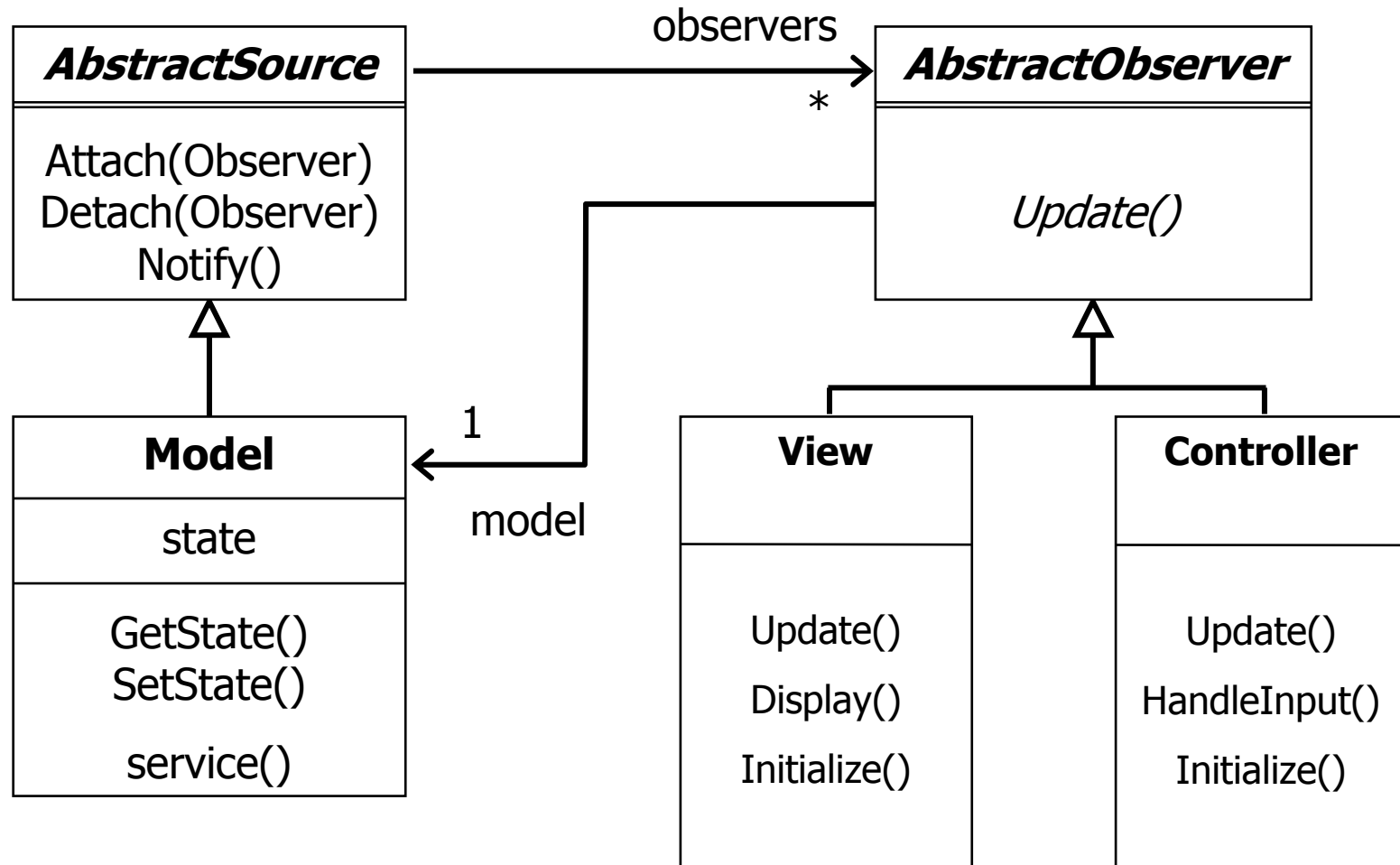
- Recap: Patterns for Architecture and Design
- Adapter
- Decorator
- Proxy
- Observer
- Model-View-Controller
- Recursive Control

MVC

- Context & Problem
 - Application data needs to be maintained and presented via the user interface
 - Multiple different output formats need to be supported
 - Multiple different forms of input need to be supported
 - How to establish consistency?
- Solution:
 - Introduce separate components for storing and processing of data (model), data presentation (view), and for handling input (controller)
 - The model represents the functional core; it registers dependent components (views and controllers) and notifies them about data changes
 - The view retrieves data from the model and displays it
 - The controller translates user input into events for the model; it may also change the UI to mirror data changes

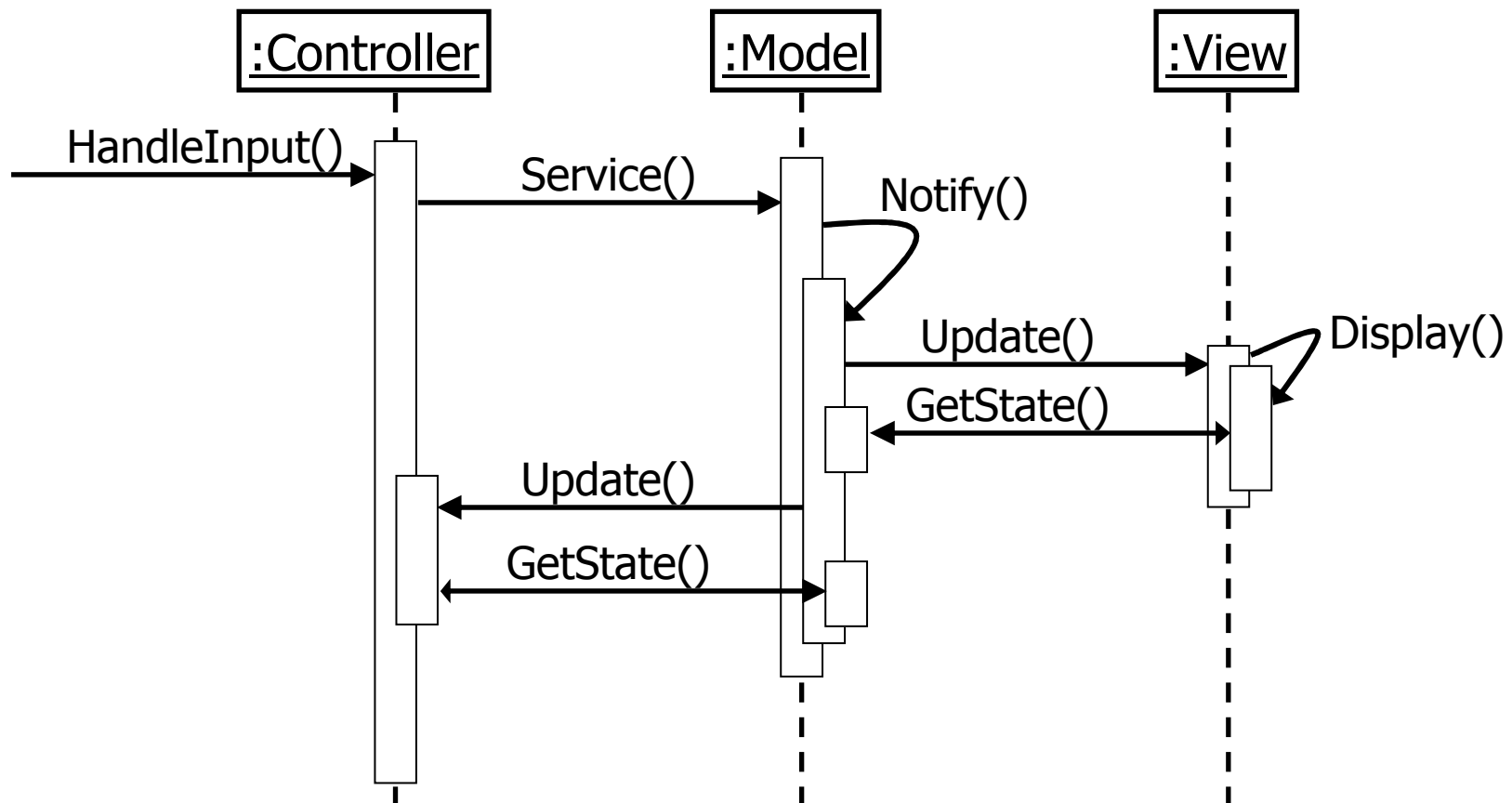
MVC

- Structure:



MVC

- Behavior:



MVC

- Consequences:
 - Decoupling of application data from presentation and input mechanism
 - Consistency of user interface and underlying data model
 - Increase of structural and dynamic complexity
 - Potential loss of performance

Overview

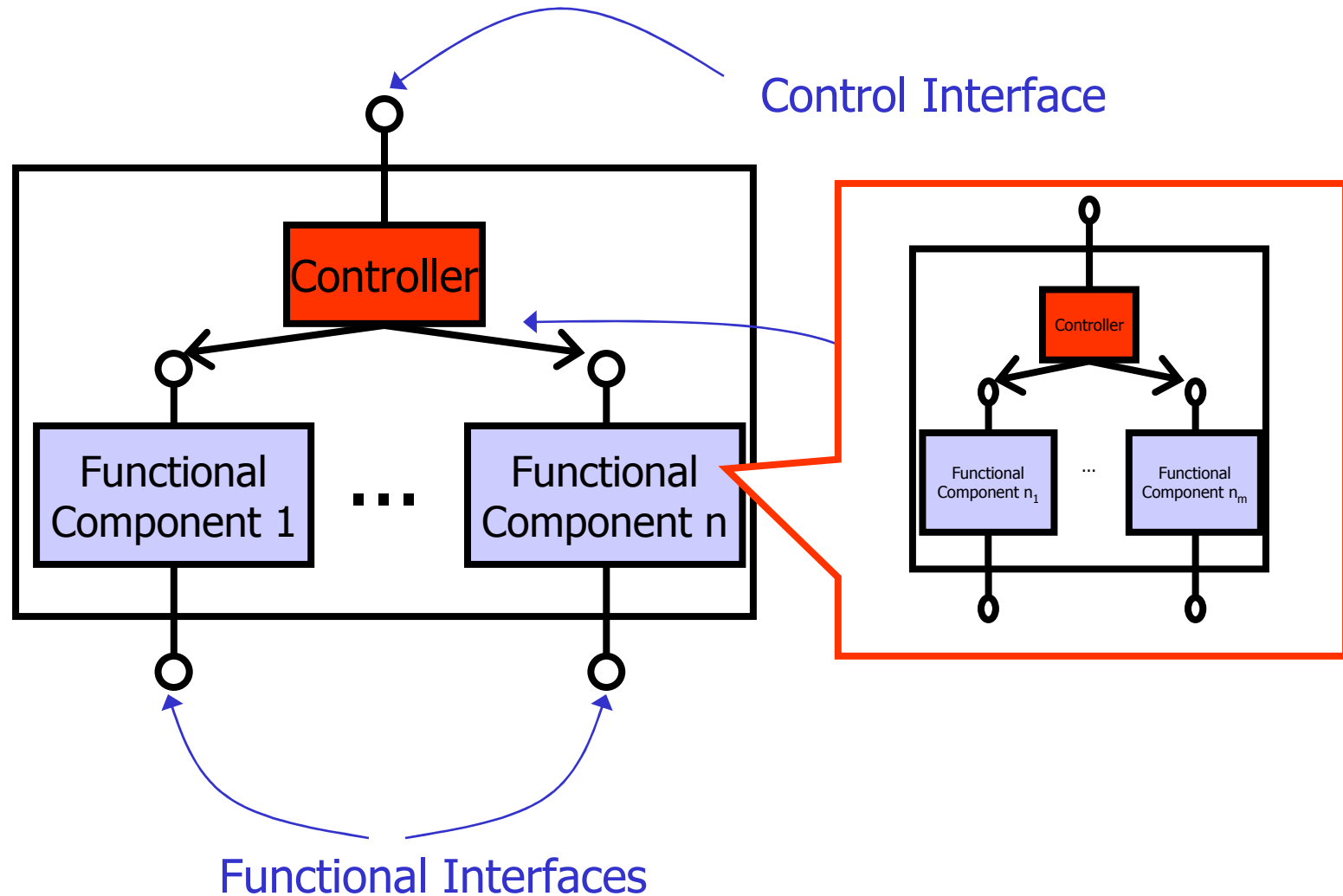
- Recap: Patterns for Architecture and Design
- Adapter
- Decorator
- Proxy
- Observer
- Model-View-Controller
- Recursive Control

Recursive Control

- Context & Problem
 - Decomposition of complex system into subsystems
 - Subsystems have significant control part:
 - Installation of new components at runtime
 - System start, -initialization, -restart and -stop
 - Fault diagnosis and -handling
 - Performance monitoring
 - Synchronization with environment
 - ...
 - Control part “overloads” functional design
- Solution:
 - Decoupling of control and function in every component
 - Separation of interfaces for control and function
 - Hierarchical composition

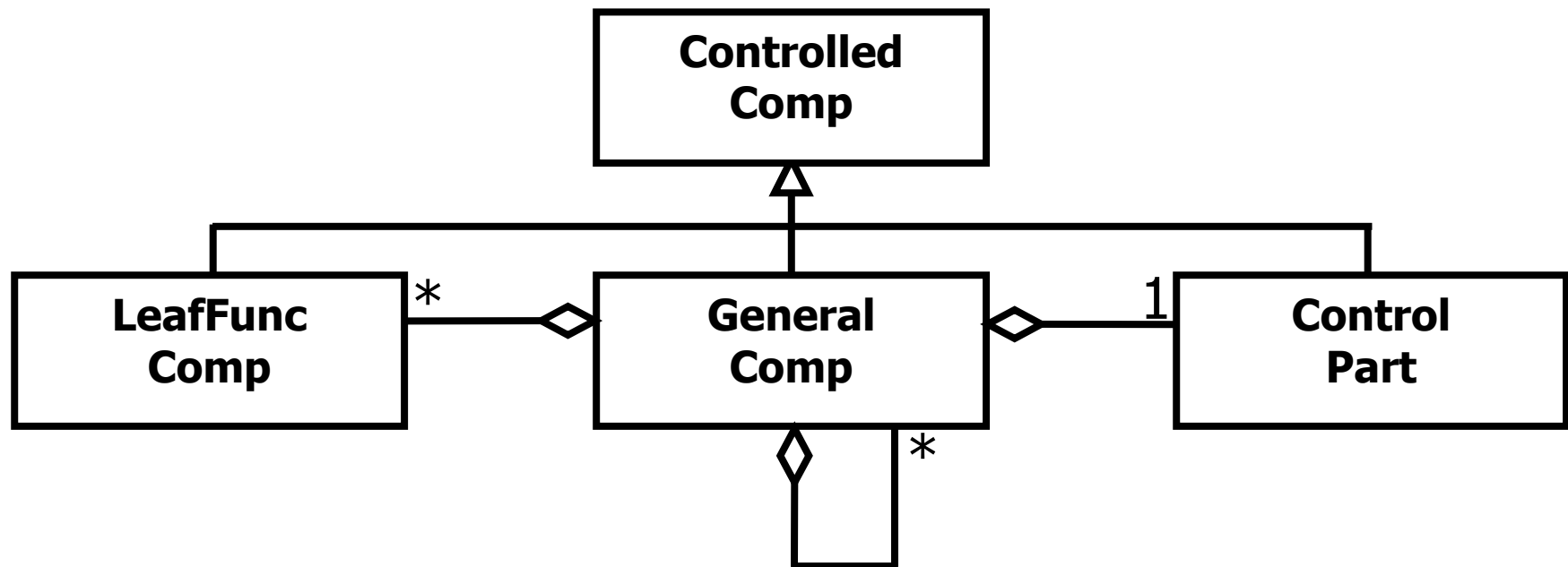
Recursive Control

- Structure:



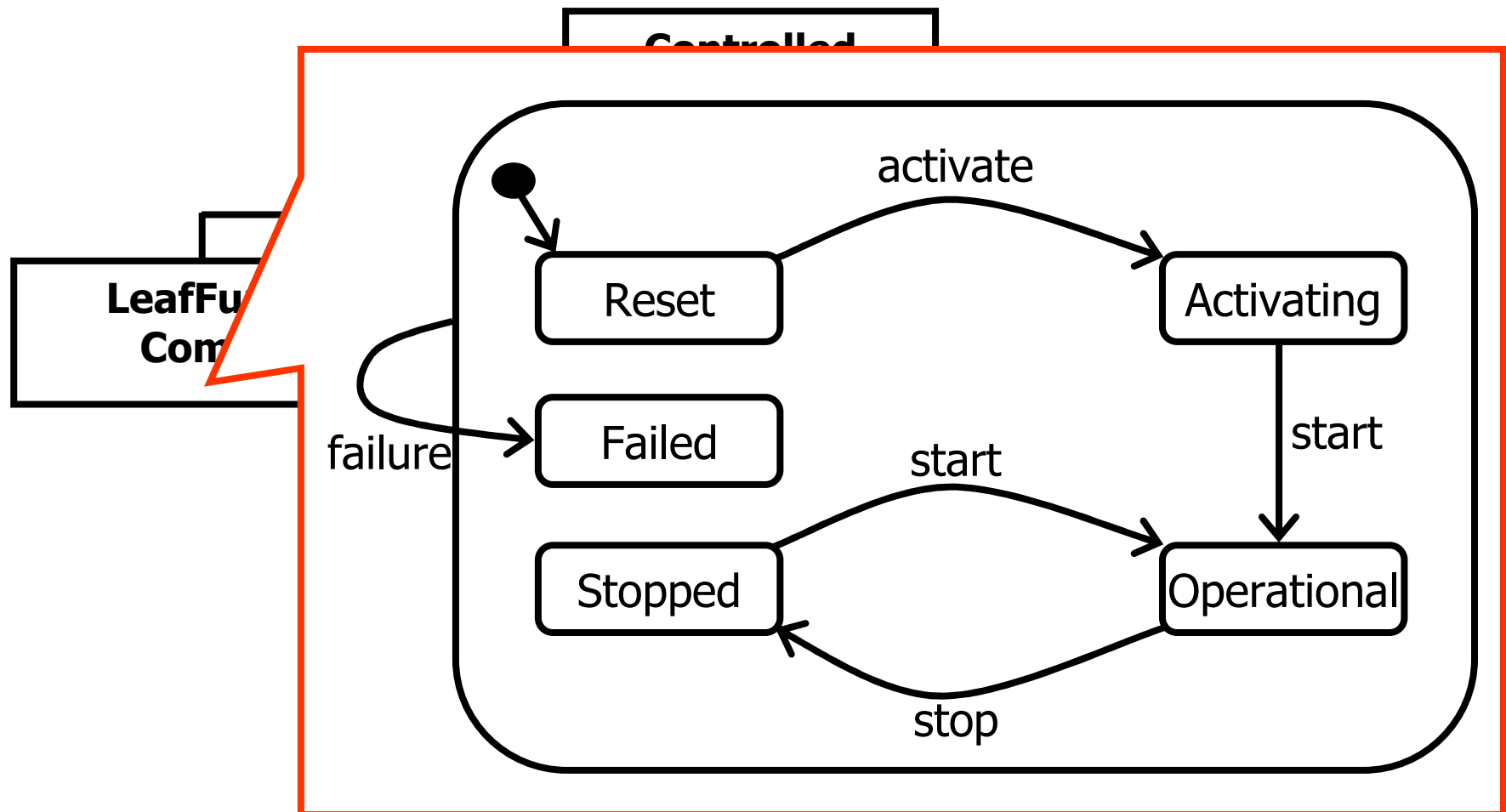
Recursive Control

- Structure:



Recursive Control

- Behavior
 - Usage of hierarchical state machines

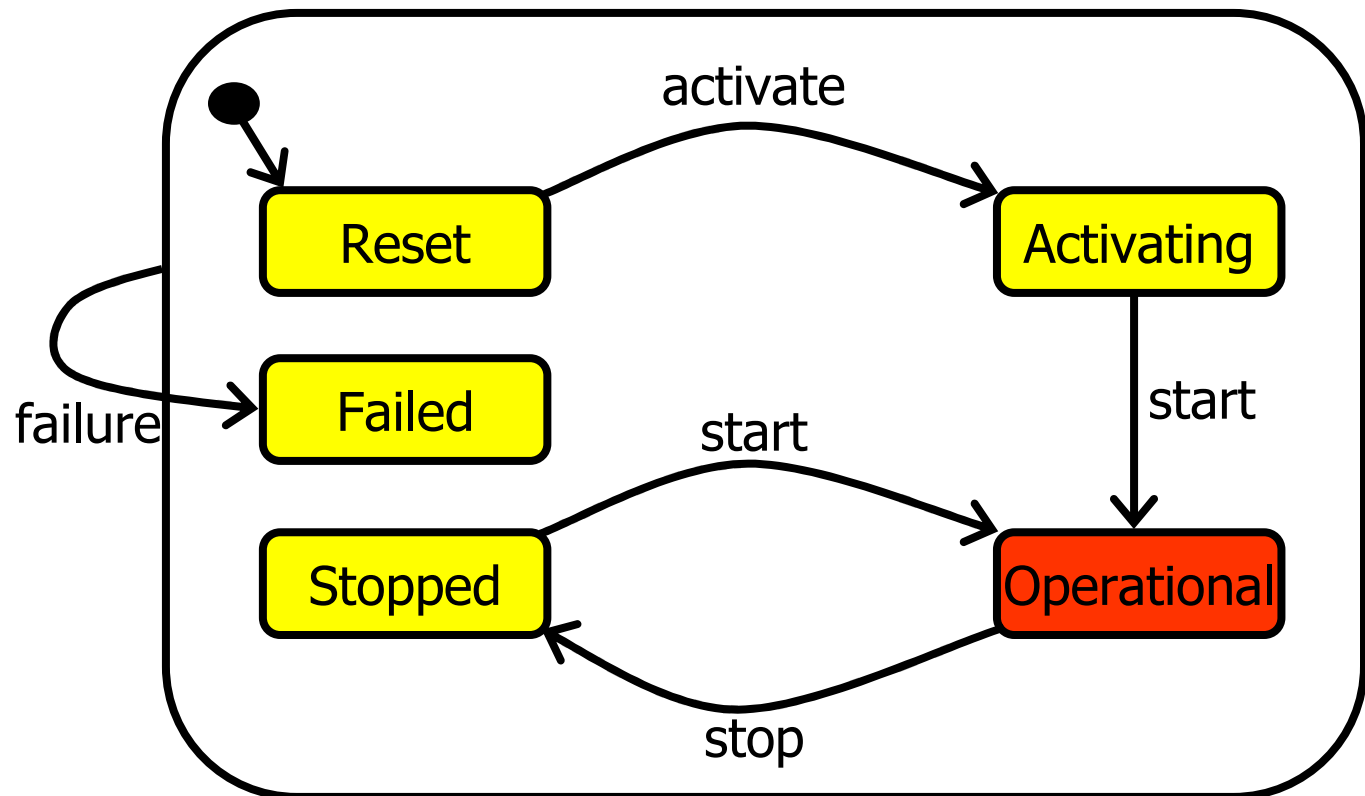


Recursive Control

- Behavior
 - Usage of hierarchical state machines

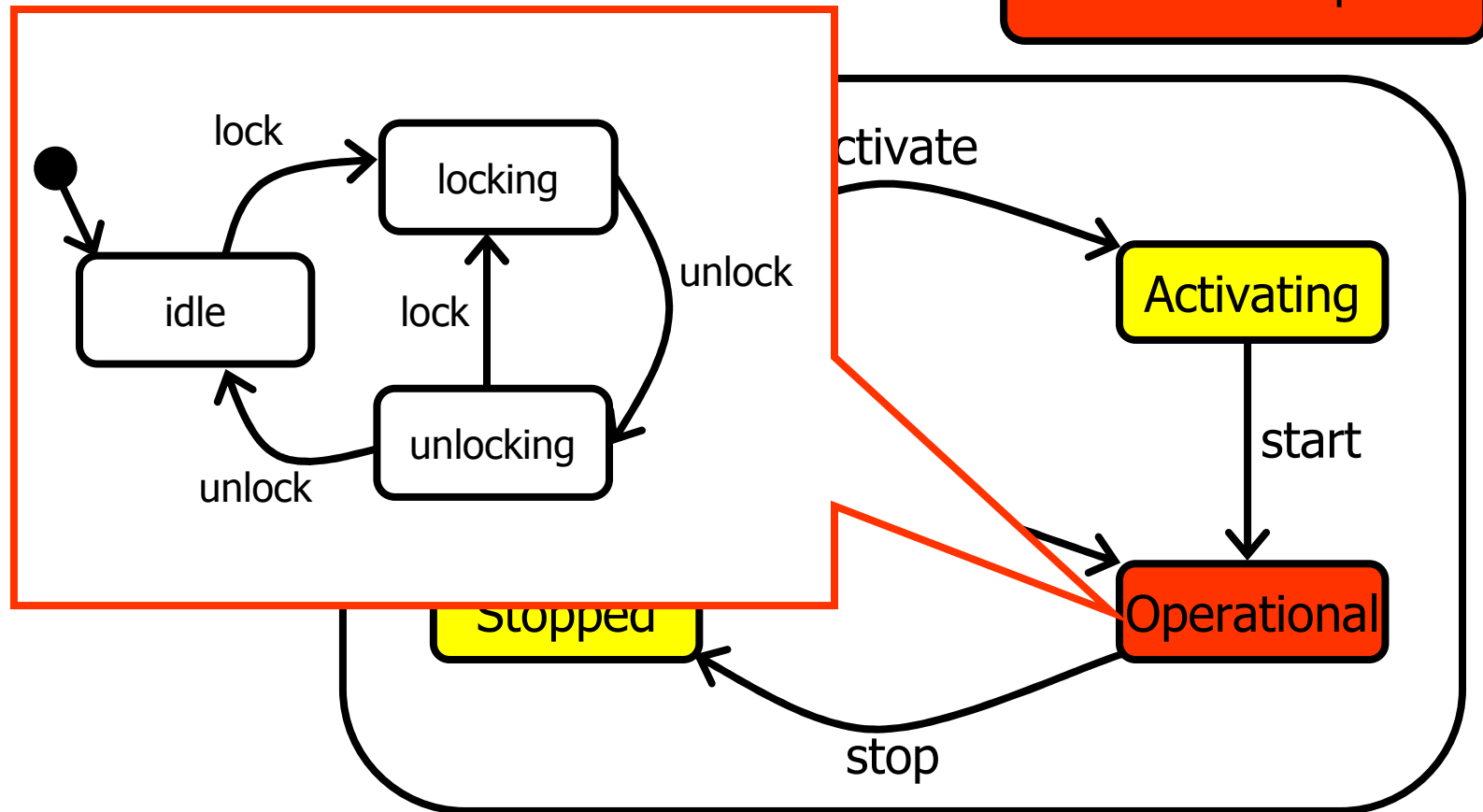
Control Aspects

Functional Aspects



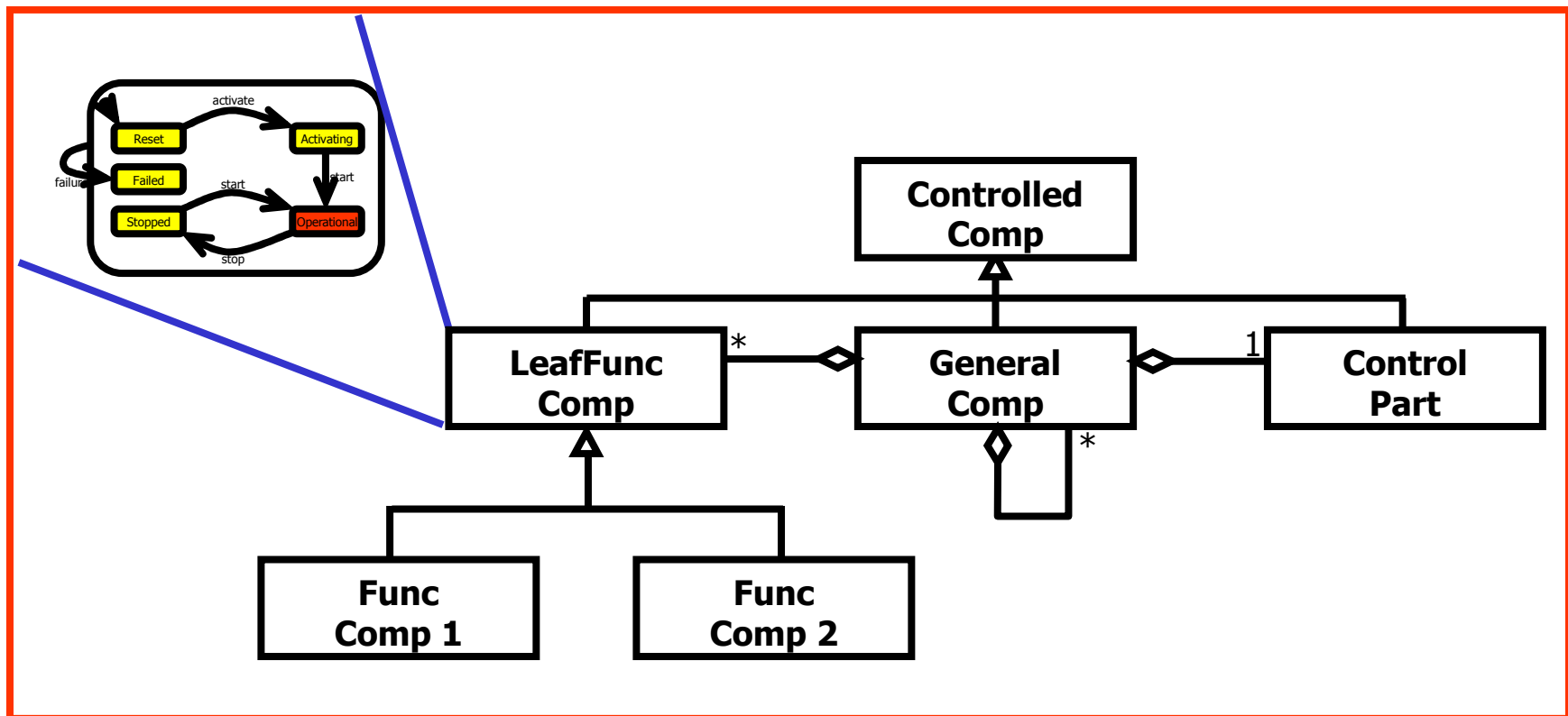
Recursive Control

- Behavior
 - Usage of hierarchical state machines



Recursive Control

- Example Applications:
 - Control-Theory
 - Telecommunication: Switches



Sources/References

- Adapter*
- Decorator *
- Proxy*
- Observer*
- Model-View-Controller+
- Recursive Control#

* Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

+ Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996

Bran Selic. *An Architectural Pattern for Real-time Control Software*. In Vlissides, J.M., J.O. Coplien, and N.L. Kerth. (eds.): *Pattern Languages of Program Design 2*. Addison-Wesley, 1996.