TACT factory
mobile agency

Design Patterns

Fournisseur **souple** et **réactif** d'applications mobiles innovantes avec une **méthodologie** projet **rigoureuse**.

Présentation

Mickael Gaillard (Architecte logiciel)
mickael.gaillard@tactfactory.com

Yoan Pintas (Project Manager)
yoan.pintas@tactfactory.com

Antoine Cronier (Developper)
antoine.cronier@tactfactory.com

Histoire

- 1994 :
    - Design patterns – elements of reusable software
    - Gang of Four
    - 23 patrons de conceptions

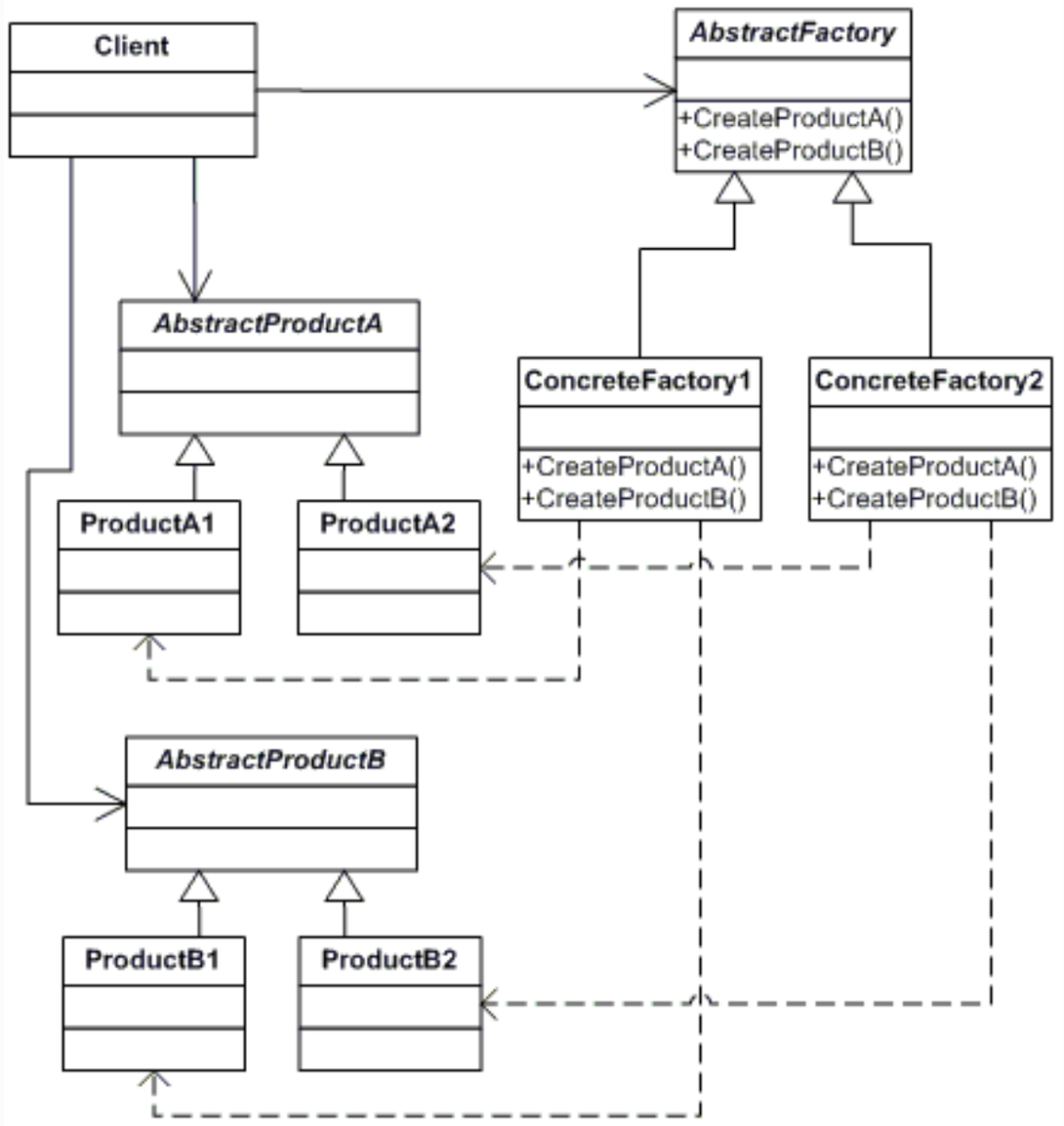
« Chaque patron décrit un problème qui se manifeste constamment dans notre environnement, et donc décrit le cœur de la solution à ce problème, d'une façon telle que l'on puisse réutiliser cette solution des millions de fois, sans jamais le faire deux fois de la même manière »

Les Types de Patrons

- 3 types :
    - construction :
        Définit l'instanciation et la configuration des classes et des objets
- structuraux :
    Définit l'organisation des classes séparant interface et implémentation
- comportementaux :
    Définit l'organisation et la coercition des objets

Creational Patterns

- Abstract Factory
- Use : High
- Allow to create objects without specifying their concrete classes by an interface
- Used classes need to define similar methods and have relationship, their differences are set by the using of those methods

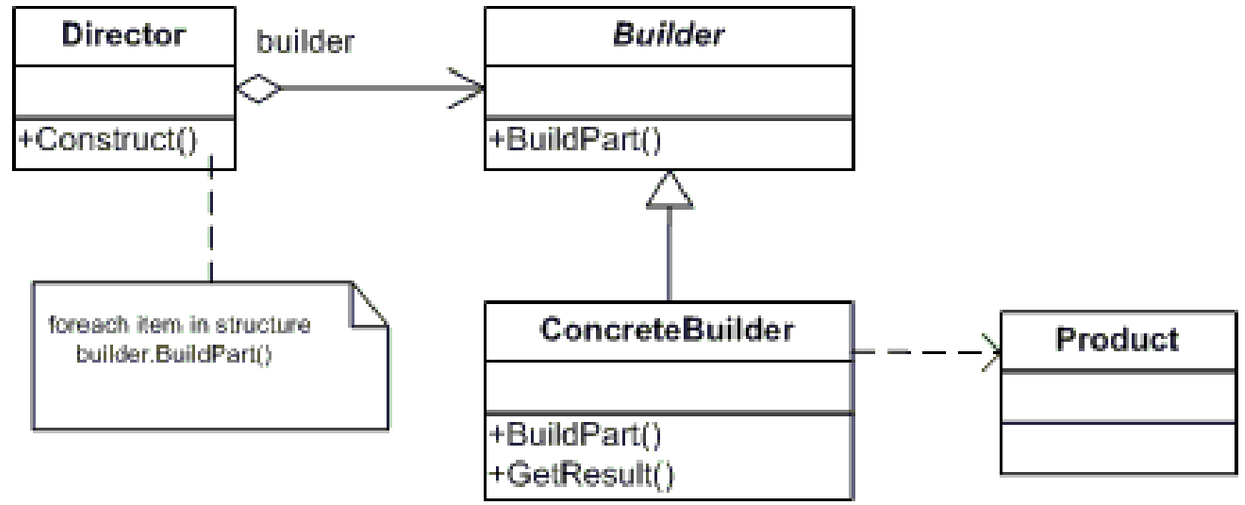


Creational Patterns

- Factory Method
- Implementation explanation :
    - Abstract Factory : set the differents commons operations for the construction
- Concrete Factory : set the real implementation of construction for a determined object
- Abstract Products : set products interfaces
- Concrete Products : set the products to implement
- Client : only use interfaces from abstract factory to implement objects

Creational Patterns

- Builder
- Use : Medium
- Separate construction of complex object from its representation
- Used when a classe get many arguments and most of them are optional



Creational Patterns

- Builder
- Implementation explanation :
    - Builder : specifies an abstract interface for creating parts of a Product object
    - ConcreteBuilder : constructs and assembles parts of the product by implementing the Builder interface defines and keeps track of the representation it creates provides an interface for retrieving the product
    - Director : constructs an object using the Builder interface

Creational Patterns

- Builder
- Implementation explanation :
    - Product : represents the complex object under construction.
ConcreteBuilder builds the product's internal representation and defines the process by which it's assembled includes classes that define the constituent parts, including interfaces for assembling the parts into the final result

Creational Patterns

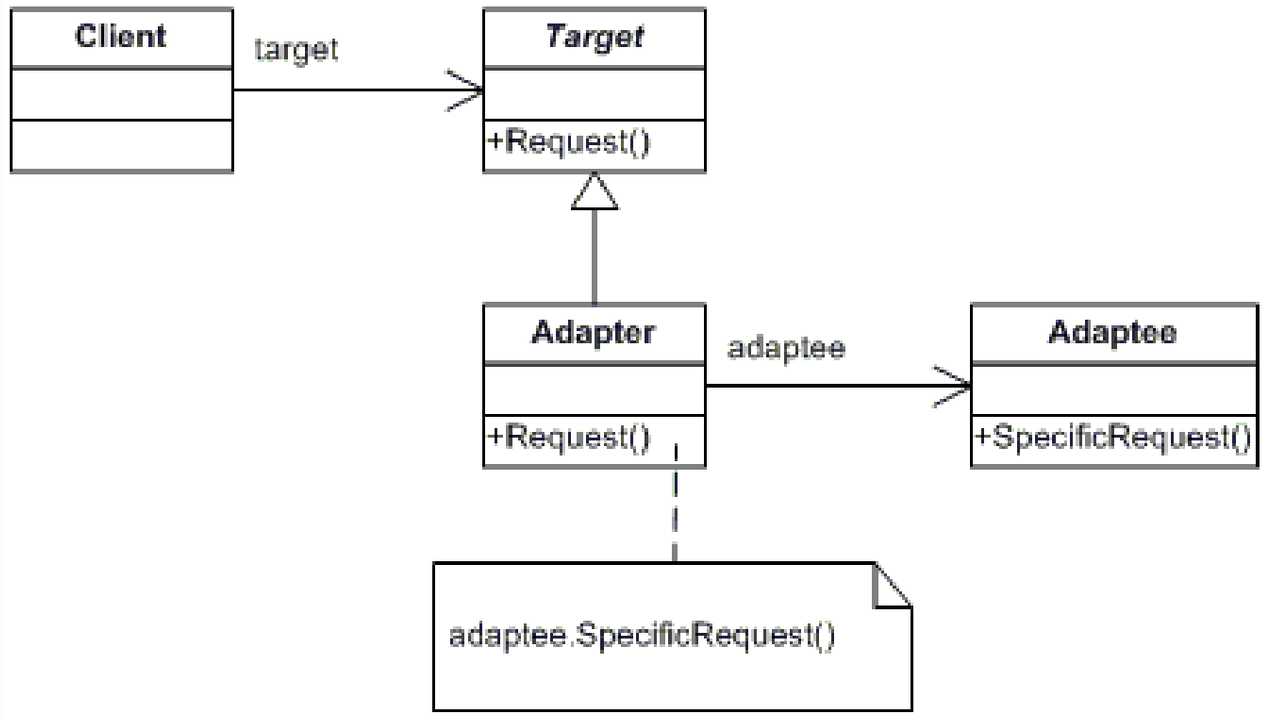
- Singleton
- Use : High
- Ensure a class has only one instance and provide a global point of access to it
- Provide access to object between code that be the same allover application run

Creational Patterns

- Singleton
- Implementation explanation :
    - Singleton : defines an Instance operation that lets clients access its unique instance. Instance is a class operation.
responsible for creating and maintaining its own unique instance.

Structural Patterns

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

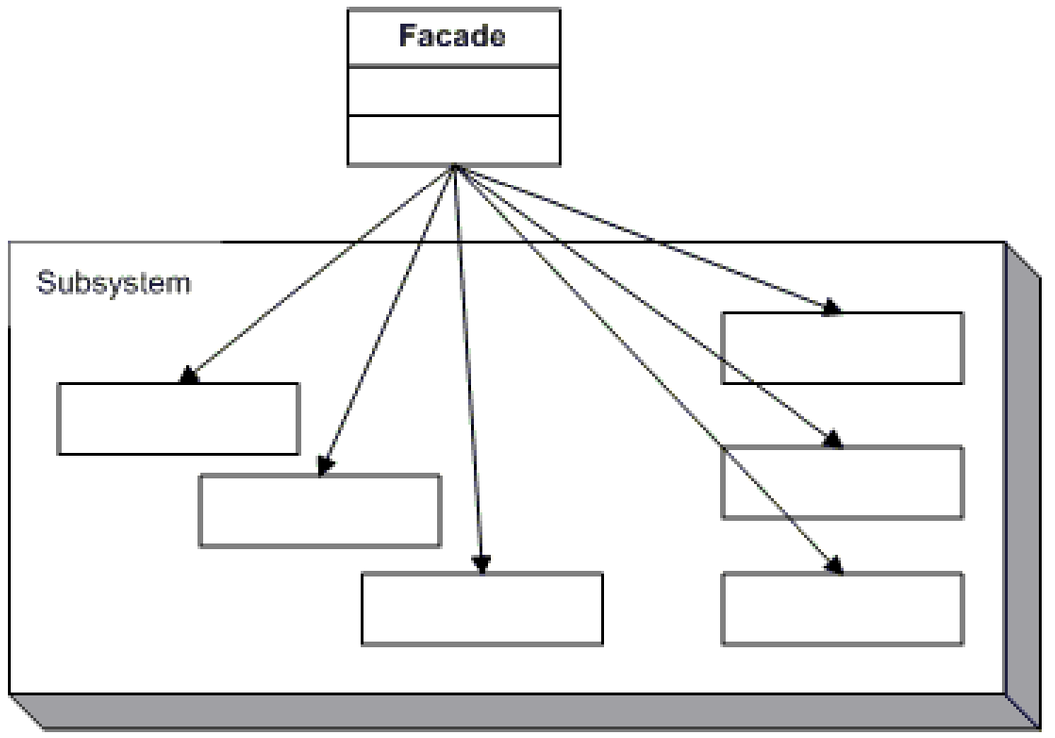


Structural Patterns

- Adapter
- Implementation explanation :
    - Target : defines the domain-specific interface that Client uses.
    - Adapter : adapts the interface Adaptee to the Target interface.
    - Adaptee : defines an existing interface that needs adapting.
    - Client : collaborates with objects conforming to the Target interface.

Structural Patterns

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

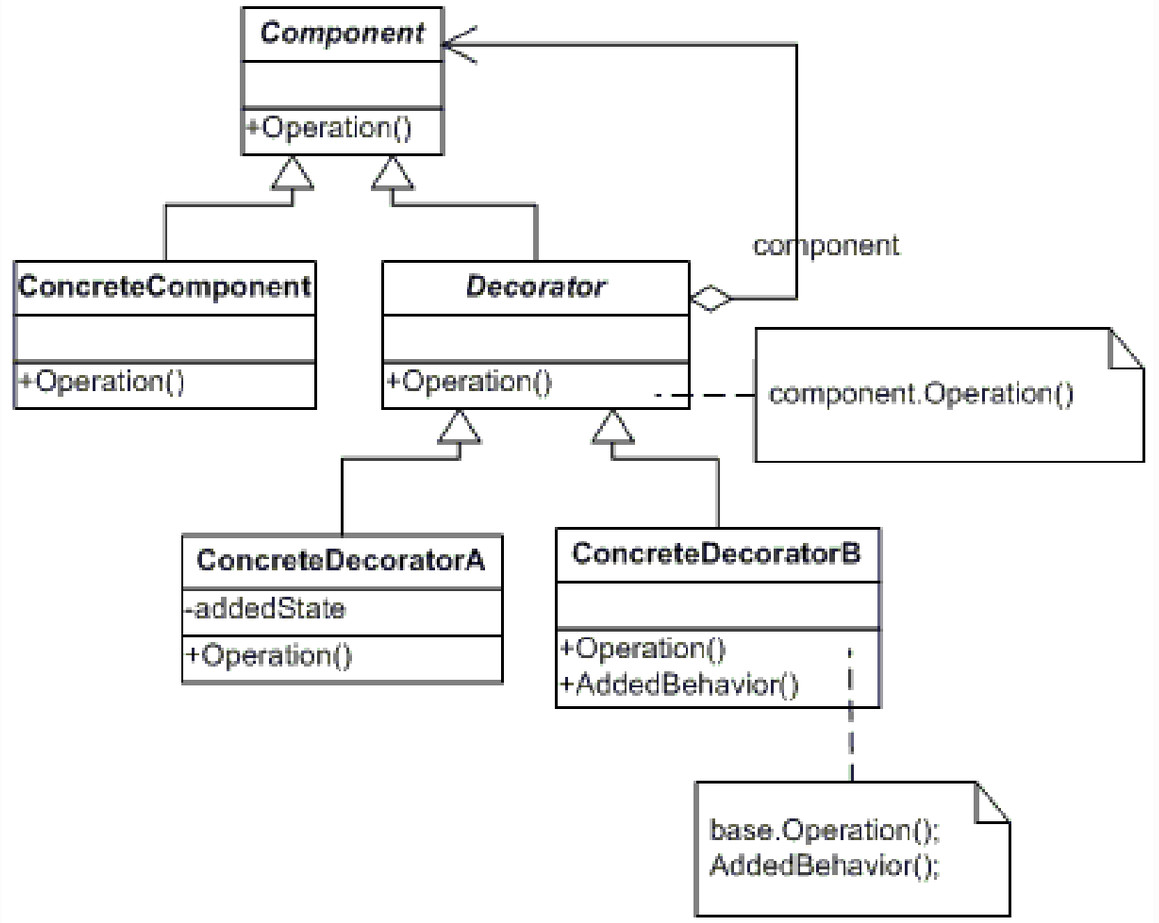


Structural Patterns

- Facade
- Implementation explanation :
    - Facade : knows which subsystem classes are responsible for a request.
      delegates client requests to appropriate subsystem objects.
- Subsystem classes : implement subsystem functionality.
    handle work assigned by the Facade object.
    have no knowledge of the facade and keep no reference to it.

Structural Patterns

- Decorator
- Use : Medium
- Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

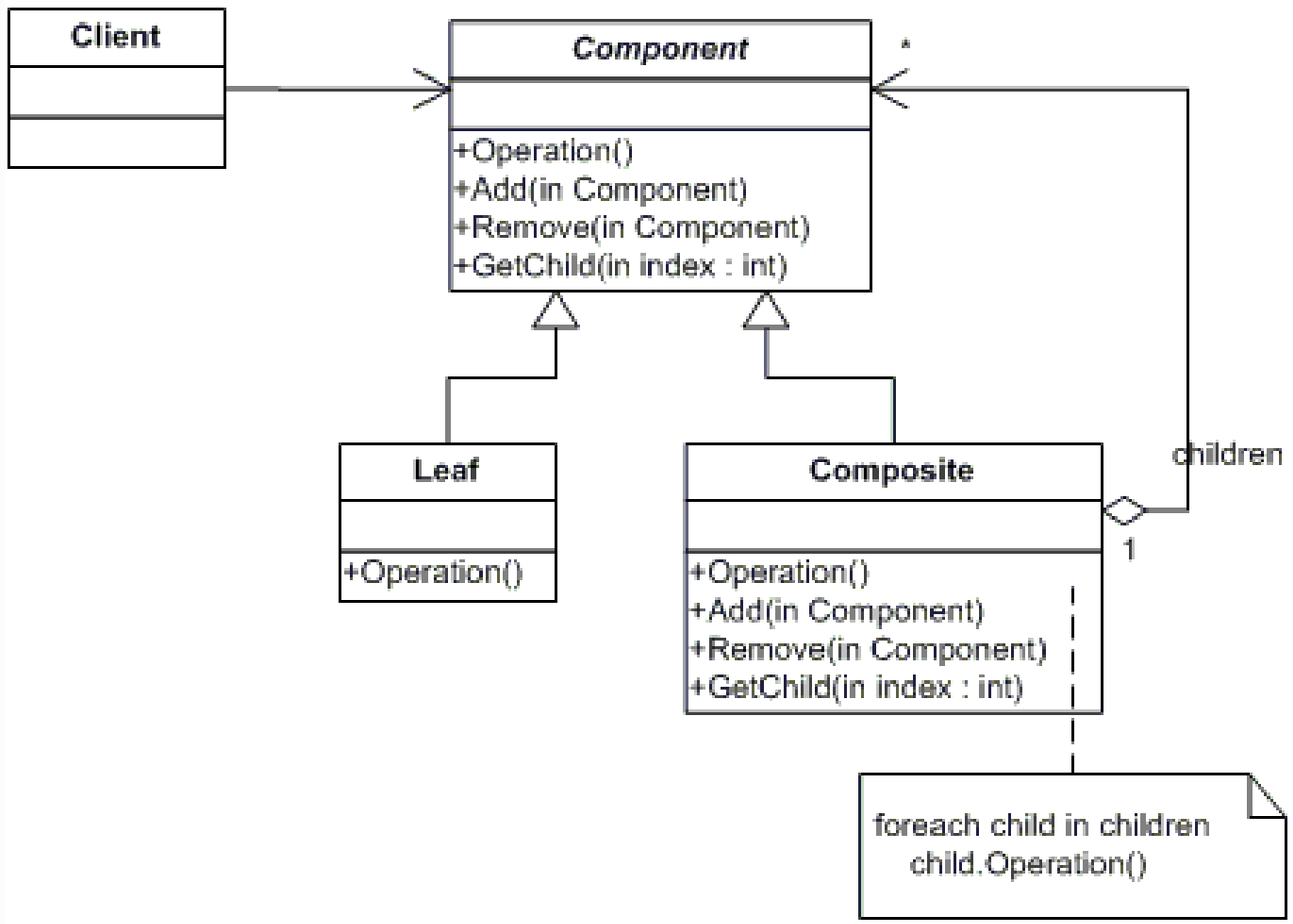


Structural Patterns

- Decorator
- Implementation explanation :
    - Component : defines the interface for objects that can have responsibilities added to them dynamically.
- ConcreteComponent : defines an object to which additional responsibilities can be attached.
- Decorator : maintains a reference to a Component object and defines an interface that conforms to Component's interface.
- ConcreteDecorator : adds responsibilities to the component.

Structural Patterns

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



Structural Patterns

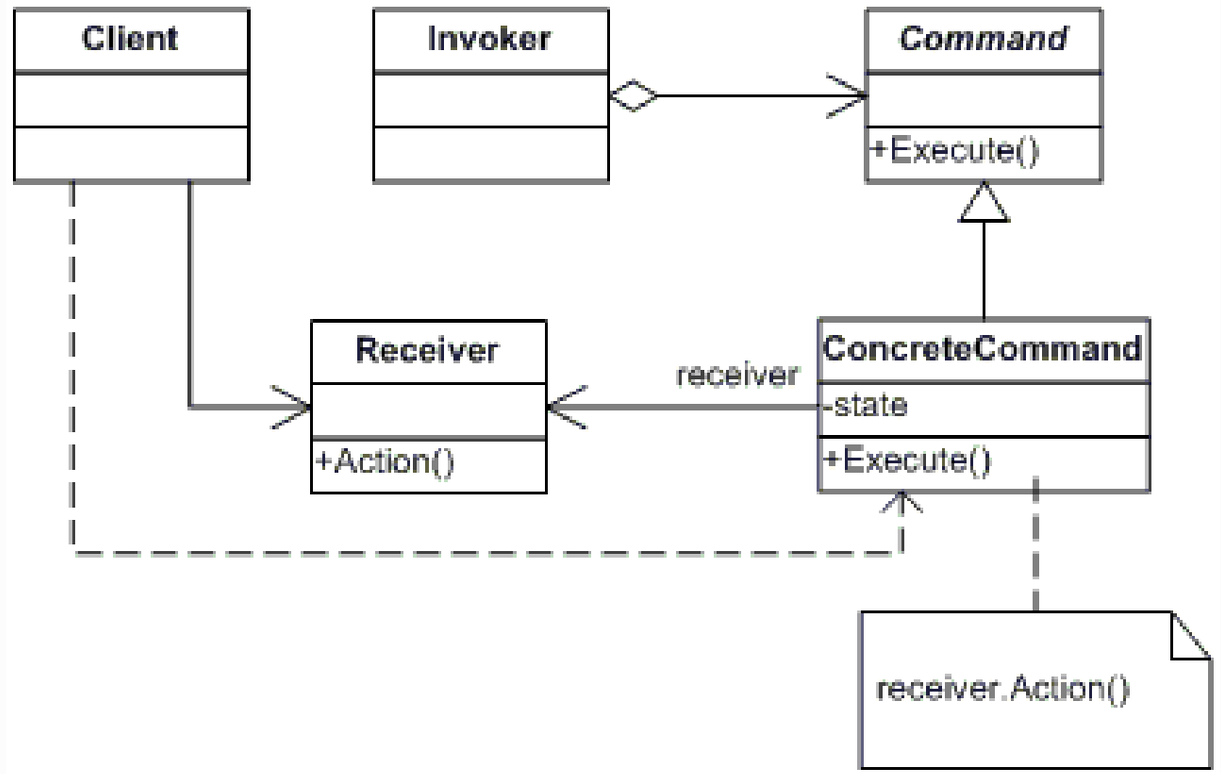
- Composite
- Implementation explanation :
    - Component : declares the interface for objects in the composition.
    implements default behavior for the interface common to all classes, as appropriate.
    declares an interface for accessing and managing its child components.
    (optional) defines an interface for accessing a component's parent in the recursive structure, and implements it if that's appropriate.

Structural Patterns

- Composite
- Implementation explanation :
    - Leaf : represents leaf objects in the composition. A leaf has no children.
      defines behavior for primitive objects in the composition.
- Composite : defines behavior for components having children.
      stores child components.
      implements child-related operations in the Component interface.
- Client : manipulates objects in the composition throught interface.

Behavioral Patterns

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



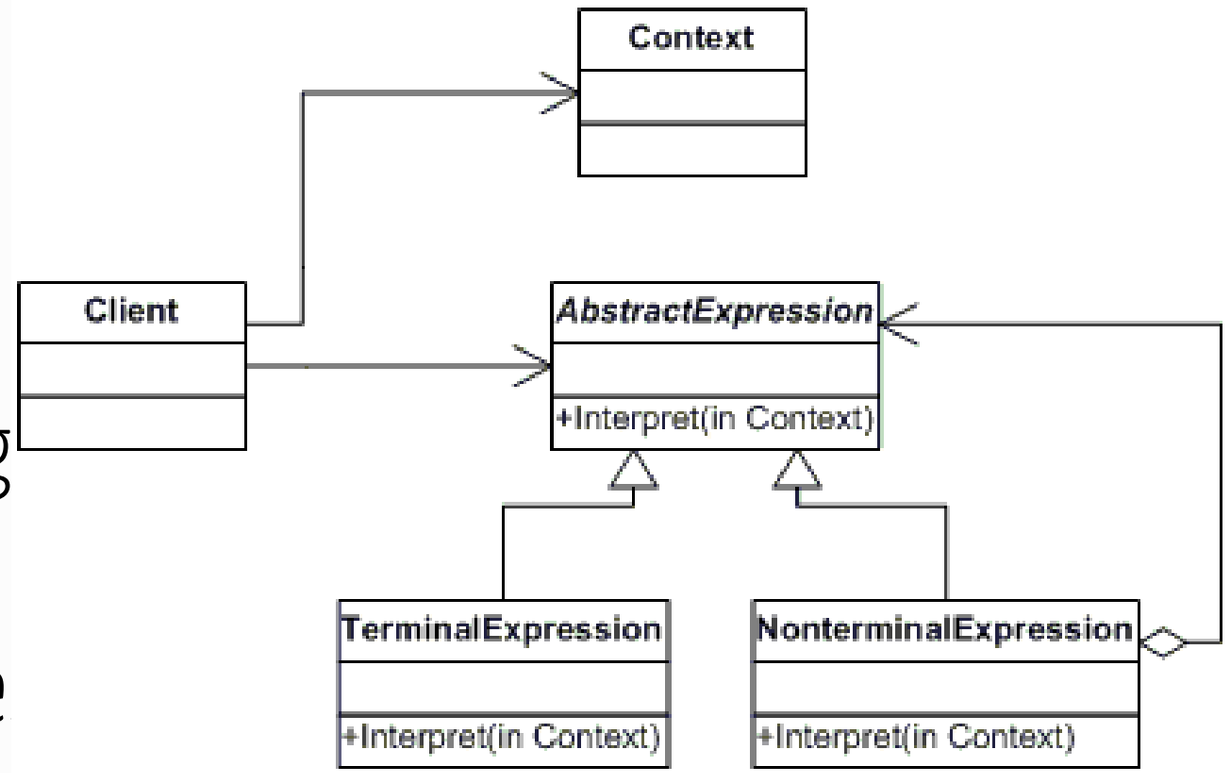
Behavioral Patterns

- Command
- Implementation explanation :
    - Command : declares an interface for executing an operation
    - ConcreteCommand : defines a binding between a Receiver and action
        implements Execute by invoking the corresponding operation(s)
        on Receiver.
    - Client : creates a ConcreteCommand object and sets its receiver
    - Invoker : asks the command to carry out the request

Receiver : knows how to perform the operations associated withthe request.

Behavioral Patterns

- Interpreter
- Use : Low
- Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentence in the language.



Behavioral Patterns

- Interpreter
- Implementation explanation :
    - AbstractExpression : declares an interface for executing an operation
    - TerminalExpression : implements an Interpret operation associated with terminal symbols in the grammar.
      an instance is required for every terminal symbol in the sentence.

Behavioral Patterns

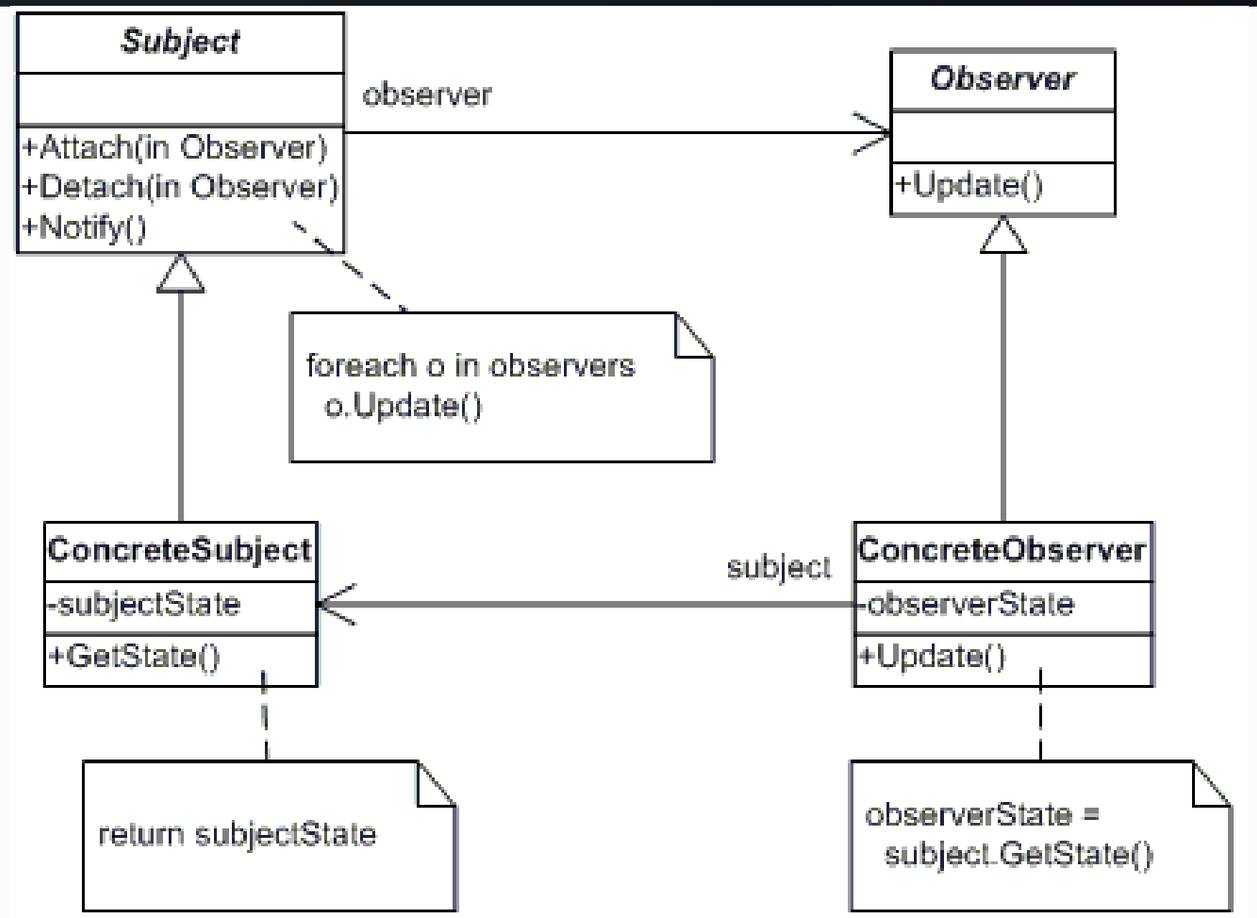
- Interpreter
- Implementation explanation :
    - NonterminalExpression : one such class is required for every rule R ::= R1R2...Rn in the grammar
      maintains instance variables of type AbstractExpression for each of the symbols R1 through Rn.
      implements an Interpret operation for nonterminal symbols in the grammar. Interpret typically calls itself recursively on the variables representing R1 through Rn.

Behavioral Patterns

- Interpreter
- Implementation explanation :
    - Context : contains information that is global to the interpreter
    - Client : builds (or is given) an abstract syntax tree representing a particular sentence in the language that the grammar defines. The abstract syntax tree is assembled from instances of the NonterminalExpression and TerminalExpression classes invokes the Interpret operation

Behavioral Patterns

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



Behavioral Patterns

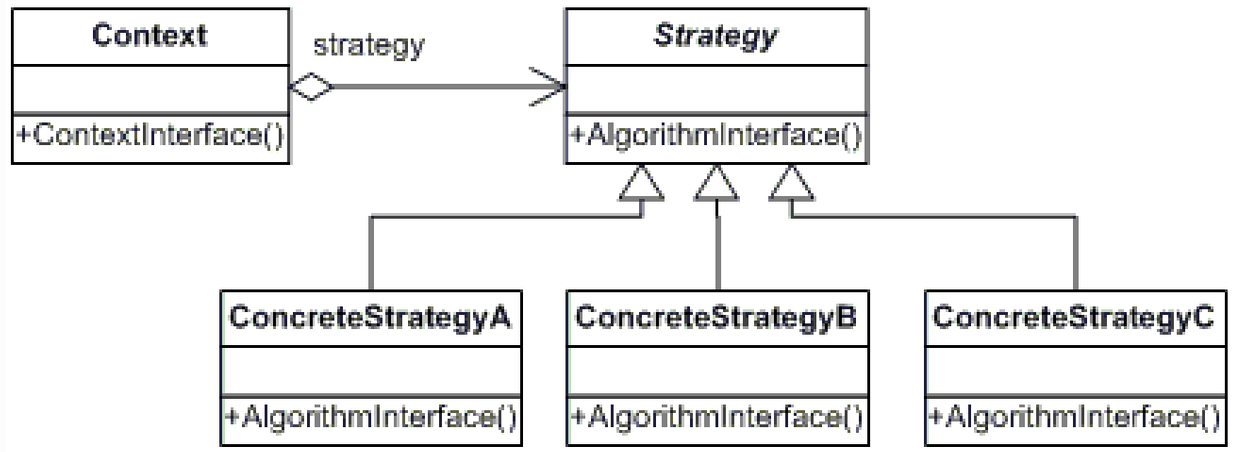
- Observer
- Implementation explanation :
    - Subject : knows its observers. Any number of Observer objects may observe a subject
      provides an interface for attaching and detaching Observer objects.
- ConcreteSubject : stores state of interest to ConcreteObserver
  sends a notification to its observers when its state changes

Behavioral Patterns

- Observer
- Implementation explanation :
    - Observer : defines an updating interface for objects that should be notified of changes in a subject.
- ConcreteObserver : maintains a reference to a ConcreteSubject object stores state that should stay consistent with the subject's implements the Observer updating interface to keep its state consistent with the subject's

Behavioral Patterns

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

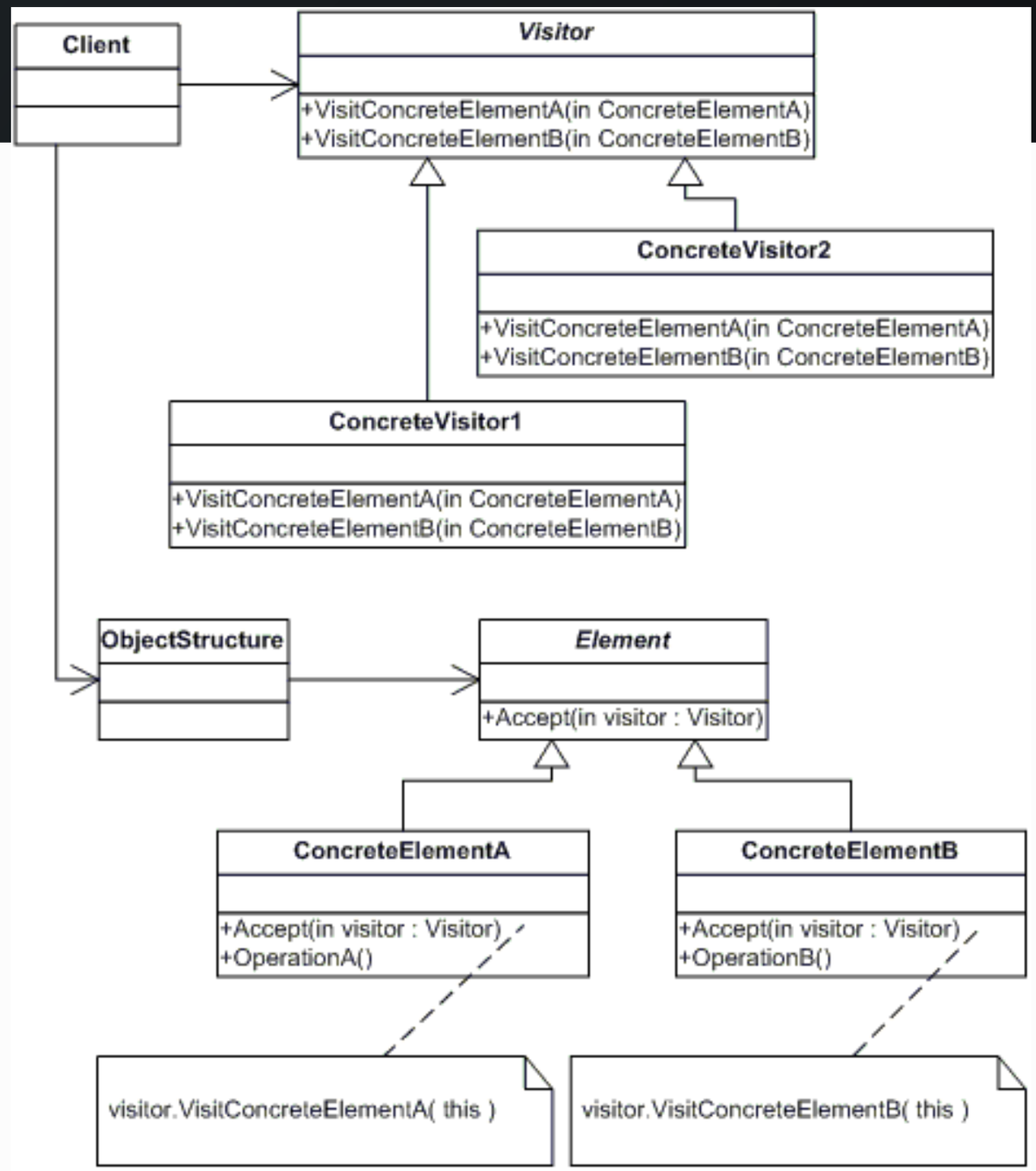


Behavioral Patterns

- Strategy
- Implementation explanation :
    - Strategy : declares an interface common to all supported algorithms.
      Context uses this interface to call the algorithm
- ConcreteStrategy : implements the algorithm using the Strategy
  interface
- Context : is configured with a ConcreteStrategy object
  maintains a reference to a Strategy object
  may define an interface that lets Strategy access its data.

Behavioral Patterns

- Visitor
- Use : Low
- 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.



Behavioral Patterns

- Visitor
- Implementation explanation :
    - Visitor : declares a Visit operation for each class of ConcreteElement in the object structure. The operation's name and signature identifies the class that sends the Visit request to the visitor. That lets the visitor determine the concrete class of the element being visited. Then the visitor can access the elements directly through its particular interface

Behavioral Patterns

- Visitor
- Implementation explanation :
    - ConcreteVisitor : implements each operation declared by Visitor. Each operation implements a fragment of the algorithm defined for the corresponding class or object in the structure. ConcreteVisitor provides the context for the algorithm and stores its local state. This state often accumulates results during the traversal of the structure.

Behavioral Patterns

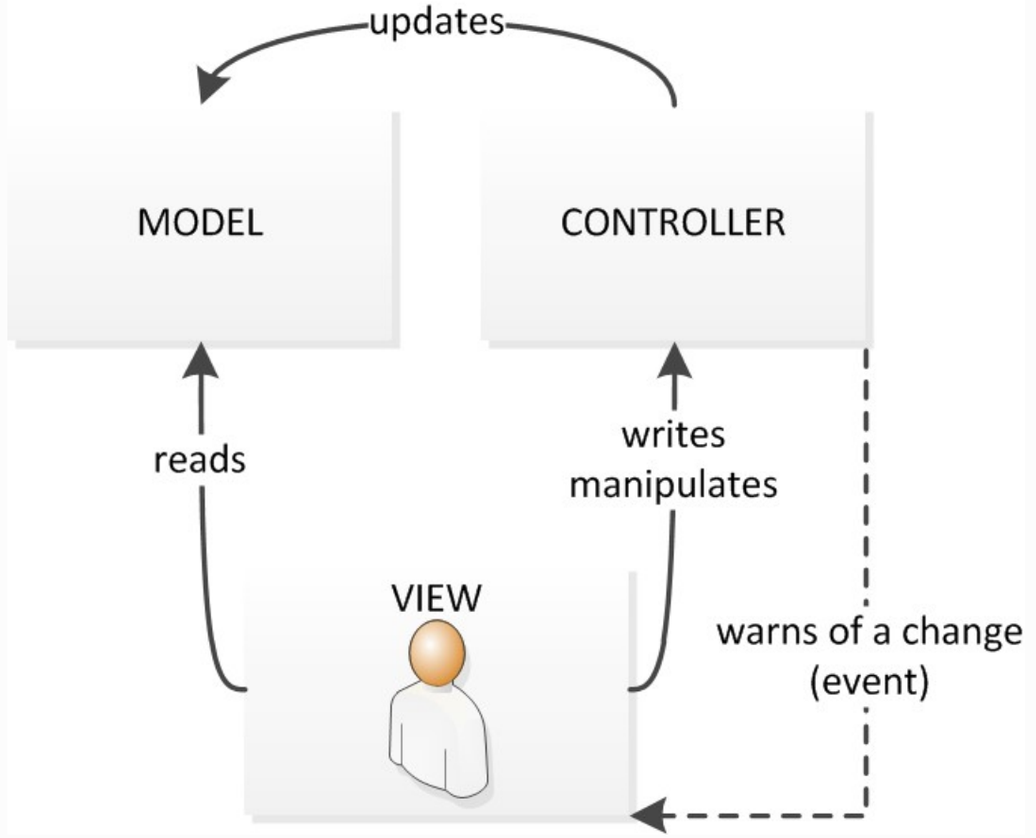
- Visitor
- Implementation explanation :
    - Element : defines an Accept operation that takes a visitor as an argument.
    - ConcreteElement : implements an Accept operation that takes a visitor as an argument

Behavioral Patterns

- Visitor
- Implementation explanation :
    - ObjectStructure : can enumerate its elements
      may provide a high-level interface to allow the visitor to visit its elements
      may either be a Composite (pattern) or a collection such as a list or a set

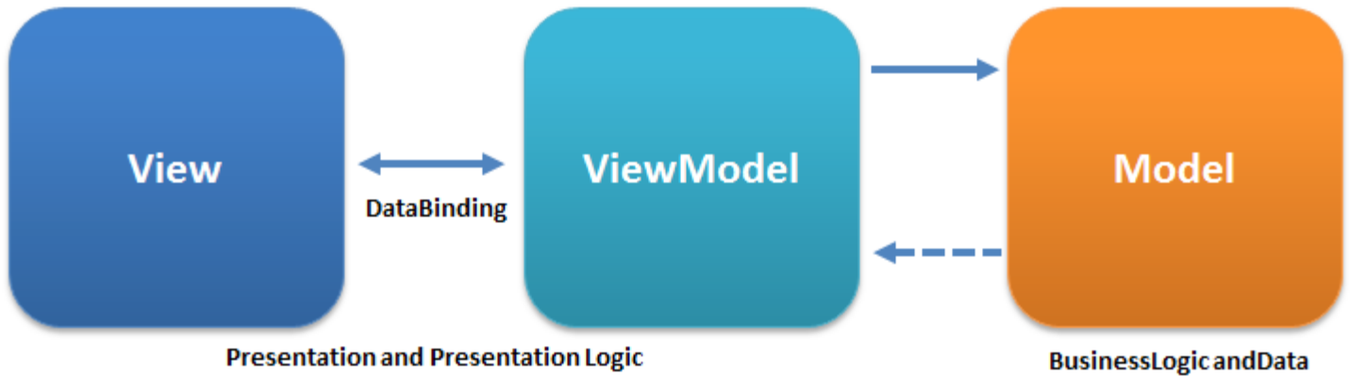
MVC

- MVC : Model View Controler
- Model : data model
- View : presentation, IHM
- Controler : logic, event management
  synchronisation
- Dedicated to graphical apps
- Base architecture for apps conception but
  do not solve all issue it is just a start point



MVVM

- MVVM : Model View View-Model
- Windows Presentation Foundation technologie



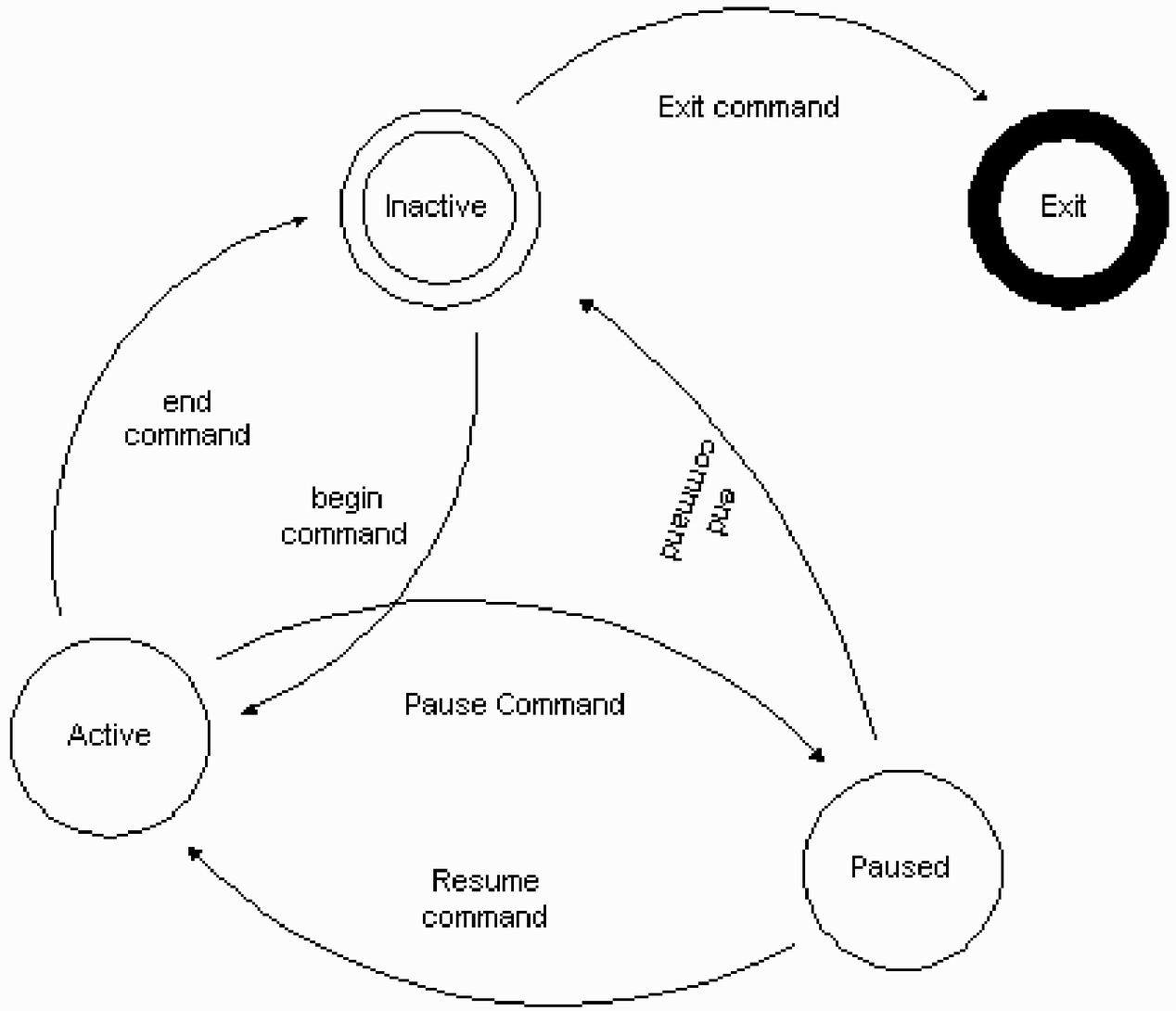
- Silverlight
- Model : data model, logic
- View : presentation, IHM, native data binding with View-Model
- View-Model : view logic, event management, native data binding with View
- Base architecture for Microsoft app conception (WPF, Universal App ...)

Command Worker C#

- Use in multi-threading context
- Allow to couple thread management with command execution
- Utility class
- Allow to exec action in wished order from an action poll
- Can set one element to first, set differents elements first and do not
  let them time to exec will result to dismiss them
- Can be started and stoped keeping saved elements

State Machine C#

- Use to make sure that we process stuff in correct order
- Depend of algorithme logic transition
- Can make stuff for current state
- Can do stuff on bad transition
- Ensure that app do not run in a none wished state



Questions ?

Change log

- Antoine Cronier 2016 : Initial document