A CLASSIFICATION OF OBJECT-ORIENTED DESIGN PATTERNS

Magnus Kardell (excerpted from)

Umeå University

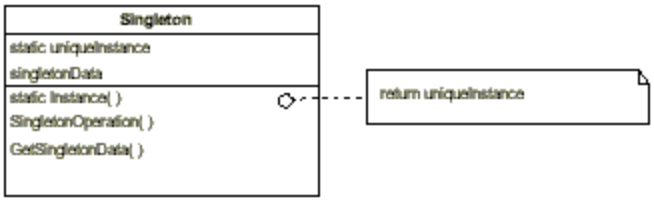
Structural mechanisms

Structural criteria can be found in the approaches by Pree 9, Gamma 3 and Coad 4. Combining the brings these categories or groups of patterns:

- patterns based on abstract coupling
- patterns based on recursive structures
- basic interaction and inheritance patterns
- patterns for structuring object-oriented software systems
- patterns related to the MFC-framework
- aggregate patterns
- interaction patterns
- class patterns
- object patterns

These categories can not be used together as they intersect each other. I will not investigate interpretations of these groups further because I do not believe that structural mechanisms is a good criterion. The difficult part concerning structural mechanisms is that it is very hard to define categories. Pree 9 investigates patterns structurally from the aspects of hook and template methods and defines patterns at a higher level - metapatterns.

APPENDIX A - DESIGN PATTERN CATALOG

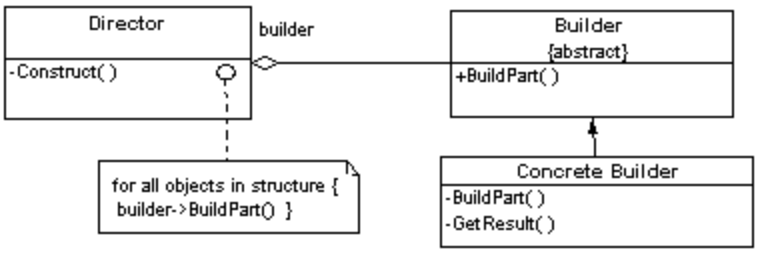


Singleton

Intent: Ensure that a class only has one instance, and provide a global point of access to it.

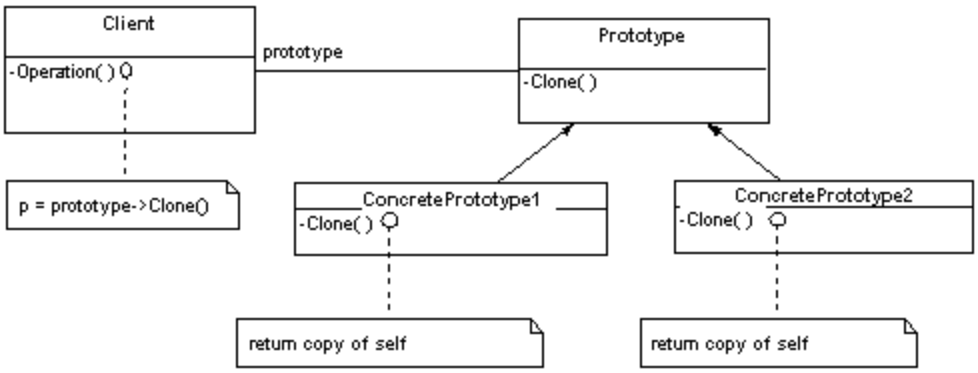
Classification: Creational, Object

Builder



Intent: Separate the building of a complex object from its representation so that the same construction process can create different representations.

Classification: Creational, Object

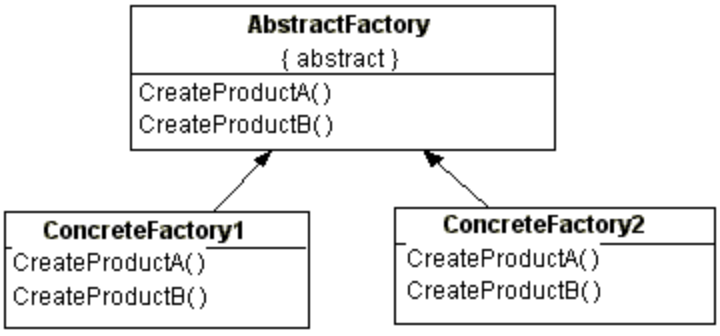


Prototype

Intent: Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Classification: Creational, Object

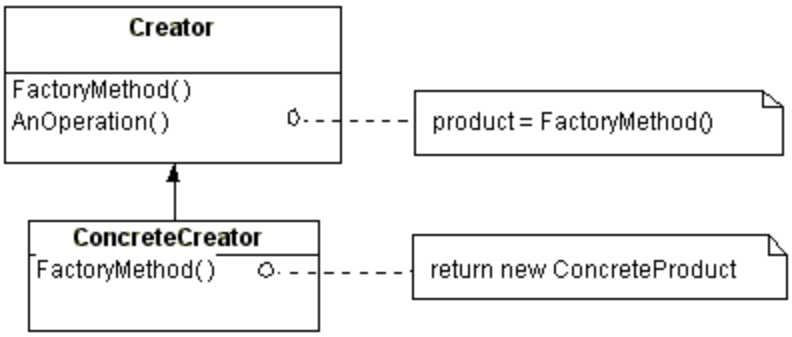
Abstract Factory



Intent: Provide an interface for creating families of related objects without specifying their concrete classes.

Classification: Creational, Class

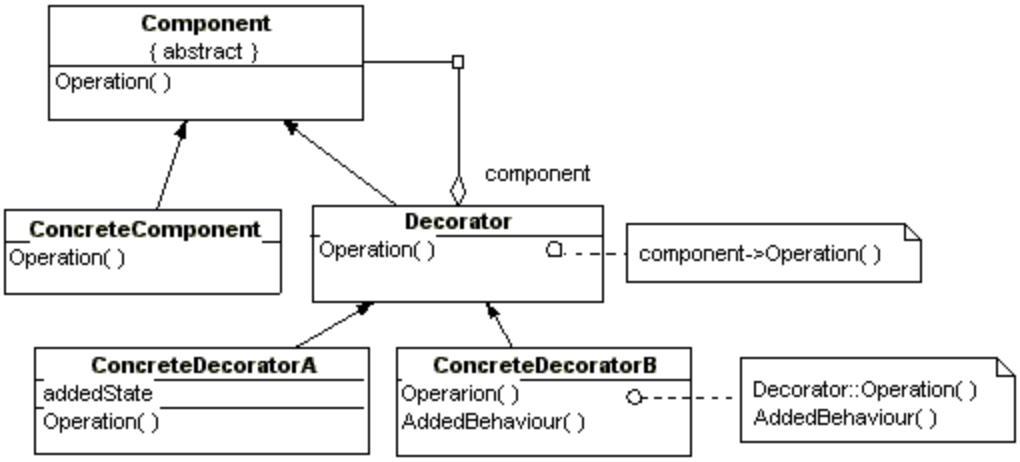
Factory Method



Intent: Define an interface for creating an object, but let subclasses decide which class to instantiate.

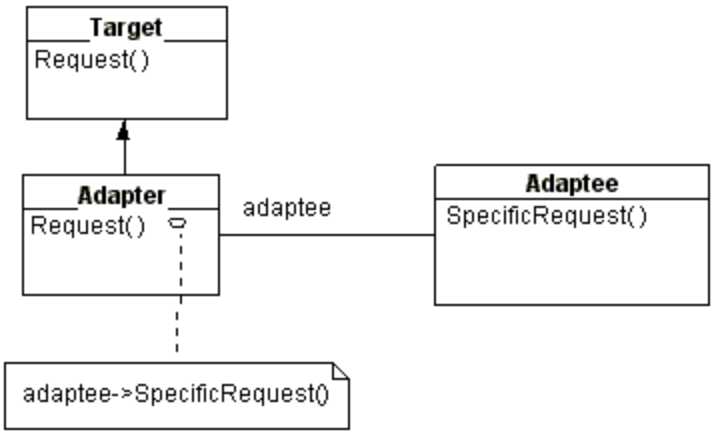
Classification: Creational, Class

Decorator



Intent: Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extended functionality.

Classification: Structural, Object

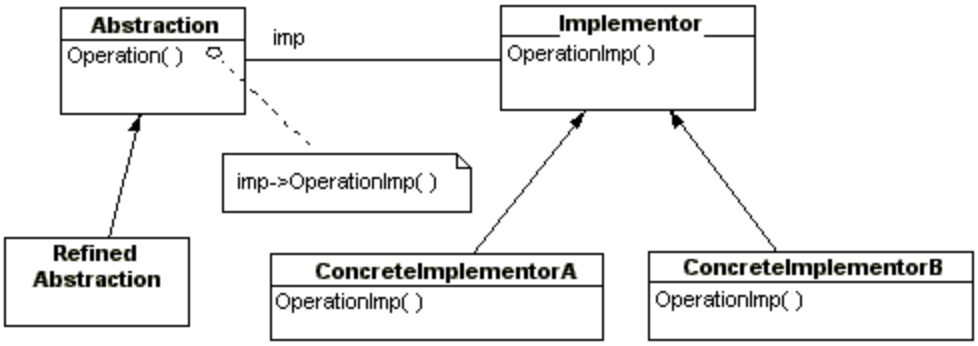


Adapter(object)

Intent: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

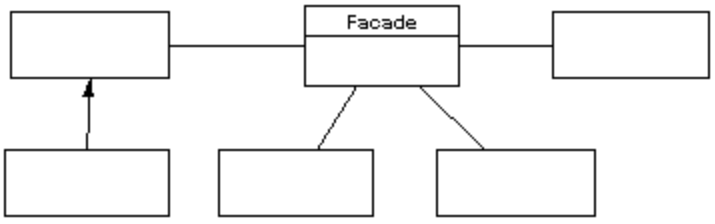
Classification: Structural, Object

Bridge



Intent: Decouple an abstraction from its implementation so that the two can vary independently.

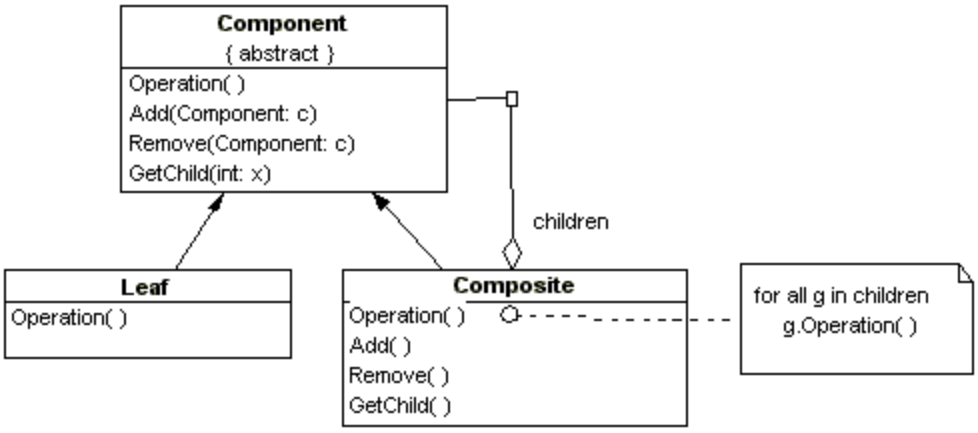
Classification: Structural, Object



Façade

Intent: Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

Classification: Structural, Object

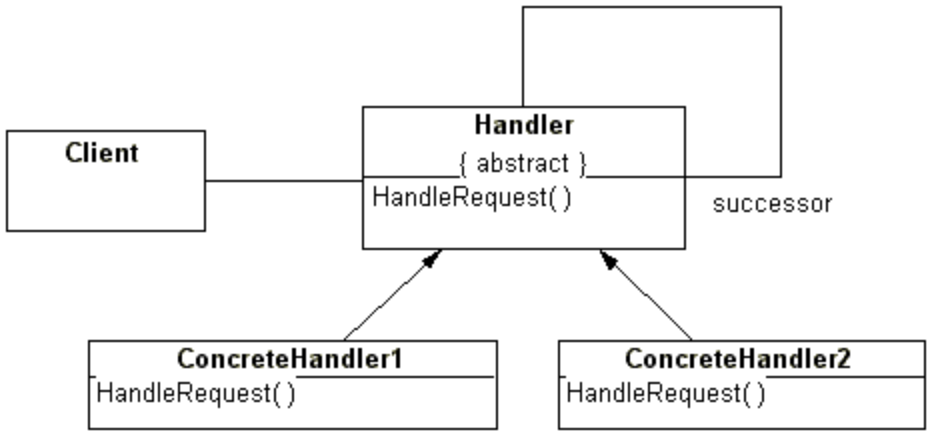


Composite

Intent: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Classification: Structural, object

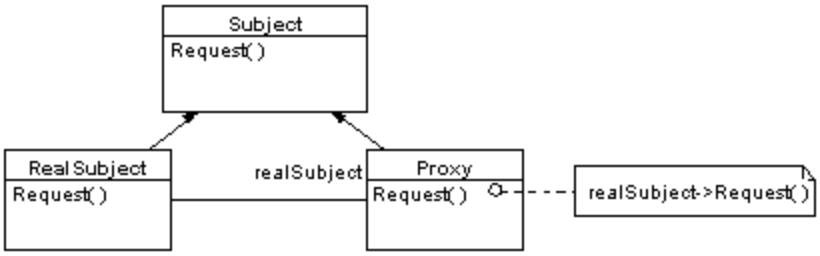
Chain of responsibility



Intent: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Classification:

Behavioural, object

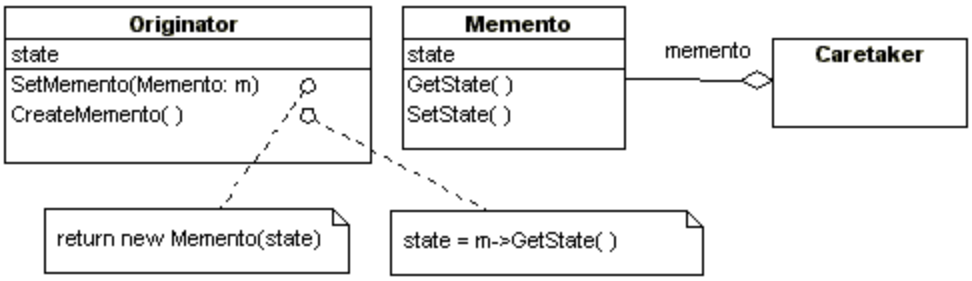


Proxy

Intent: Provide a surrogate or placeholder for another object to control access to it.

Classification: Structural, object

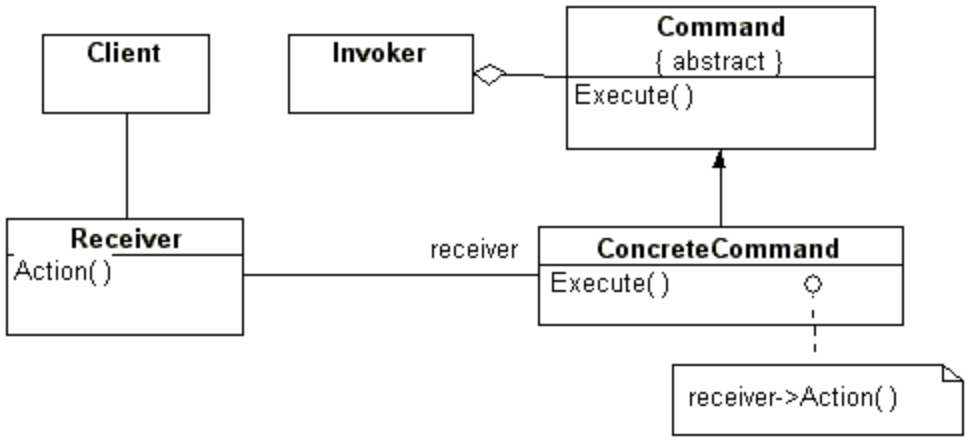
Memento



Intent: Without violating encapsulation, capture an externalize an object's internal state so that the object can be restored to this state later.

Classification: Behavioural, object

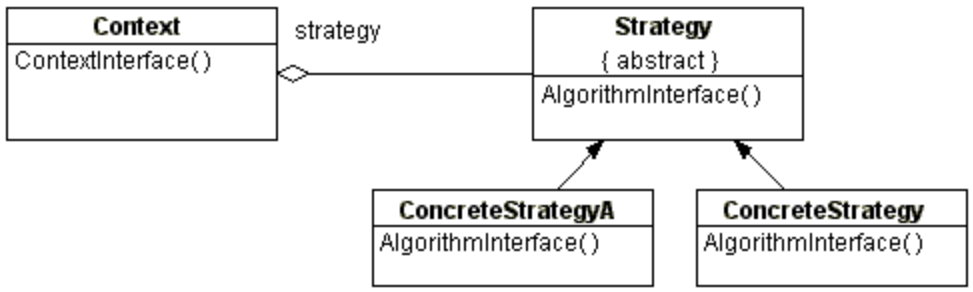
Command



Intent: Encapsulate a request as an object thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Classification: Behavioural, object

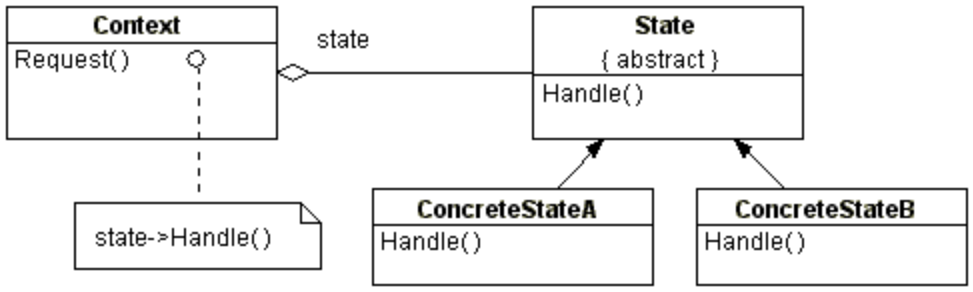
Strategy



Intent: Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.

Classification: Behavioural, object

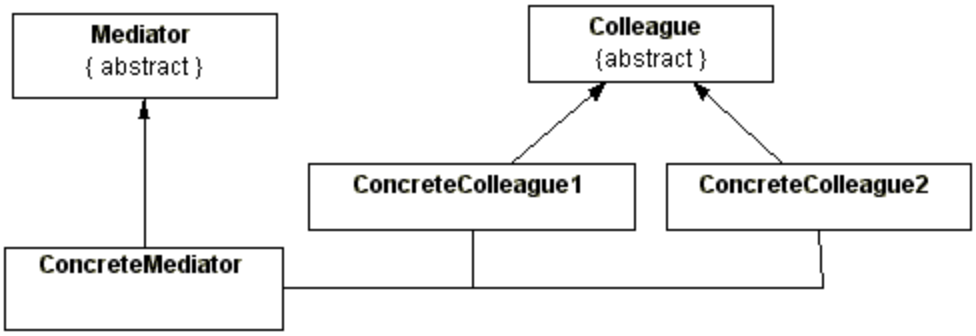
State



Intent: Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class.

Classification: Behavioural, object

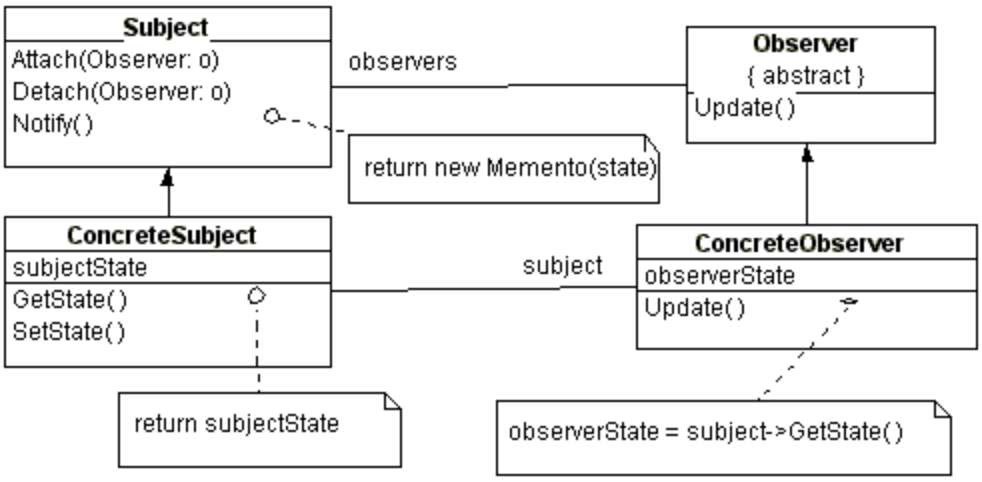
Mediator



Intent: Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling between by keeping objects from referring to each other explicitely, and it lets you vary their interaction independently.

Classification: Behavioural, object

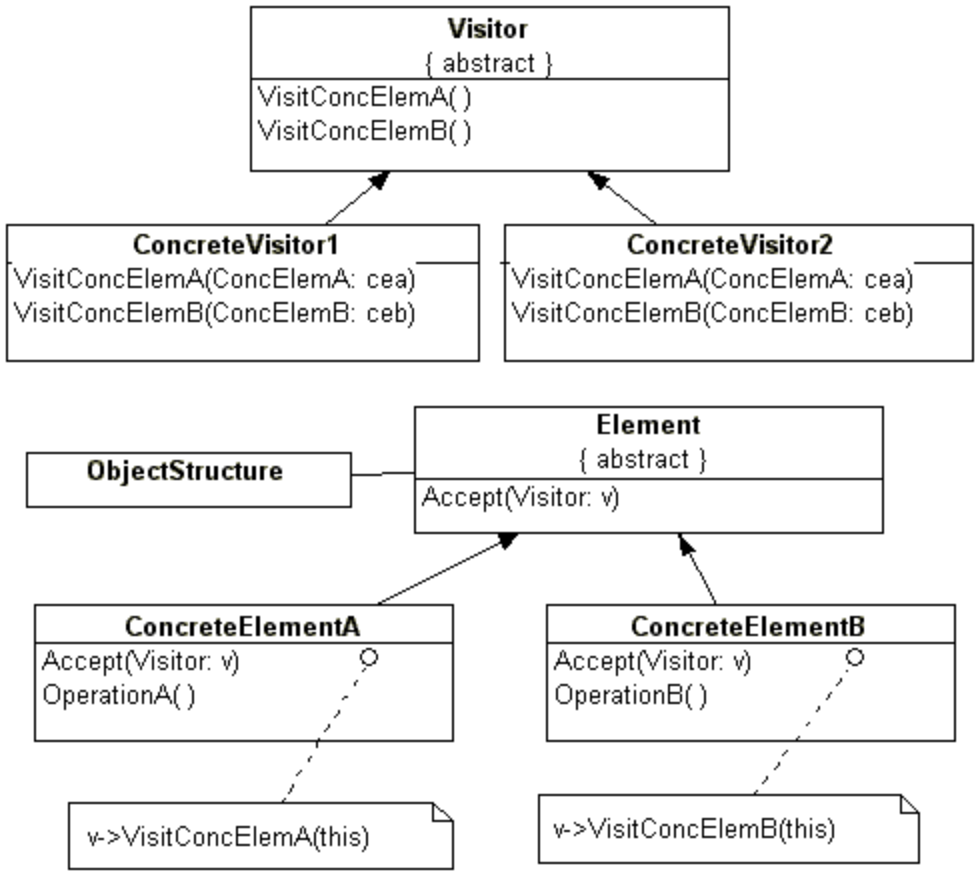
Observer



Intent: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Classification: Behavioural, object

Visitor

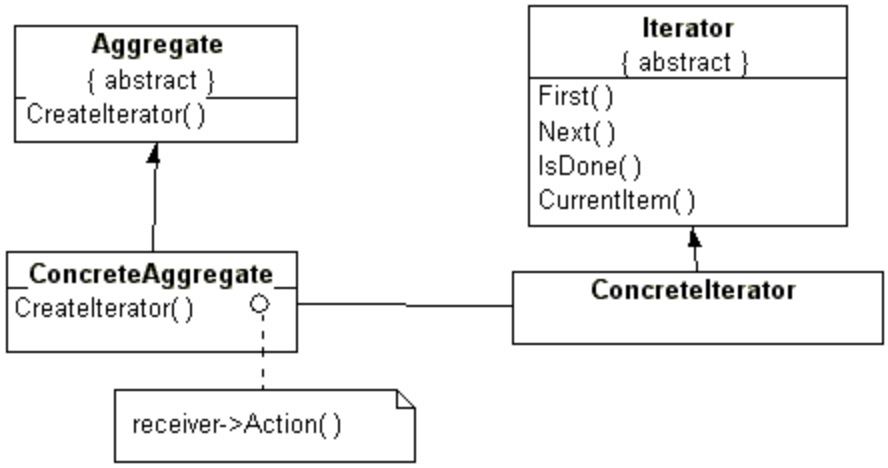


Intent: Represent an operation to be performed on the elements of an object structure. Visitor lets you define a

new operation without changing the classes of the elements on which it operates.

Classification: Behavioural, object

Iterator

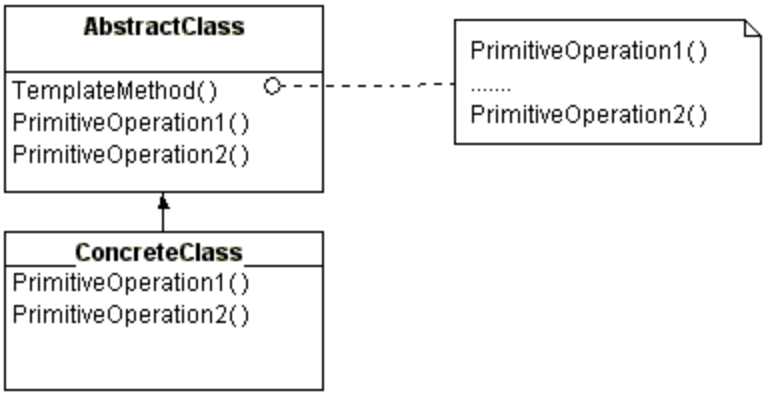


Intent:

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Classification:

Behavioural, object



Template method

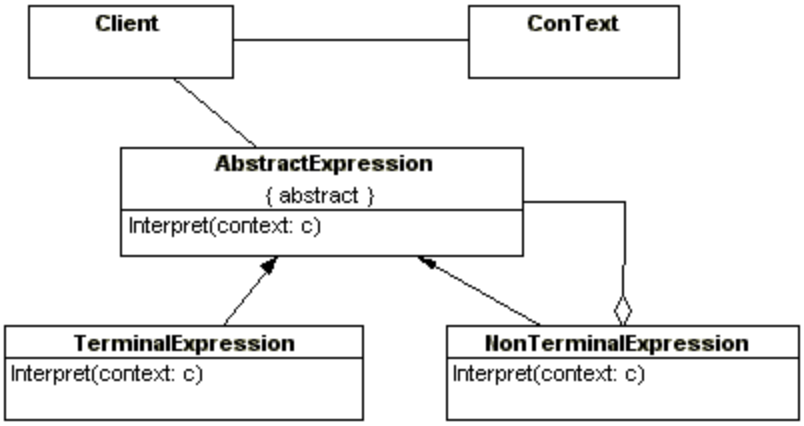
Intent:

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. TM lets subclasses redefine certain steps of an algorithm without changing the algorithms structure.

Classification:

Behavioural, class

Interpreter



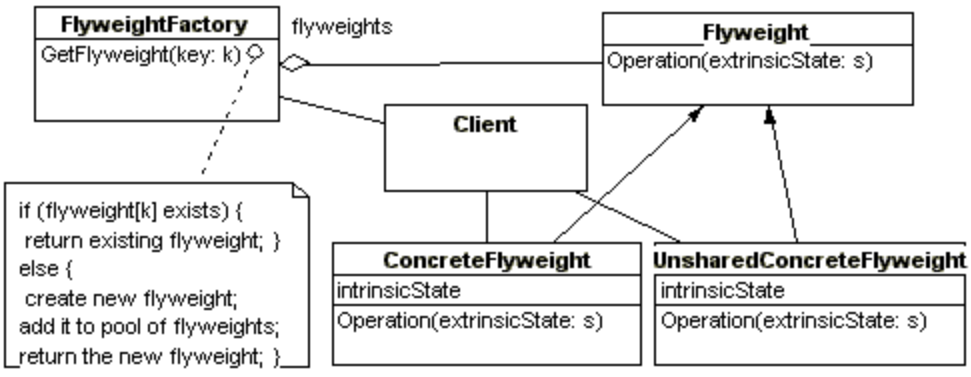
Intent:

Given a language, define a representation for its grammar, along with an interpreter that uses the representation to interpret sentences in the language.

Classification:

Behavioural, class

Flyweight



Intent: Use sharing to support large numbers of fine-grained objects efficiently.

Classification: Structural, object

APPENDIX B - OBJECT MODELS

To keep down the length of this appendix I will present a selected version of Coad's patterns. First I present a full version of the fundamental pattern then I give an overview over the pattern families. The interaction family is covered in greater detail.

The fundamental pattern (full)

#1 “Collection-Worker" Pattern the fundamental pattern

- Collection-worker is the fundamental object-model pattern
- All other object-model patterns are variations on this theme
- **Typical object-connections**

howManycalcForMe calcOverWorkerscalcForMe

howMuchcalcForMe rankWorkersrateMe

- **Other notes**

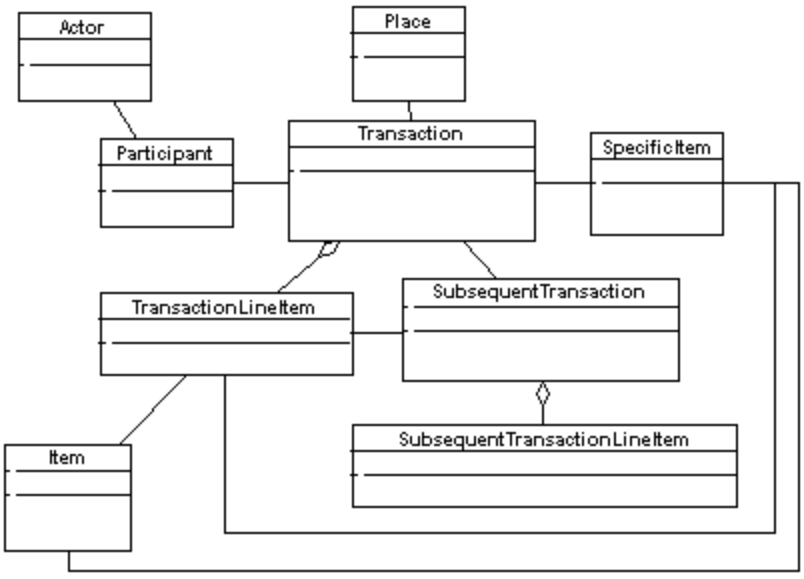
“aboutMe" help one think about what other attributes might be needed

“calcForMe" help one think about what specific calculations might be needed

“ rankMe" helps one think about what ordering or comparrison services might be

“rateMe" helps one think about what self-assessment services might be needed

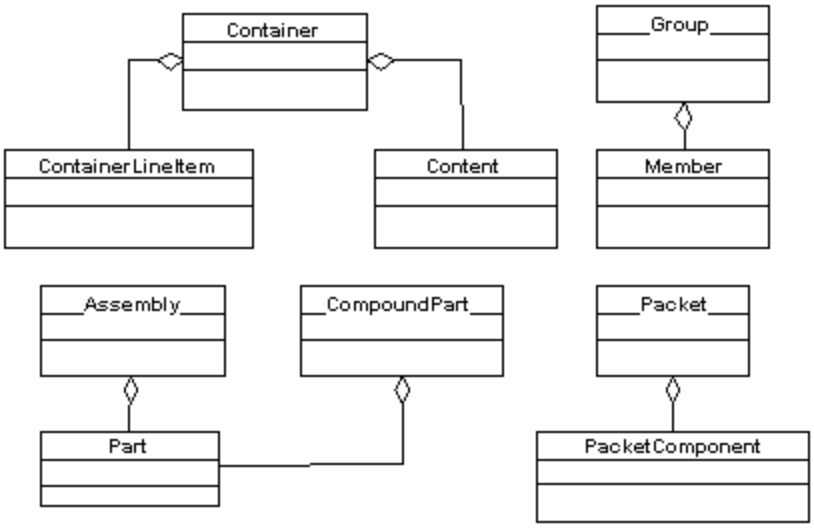
Transaction patterns (overview)



Contains:

- actor-participant
- participant-transaction
- place-transaction
- specific item-transaction
- transaction-transaction line item
- transaction-subsequent transaction
- transaction line item- subsequent transaction line item
- item-line item
- specific item-line item
- item-specific item
- associate-other associate
- specific item-hierarchical item

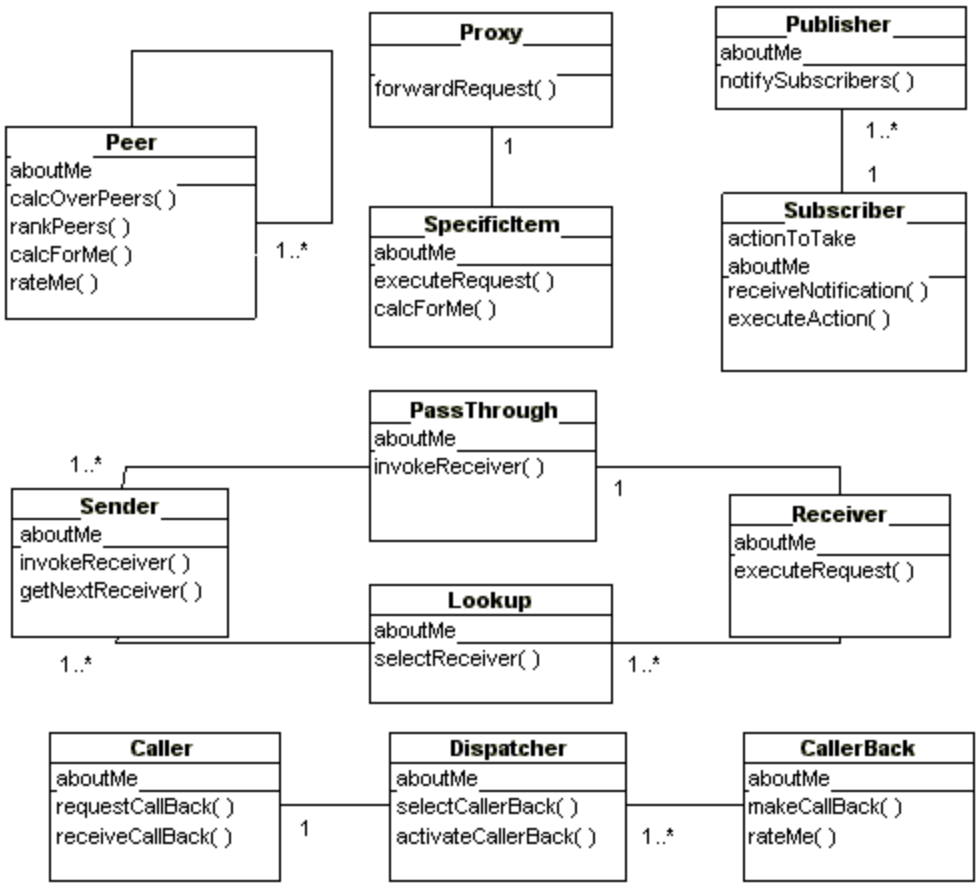
Aggregate patterns (overview)



Contains:

- container- content
- container-container line item
- group-member
- assembly-part
- compound part-part
- packet-packet component

Interaction patterns (overview)



Contains: peer-peer, proxy-specific item, publisher-subscriber, sender- pass through receiver, sender-lookup-receiver, caller-dispatcher-caller back, gatekeeper-request-resource

`http://www.cs.umu.se/~jubo/ExJobbs/MK/patterns.htm`