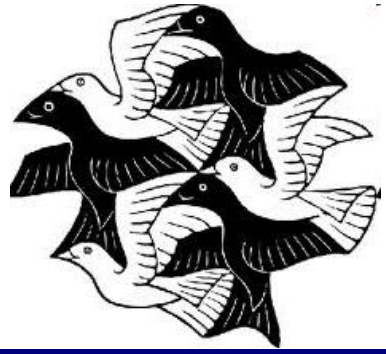


9. Patterns' relationships

SW Design Patterns,
by Boyan Bontchev,
FMI - Sofia University

Annotation

- Definitions
- Classifications
- GoF relationships
- Zimmer’s relationships
- Riehle composite patterns
- Conclusions

References

- Gamma, Helm, Johnson, Vlissides **("Gang of Four“ - GoF**) *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995
- *Design Patterns Explained*, by Allan Shalloway and James Trott, Prentice Hall, 2001
- W. Zimmer. “Relationships Between Design Patterns”In *Pattern Languages of Program Design*, J. O. Coplien and D. C. Schmidt (eds.), Reading, MA: Addison-Wesley, 1995, pp. 345
- Dirk **Riehle**. A Role-Based Design Pattern Catalog of Atomic and Composite Patterns Structured by Pattern Purpose. Ubilab Technical Report, 1997
- Dirk **Riehle**. "Bureaucracy - A Composite Pattern.“, Proc. of EuroPLoP '96, 1996.

Pattern classifications

- The increase in the number of patterns made it very important to develop proper methodologies and techniques how to organize them.
- **Pattern classification** is the organization of patterns into **groups** of patterns sharing the same set of **properties**.
- The kind of these properties is not fixed and may include criteria such as structure, intent, or applicability.
- Depending on the chosen criteria, we could define a classification schema.

Dimensions of classification schemas

- Different classification schemas can have different **dimensions**.
- A two dimensional schema, for example, uses two criteria in the classification process.
- Usually, the more dimensions a schema has, the more useful the classification is.
- The same pattern can have different kinds of properties, it can be included in more than one category.
- Each property maps the pattern to the category of that property kind.

GoF classification 1/2

In Gamma's book, patterns are classified by two criteria:

- First is **Purpose**, which reflects what pattern does. Patterns can have either creational, structural, or behavioral purpose.
  - Creational patterns concern the process of object creation.
  - Structural patterns deal with the composition of classes or objects.
  - Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility.

GoF classification 2/2

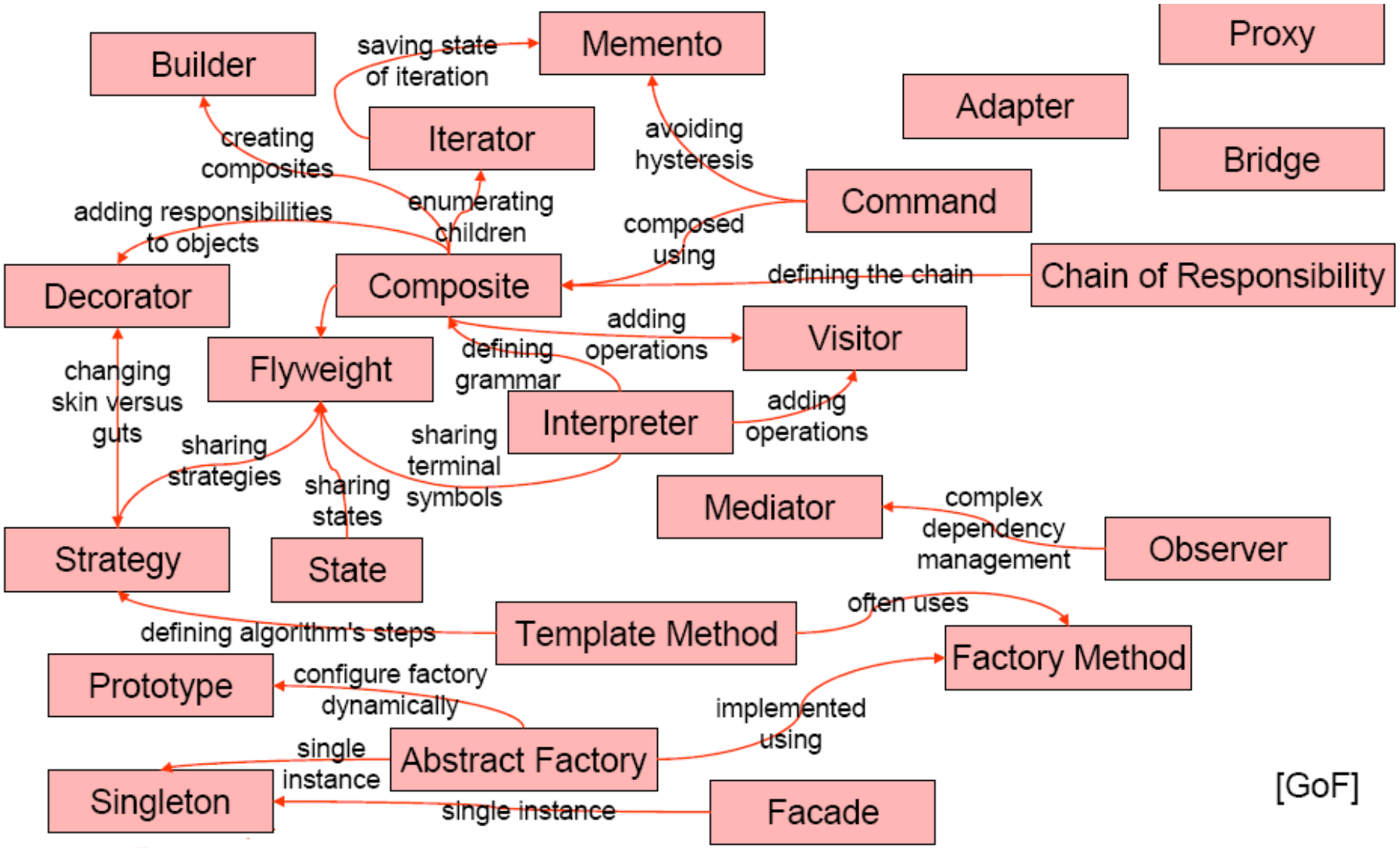
In Gamma's book, patterns are classified by two criteria:

- First is **Purpose**, which reflects what pattern does. Patterns can have either creational, structural, or behavioral purpose.
- The second criterion is **Scope**, which specifies whether the pattern applies primarily on classes or objects.
  - Class patterns deal with relationships between classes and their sub-classes. These relationships are established through inheritance, so they are static (fixed at compile time).
  - Object patterns deal with object relationships, which can be changed at run-time and are more dynamic (patterns labeled as class patterns are those that focus on class relationships.

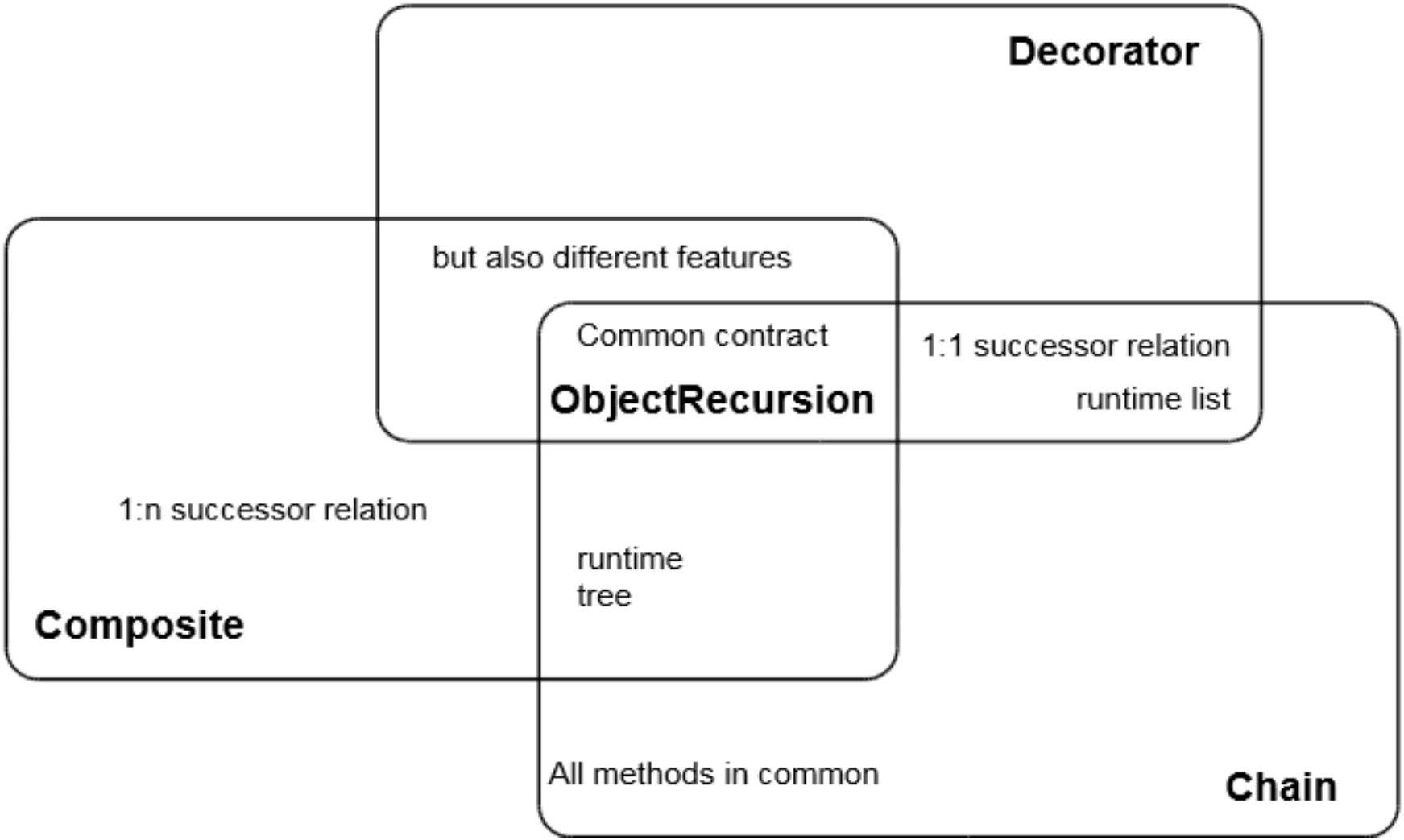
Design pattern catalog - GoF

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	• **Factory Method**	• **Adapter**	• **Interperter** • **Template Method**
	Object	• **Abstract Factory** • **Builder** • **Prototype** • **Singleton**	• **Bridge** • **Composite** • **Decorator** • **Facade** • **Flyweight** • **Proxy**	• **Chain of Responsibility** • **Command** • **Iterator** • **Mediator** • **Observer** • **State** • **Strategy** • **Visitor** • **Memento**

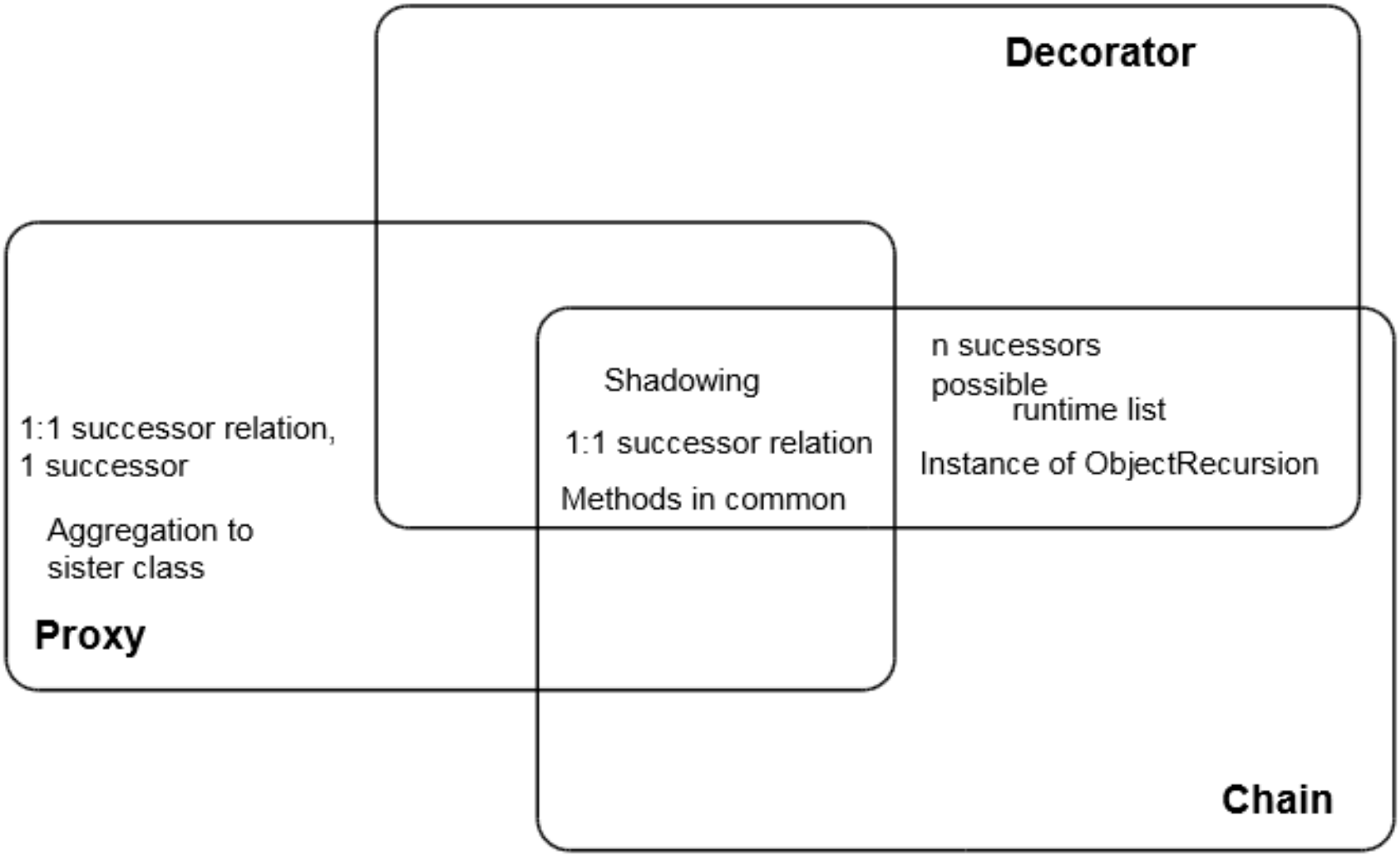
GoF pattern relationships



Composite vs Decorator vs Chain



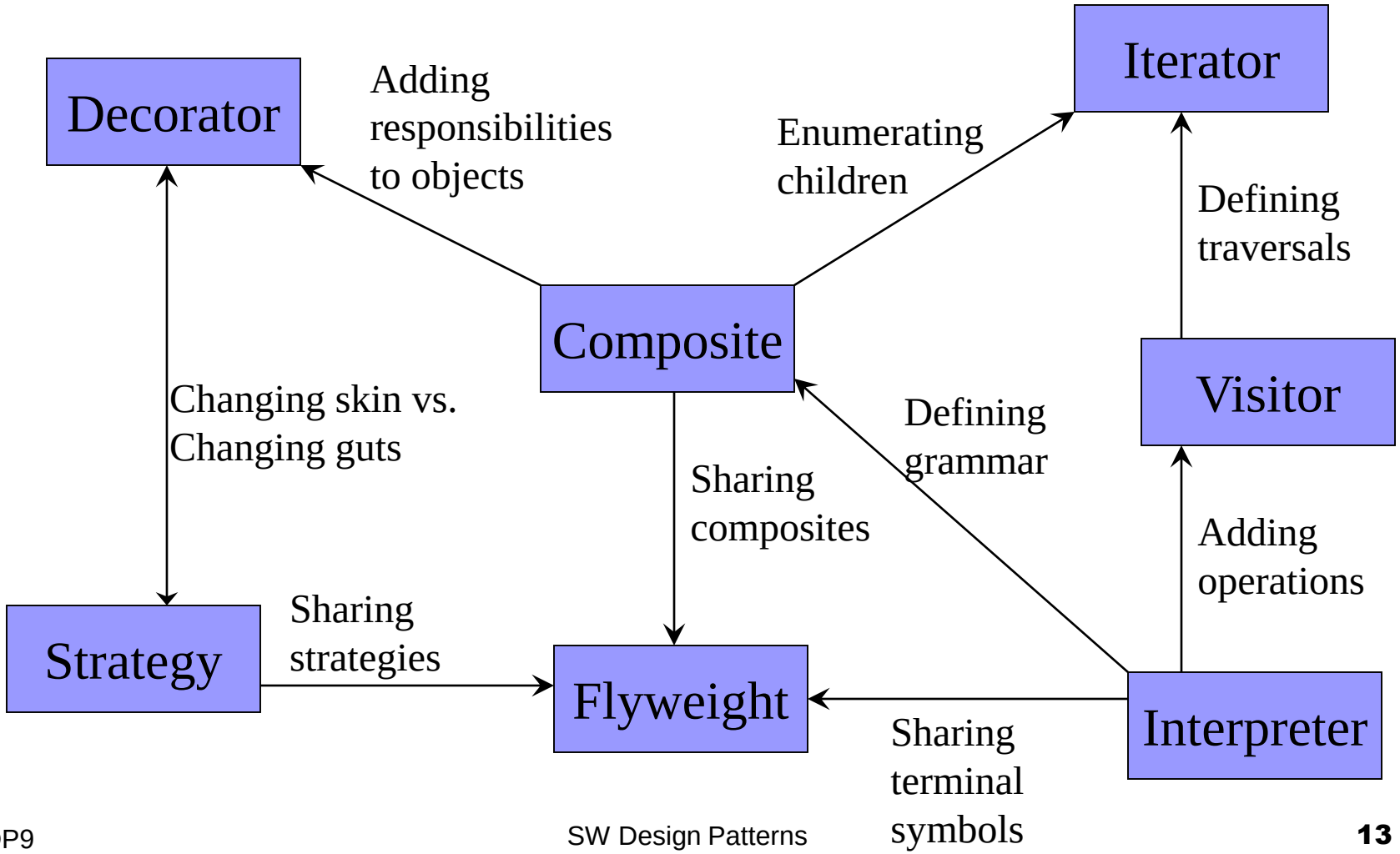
Proxy vs Decorator vs Chain



Some selected examples

- Composite – Composes objects into trees
- Iterator – Iterates over a set
- Visitor – Operates on objects in a set
- Interpreter – Interprets language
- Flyweight – Shares objects
- Decorator – Adds functionality
- Strategy – Isolates algorithm

Gamma's relationships



Why relations?

- Describe patterns containing other patterns
  - E.g. Visitor – Iterator
- Find a pattern, similar patterns
  - E.g. Strategy – Decorator
- Combinations as bigger building blocks
  - E.g. Composite – Iterator

Zimmer's relations

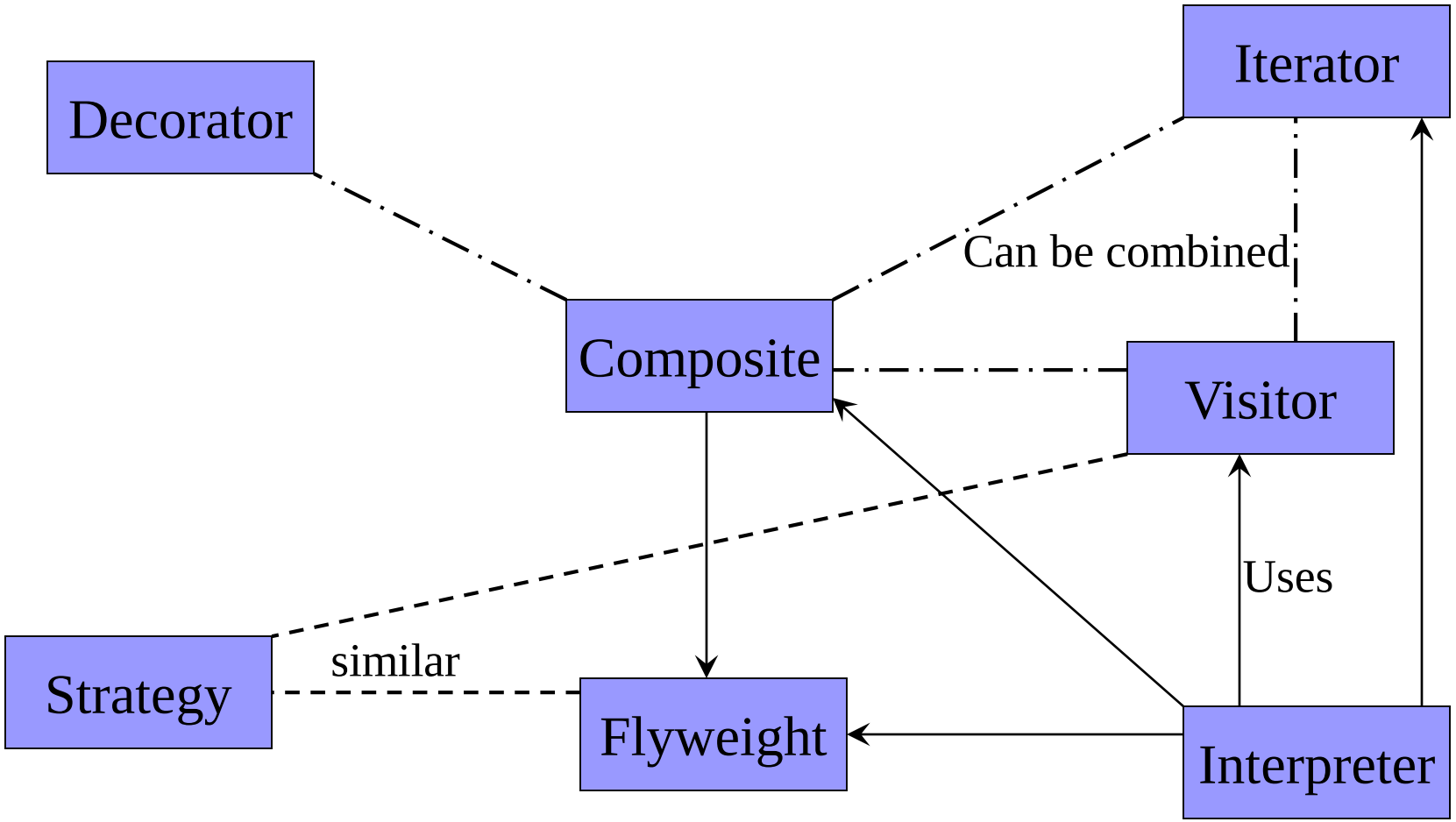
- X uses Y in its solution
  - E.g. Interpreter – Iterator
- X is similar to Y
  - E.g. Visitor – Strategy
- X can be combined with Y
  - E.g. Iterator - Visitor

Source: W. Zimmer. "Relationships Between Design Patterns"In *Pattern Languages of Program Design*, J. O. Coplien and D. C. Schmidt (eds.), Reading, MA: Addison-Wesley, 1995, pp. 345

”Why” using Zimmer’s relations

- X *uses* Y in its solution
  - Describe patterns containing other patterns
  - Combinations as bigger building blocks
- X *is similar to* Y
  - Find alternative patterns
- X *can be combined with* Y
  - Combinations as bigger building blocks
  - Find patterns

Zimmer's relationships



Zimmer's layering

By arranging the patterns along the relationships *uses, is similar to,* and *can be combined with,* Zimmer identifies three different layers:

A. **basic design patterns and techniques** – Singleton, Iterator, Mememto, Façade, Mediator, Template method, Flyweight
B. **design patterns for typical software problems** – Abs. factory, Factory method, Prototype, Builder, Observer, Bridge, Adapter, Strategy, State, Command, Composite, Decorator, Proxy, Chain of resp., Visitor
C. **design patterns specific to an application domain** - Interpreter

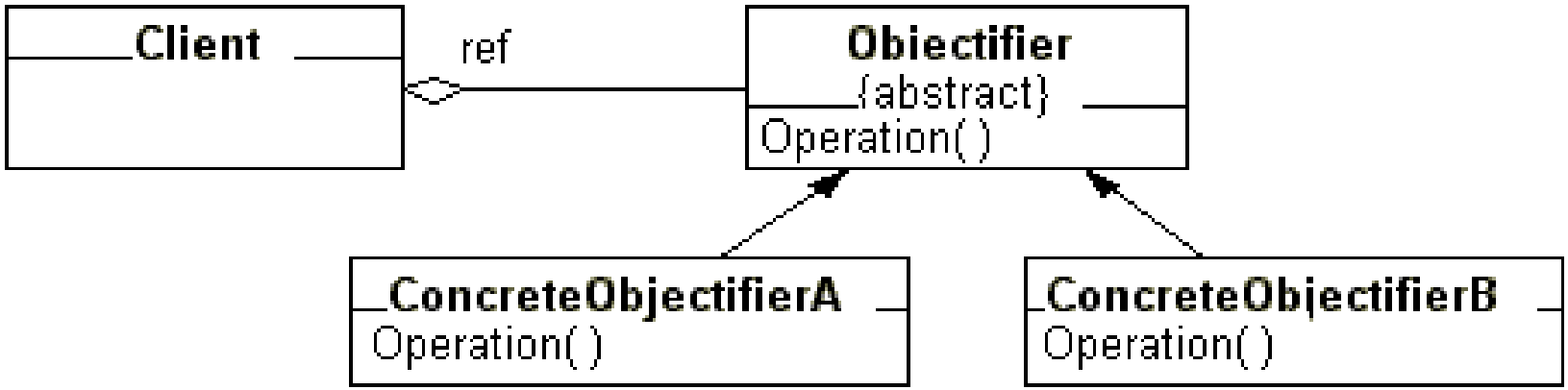
Objectifier

- ”What is it?” vs. ”What does it do?”
  - Normally “What is it?” are classes
  - “What does it do” are functions

- Objectifier is intended to “objectify similar behaviour in additional classes, so that clients can vary such behaviour independently from other behaviour”

Objectifier as a generalization of those patterns

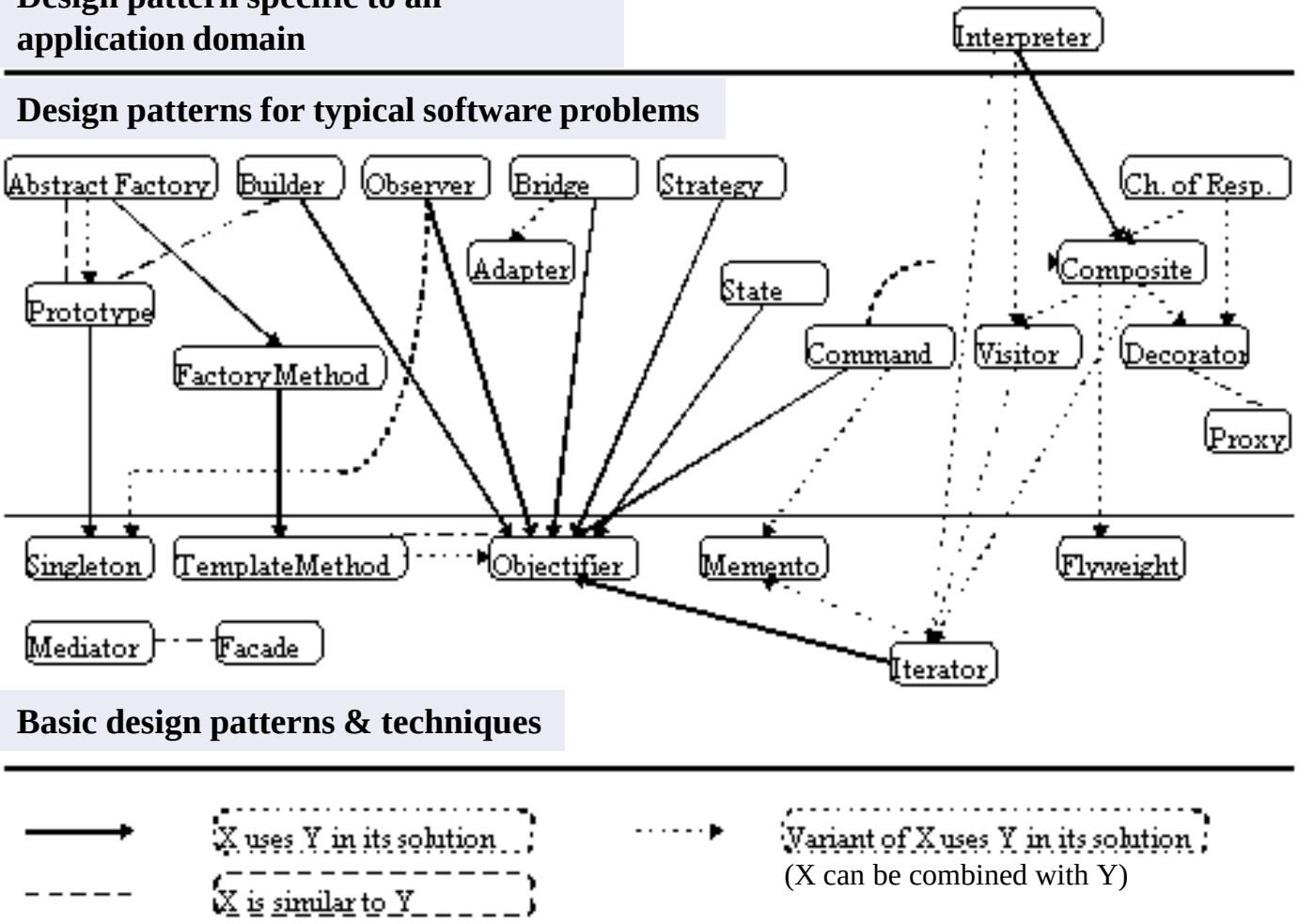
- The Objectifier pattern is related to several other patterns
- Zimmer points out that Objectifier is a generalization of those patterns which objectify behaviour



Objectifier

- Intent
  - Objectify similar behaviour
  - Objects represents behaviour or properties, but not concrete objects
- Motivation
  - Widely used, e.g. Bridge, Command, Builder, Iterator…

Zimmer's layering with Objectifier



Source: http://www8.cs.umu.se/~jubo/ExJobbs/MK/patterns.htm

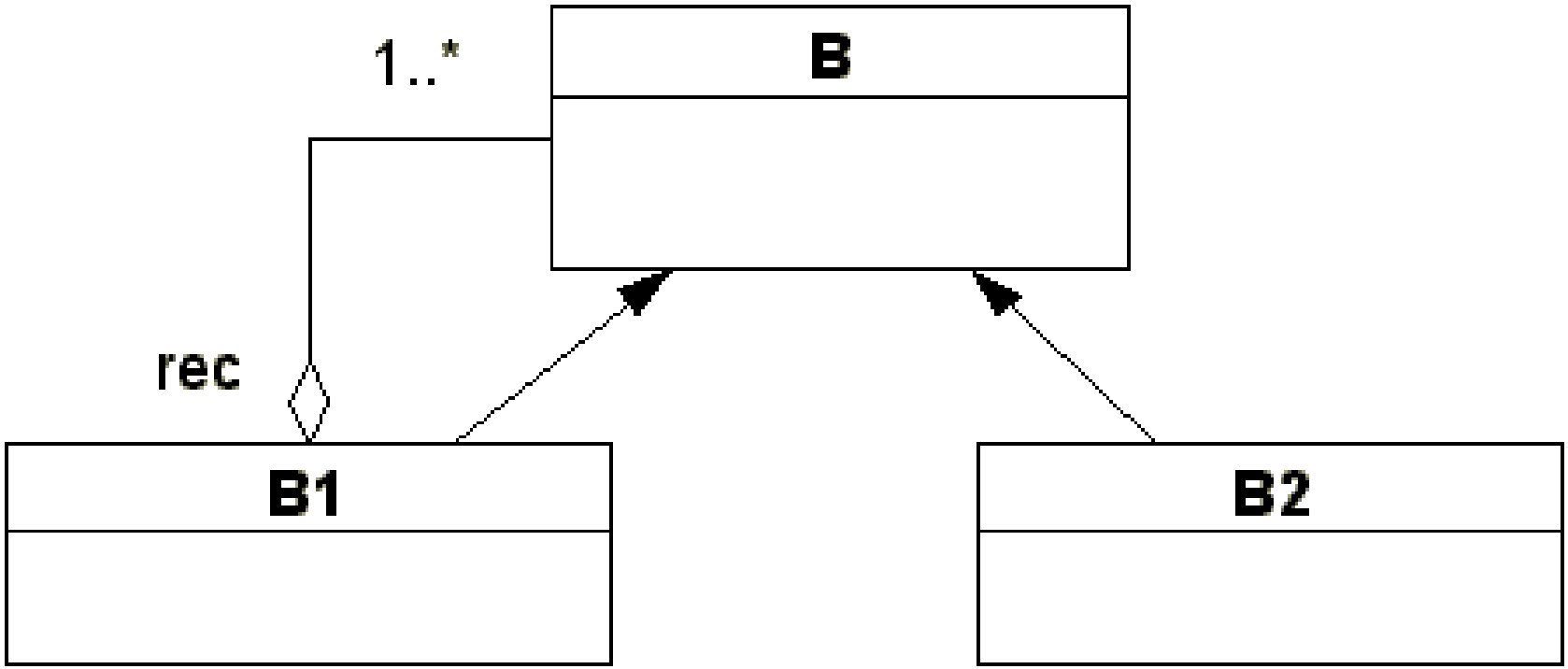
Pree divides the patterns from a structural point of view:

- **basic inheritance and interaction patterns** - patterns that encompass primarily the basic modelling OO capabilities
- **patterns for structuring OO software systems** - describe how a group of classes supports the structuring of software systems.
- **patterns based on recursive structures** - allow recursive building of hierarchies - a subclass has a reference to itself or to a superclass.
- **patterns relying on abstract coupling** - based on abstractly coupled classes. The concept of abstract coupling is when an object has a reference to an abstract class.
- **patterns related to the MVC-framework**

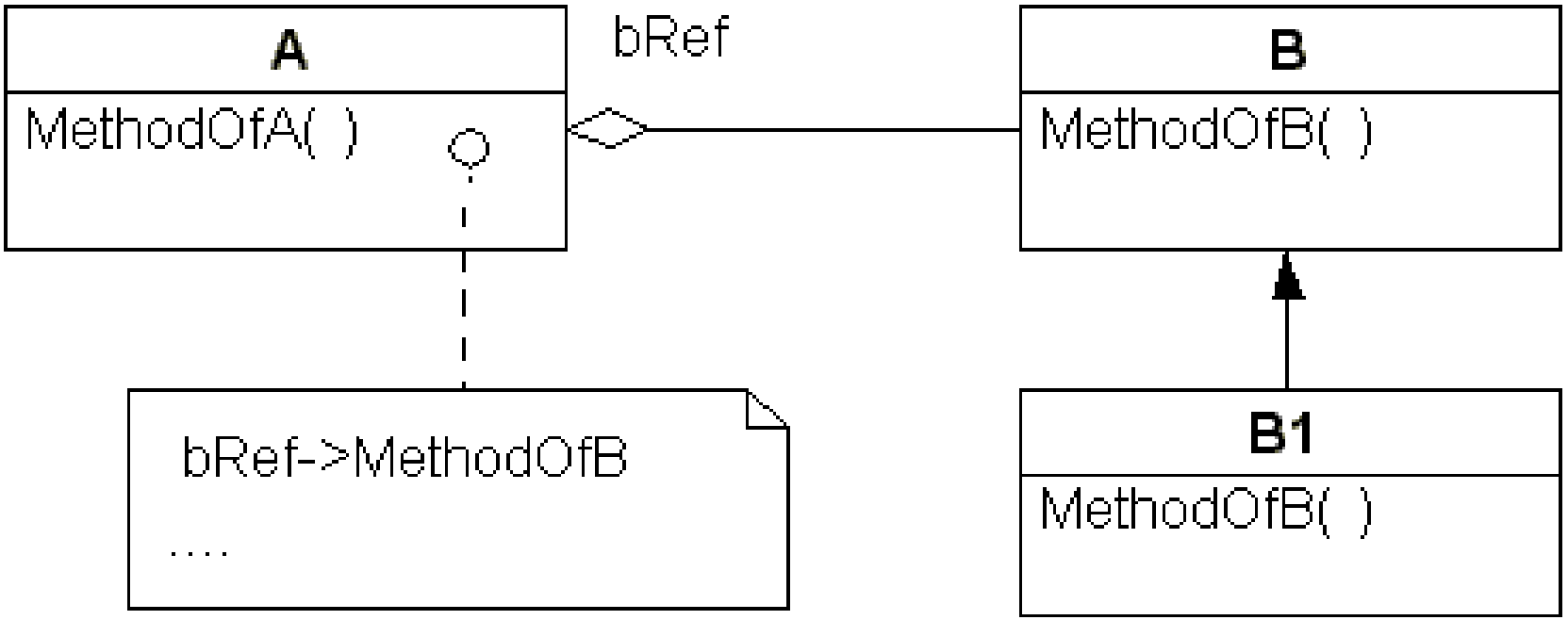
Pree. Design Patterns for Object-Oriented Software

Patterns based on recursive structures

- allow recursive building of hierarchies
- a subclass has a reference to itself or to a superclass



Patterns relying on abstract coupling - an object has a reference to an abstract class



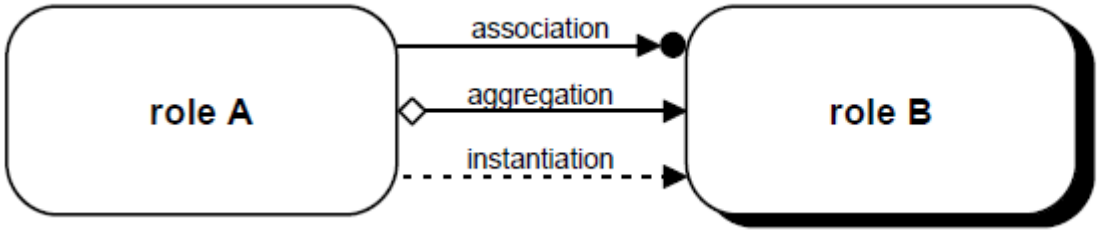
Riehle's composite patterns

- Not the Composite Pattern!
- Compose several patterns
  - Form a larger structure
  - Higher level of abstraction
  - Benefit from synergies in their implementation

Source: Dirk **Riehle**. A Role-Based Design Pattern Catalog of Atomic and Composite Patterns Structured by Pattern Purpose. Ubilab Technical Report, 1997

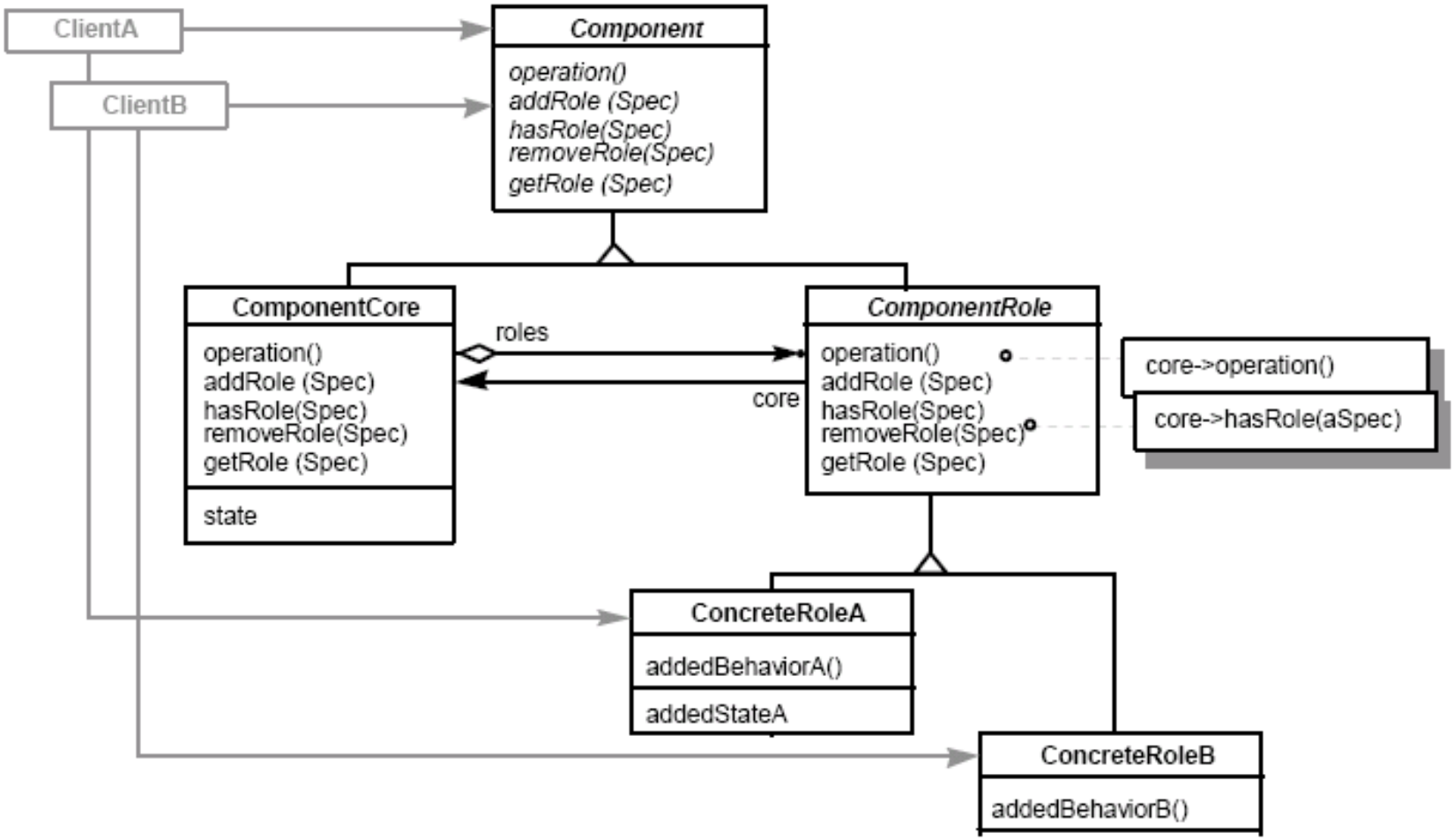
Roles

- A role diagram describes the roles objects play in a collaboration.
- A role defines the abstract state and behavior of an object in the collaboration. It can be expressed formally as a role protocol, for example using any adequate type or interface notation.
- The actual definition of the role is based on what the other roles in the collaboration require from it in order to achieve the joint purpose.



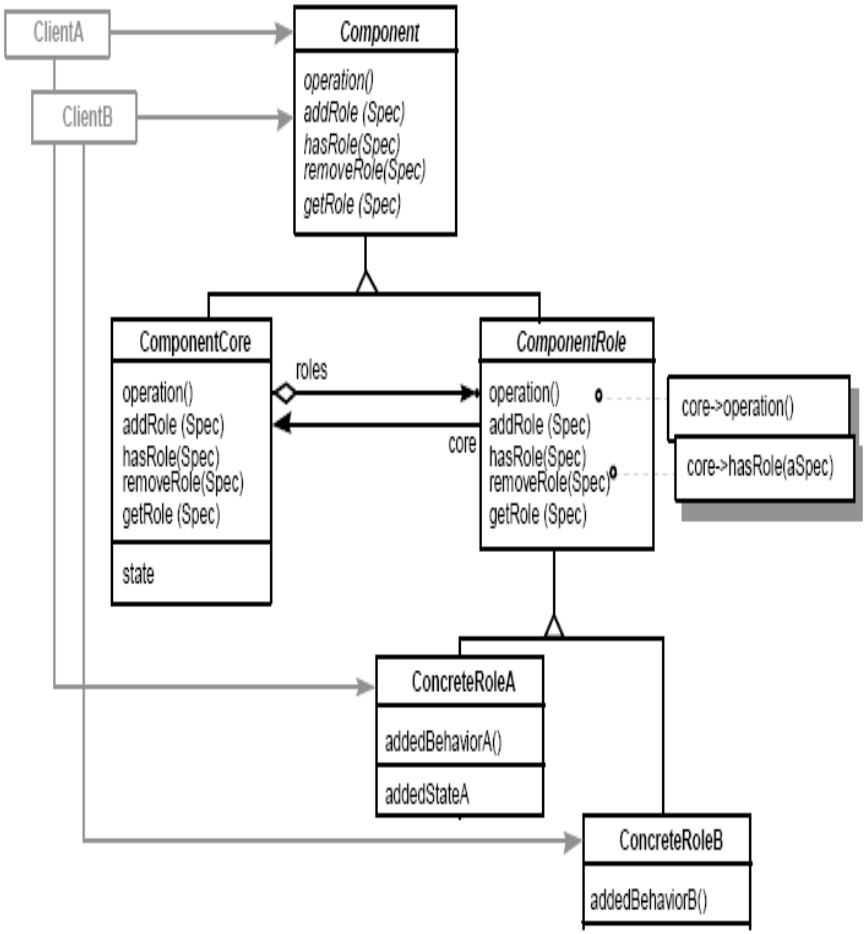
a shadow indicates any number of roles with different role protocols (as opposed to a cardinality of n)

Structure of the Role Object Pattern



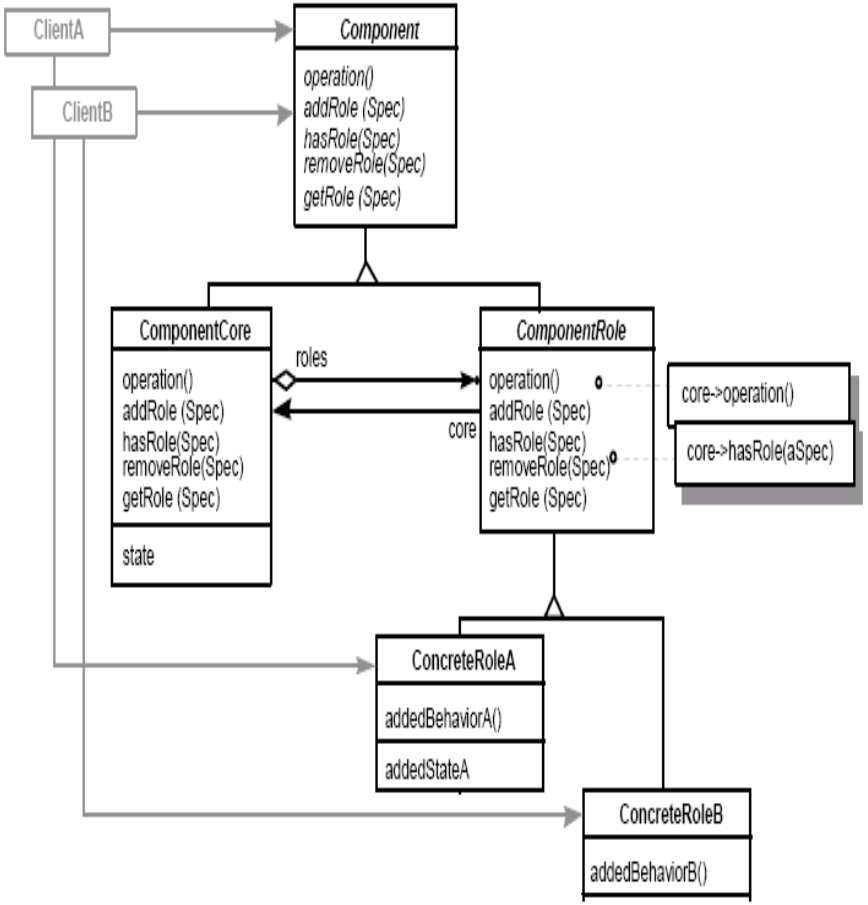
Participants 1/2

- *Component (Customer)*
  - - models a particular key abstraction by defining its interface;
  - - specifies the protocol for adding, removing, testing and querying for role objects.
  - A Client supplies a specification for a ConcreteRole subclass. In the simplest case, it is identified by a string.
- *ComponentCore (CustomerCore)*
  - - implements the Component interface including the role management protocol;
  - - creates ConcreteRole instances;
  - - manages its role objects.

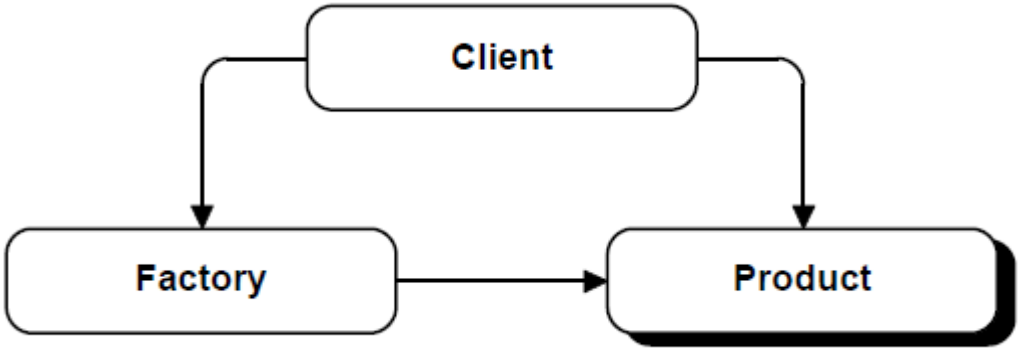


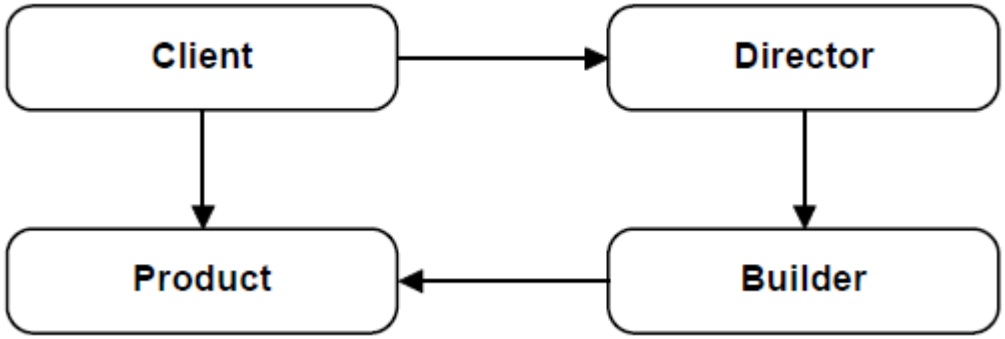
Participants 2/2

- *ComponentRole (CustomerRole)*
  - - stores a reference to the decorated ComponentCore;
  - - implements the Component interface by forwarding requests to its core attribute.
- *ConcreteRole (Investor, Borrower)*
  - - models and implements a context-specific extension of the Component interface;
  - - can be instantiated with a ComponentCore as argument.



Role diagram of the patterns Factory Method and Builder





Bureaucracy composite pattern

- Consists of
  - Composite
  - Observer
  - Chain of Responsibility
- Uses roles to find synergies

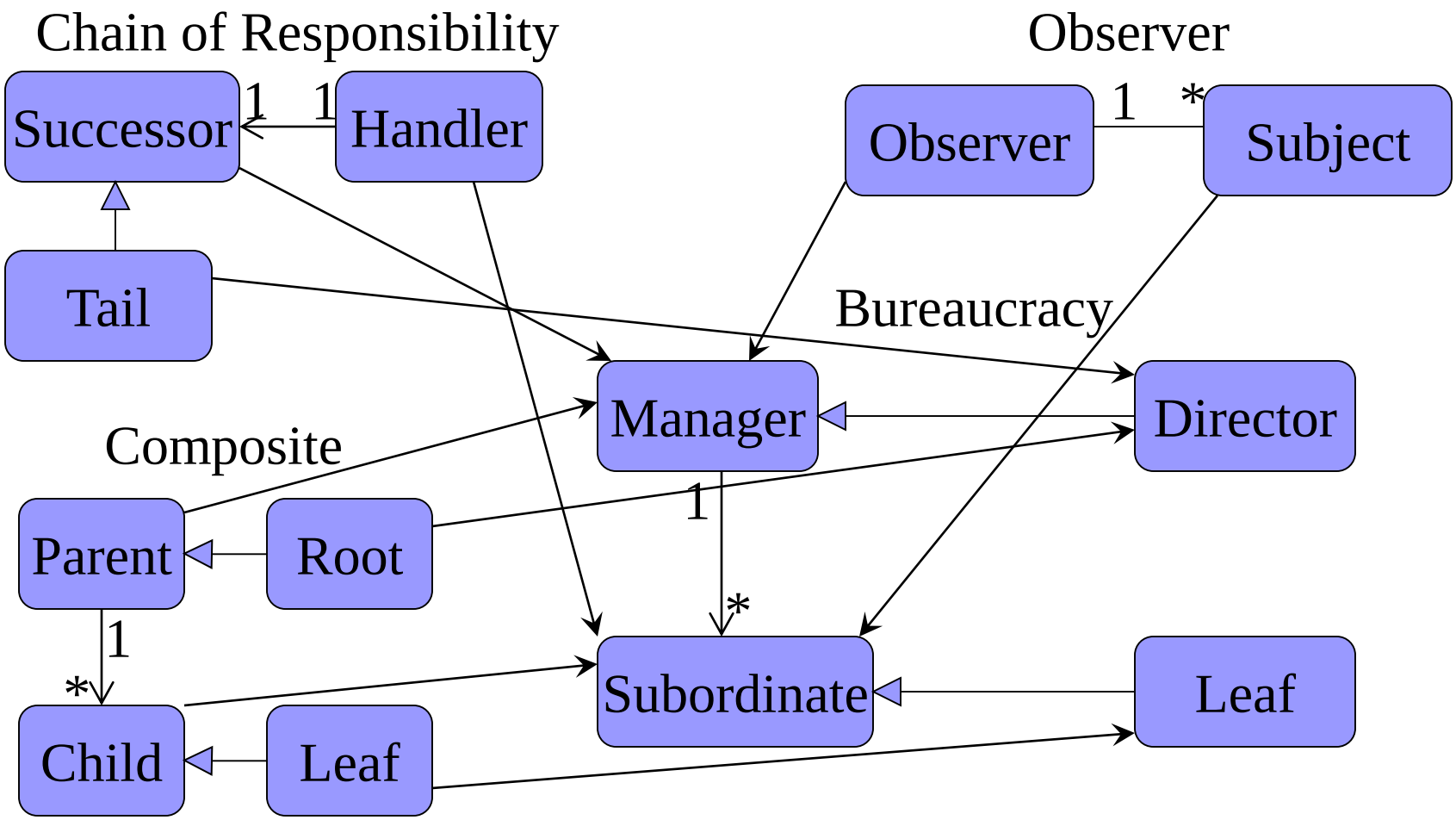
Dirk **Riehle**. "Bureaucracy - A Composite Pattern.“, Proc. of EuroPLoP '96, 1996.

The Bureaucracy pattern in brief

- a recurring design theme used to implement hierarchical object structures which allow interaction with every level of the hierarchy and maintain their inner consistency themselves.
- It is a composite pattern which is based on the Composite, Mediator, Chain of Responsibility and Observer pattern.
- Composite patterns require new presentation and modeling techniques since their complexity makes them more difficult to approach than non-composite patterns.
- Riehle use role diagrams to present the Bureaucracy pattern and to explore its design and implementation space. Role diagrams have proved to be very useful to get a grip on this complex pattern.

Source: Riehle D.. Bureaucracy. In Robert C. Martin, Dirk Riehle, Frank Buschmann (editors), *Pattern Languages of Program Design 3*, Addison-Wesley, Reading, Massachusetts, 1997.

Bureaucracy roles



Roles: Pro's and Con's

- Pros
  - Higher abstraction
  - Syngergies can be found
  - Leaves several possible implementations
- Cons
  - Describes the roles in the problem
  - Does not describe the solution
  - Leaves several possible implementations

Conclusion

- Several ways to relate patterns; most important is **”why”**
- Patterns can be classified in different ways; even the same patterns may be classified differently – for example Zimmer’s classification of the GOF design patterns.
- Roles might be useful
  - Especially combined with class diagrams