AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

Architecture & Design of Embedded Real-Time Systems (TI-AREM)

GoF Strategy Pattern
(a Behavioral Pattern)

UNIFIED
MODELING
LANGUAGE
UML TM

Version: 25-2-2015

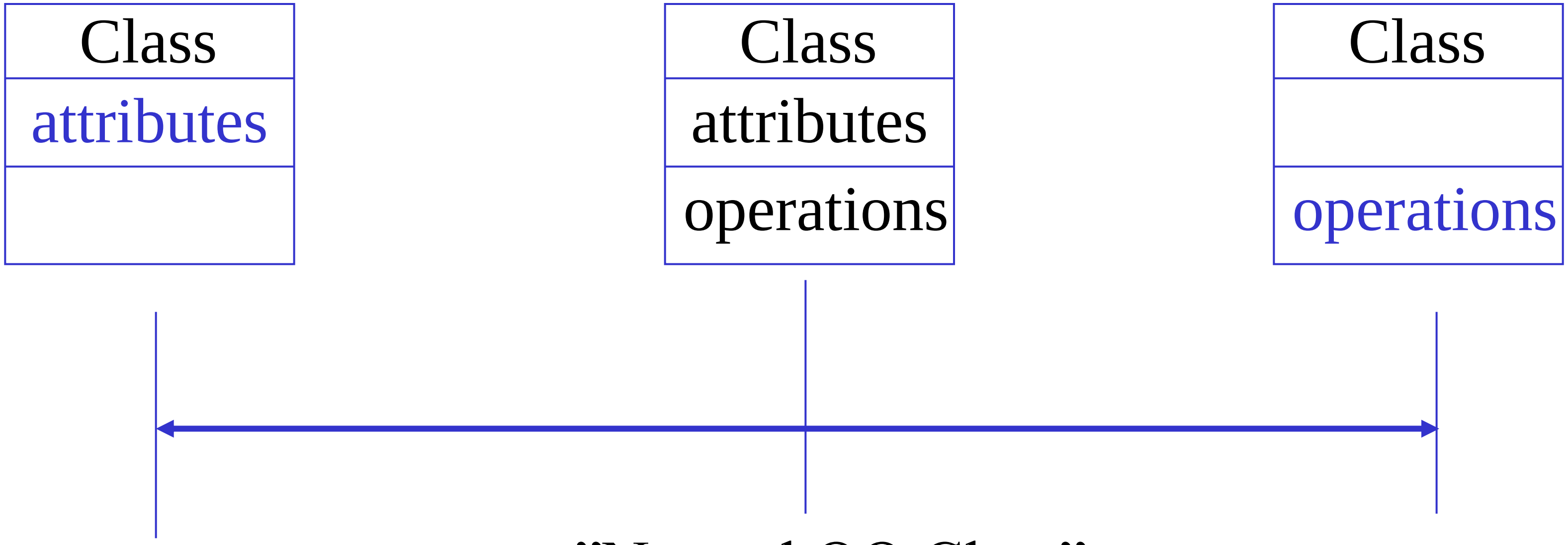
Strategy Pattern – Object Behavioral

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.

Classes and Objects



Class
attributes

Class
attributes
operations

Class
operations

”Normal OO Class”

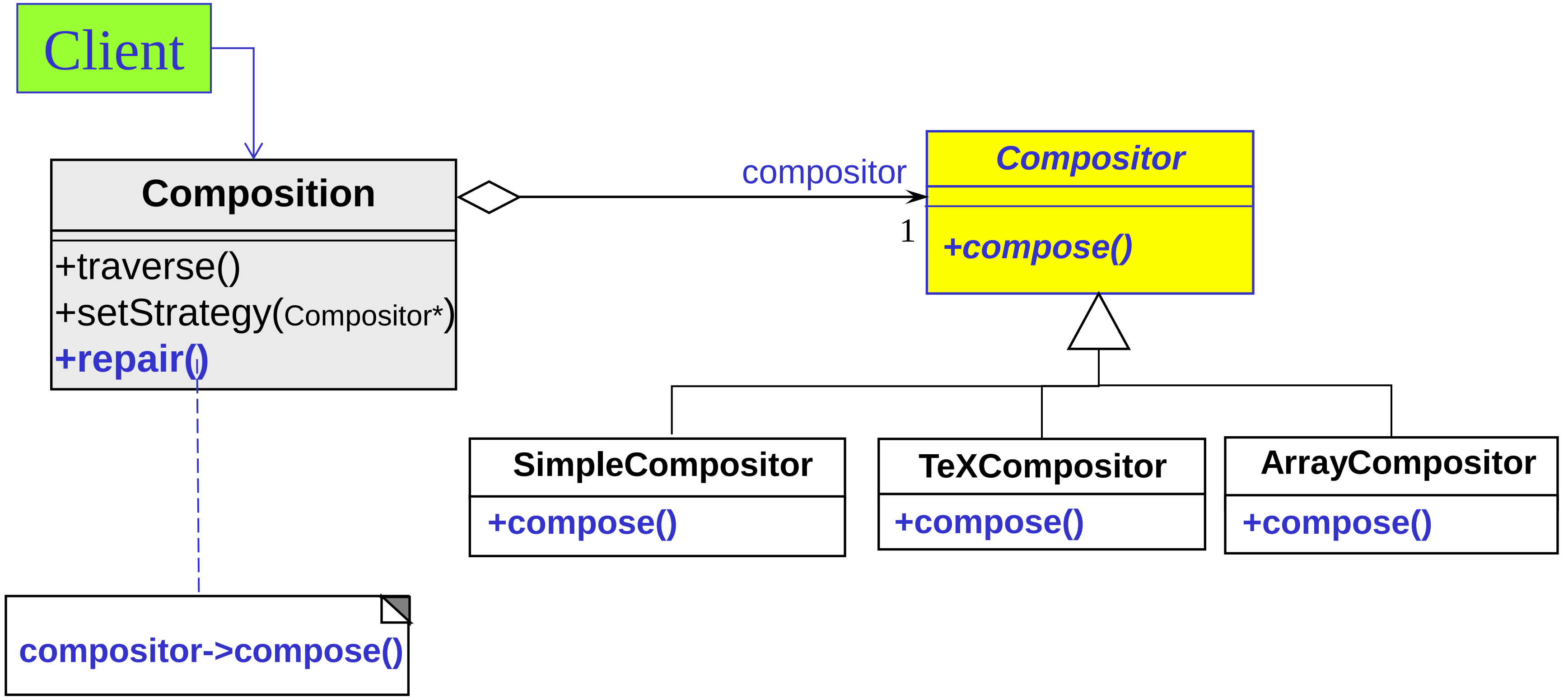
DATA

DATA +
ALGORITHMS

ALGORITHM

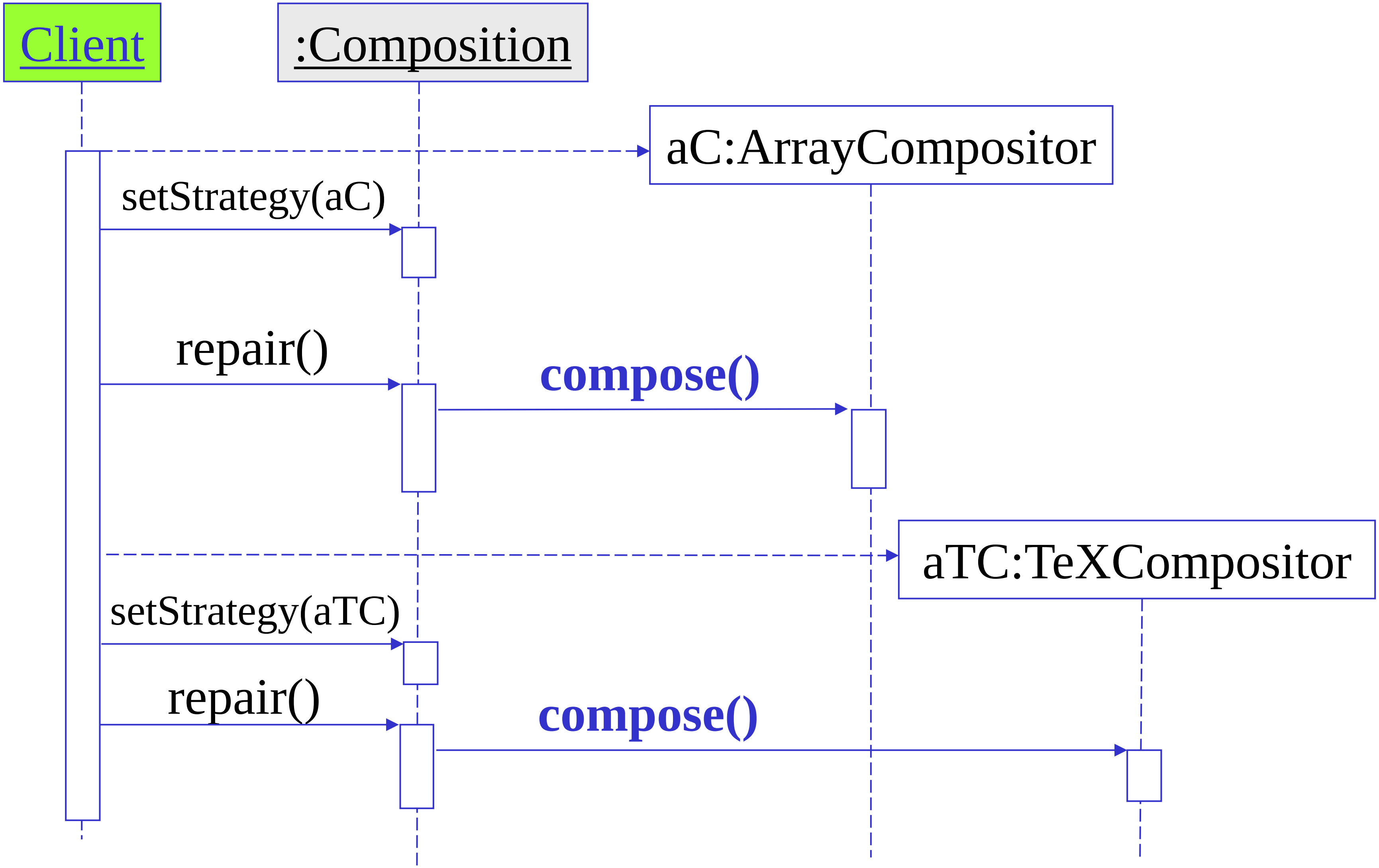
Strategy - Motivation Example

Many algorithms exist for breaking a stream of text into lines

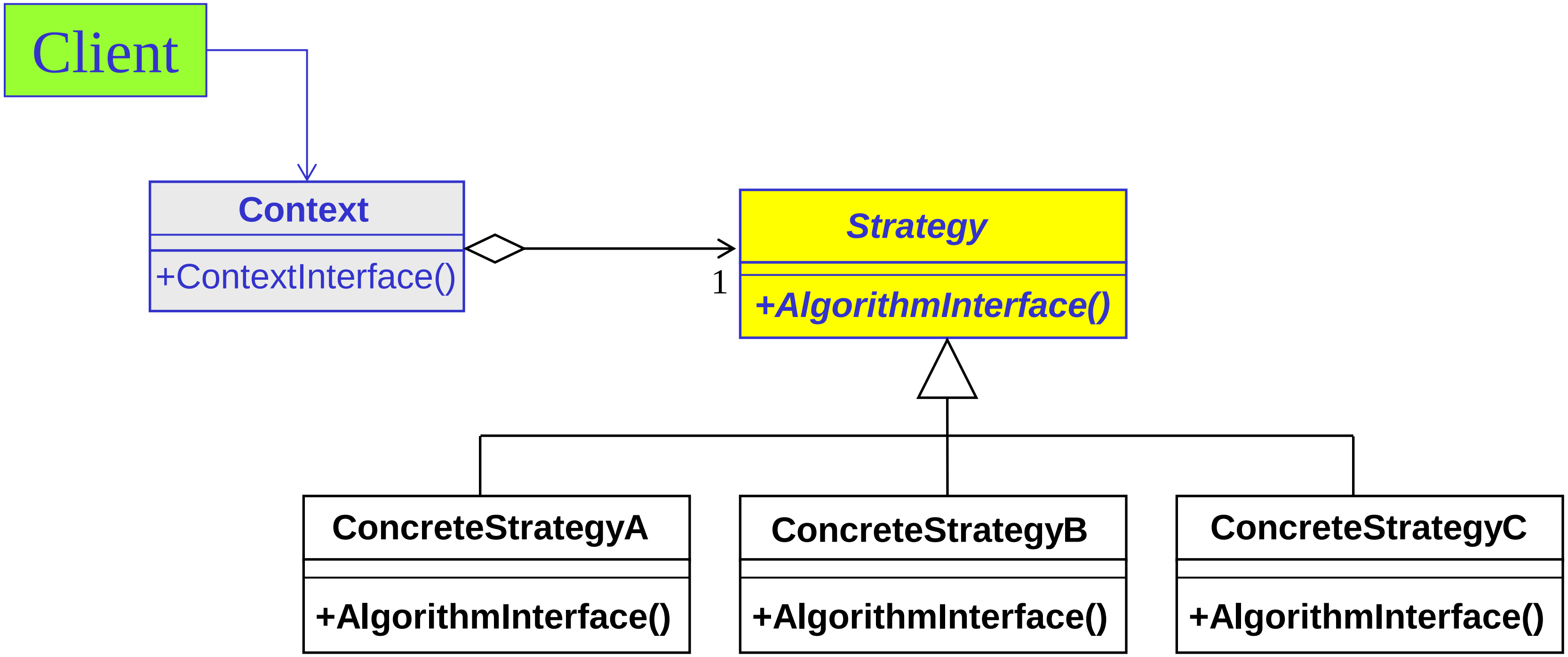


AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

Strategy – Sequence Diagram

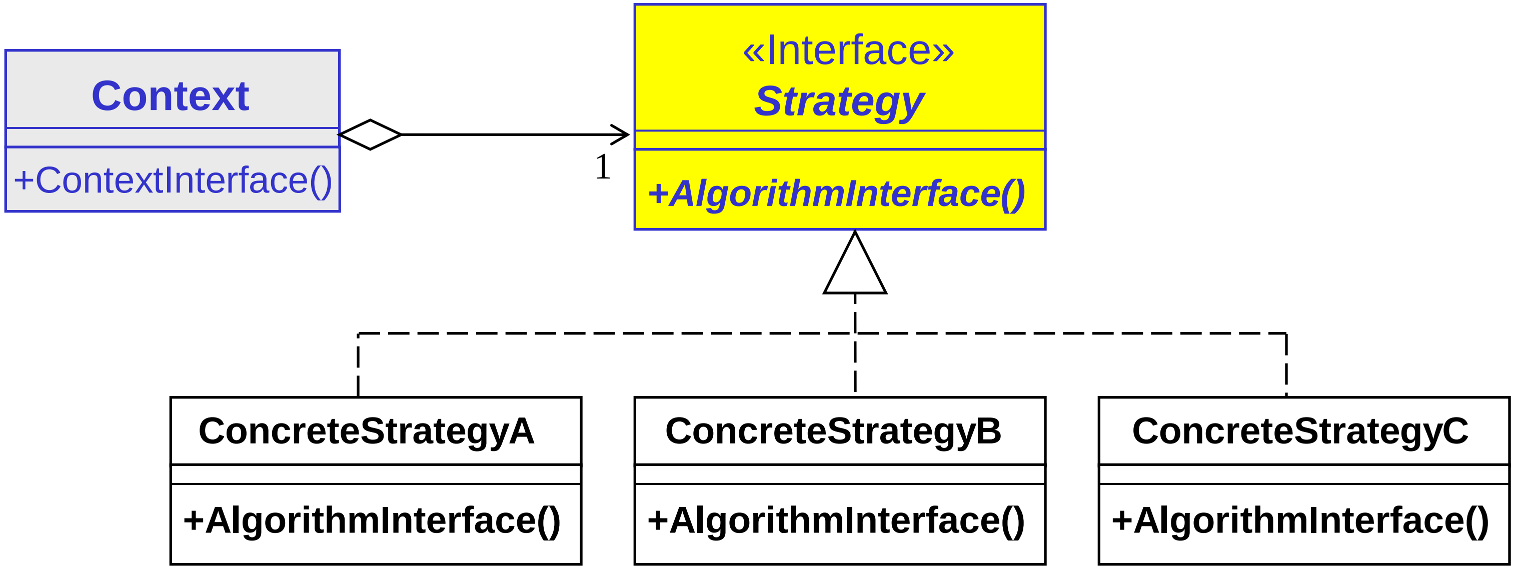


Strategy Pattern - GoF Structure (1)



Strategy Pattern - GoF Structure (2)

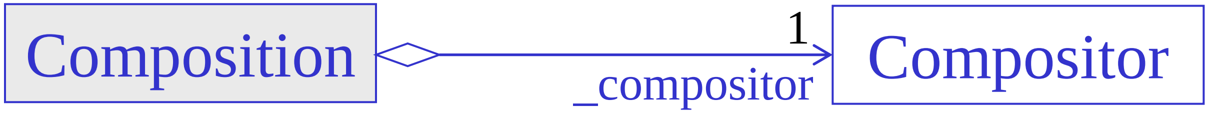
Version with interface



Strategy – Implementation Details

- Algorithm information can be handed over by algorithm parameters (**PUSH**)
- Information can be pulled from the calling context objects (**PULL**)
  - requires a bidirectional association or a context object pointer as an algorithm parameter
- The strategy object can be initialized by construction of the context object
- Or dynamic by adding a "**setStrategy()**" operation in the context class

Strategy - C++ Example (1)



```
class Composition
{
  public:
    Composition(Compositor*);
    void repair();
  private:
    Compositor * _compositor;
    Component* _components;
    int _componentCount;
    int _lineWidth, _lineCount;
    int* _lineBreaks;
};
```

Push

```
class Compositor
{
public:
    virtual int compose(
          coord *natural,
          coord *stretch,
          coord *shrink,
          int componentCount,
          int lineWidth, int breaks ) = 0;
protected:
    Compositor();
};
```

Strategy - C++ Example (2)

```
void Composition::repair()
{
    coord* natural;
    coord* streachability;
    coord* shrinkability;
    int componentCount;
    int* breaks;
    // ..
    //  Delegation call of Compose algorithm
    int breakCount = _compositor->compose(natural, streachability,
                                   shrinkability, componentCount,
                           _lineWidth, breaks);
}
```

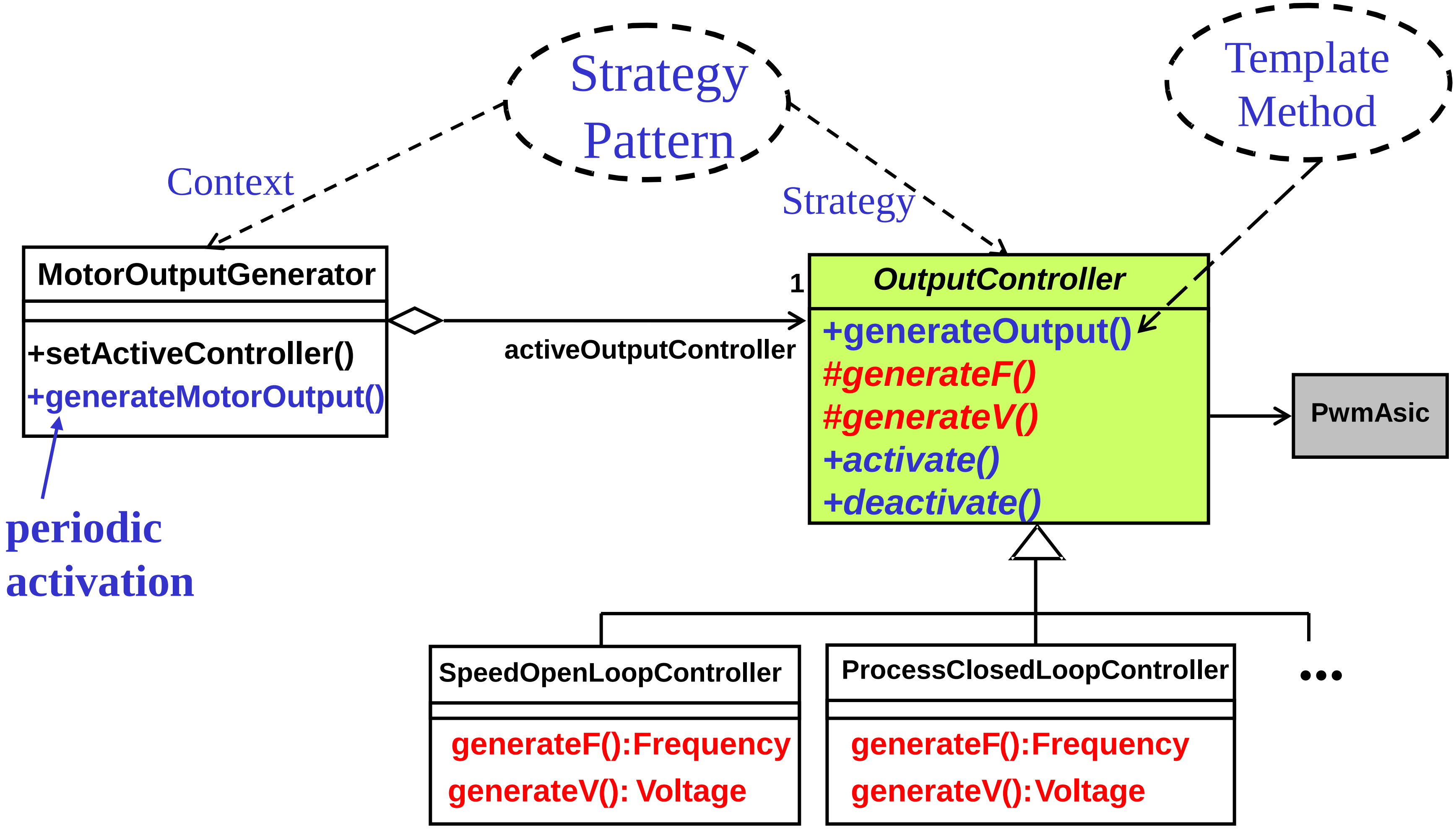
Strategy – C++ Example (3)

```
main()
{
   Composition* quick=
               new Composition(new SimpleCompositor);
   Composition* slick=
               new Composition(new TexCompositor);
   Composition myIconic(new ArrayCompositor(100));
   //
   quick->repair();
   slick->repair();
   myIconic.repair();
}
```

Strategy - Consequences

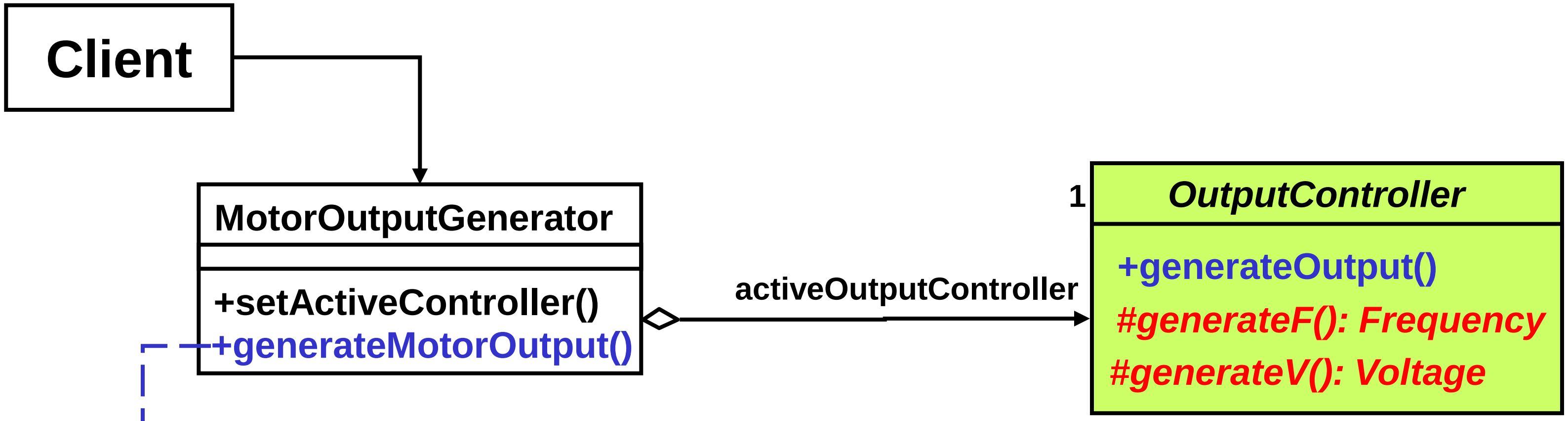
- Families of related algorithms
- An alternative to subclassing (of Context)
- Strategies eliminate conditional statements
- A choice of implementations
- Clients must be aware of different strategies
- Communication overhead between strategy and context
- Increased number of objects

Example: Strategy + Template Pattern



Ref. "Two-part model .."

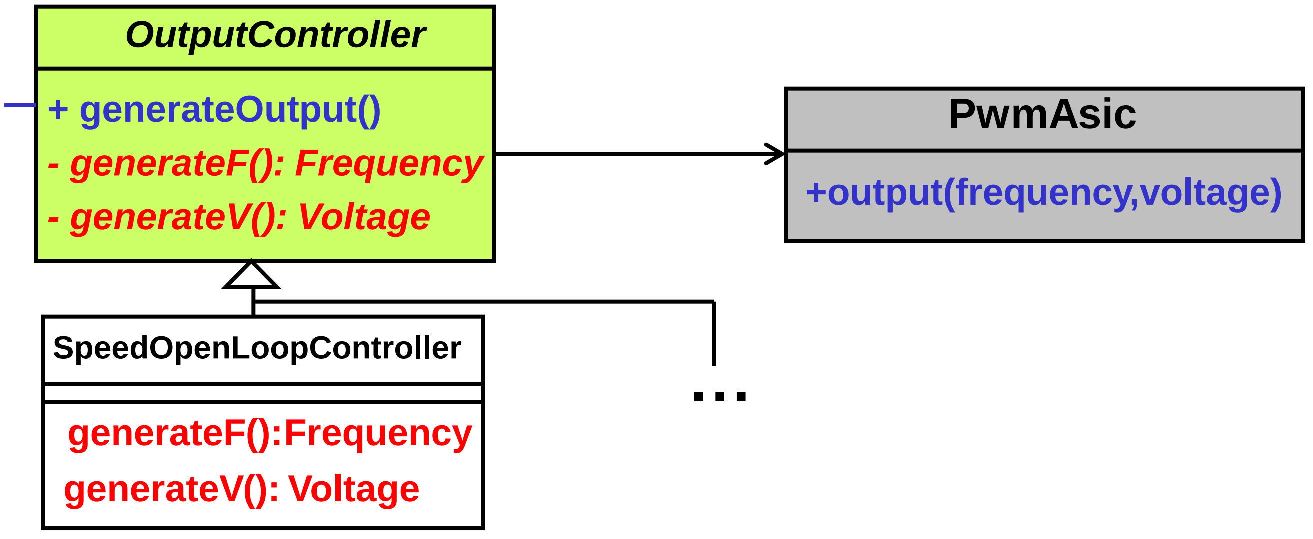
C++ Code Example for ‘generateMotorOutput()’



OutputController *activeOutputController;

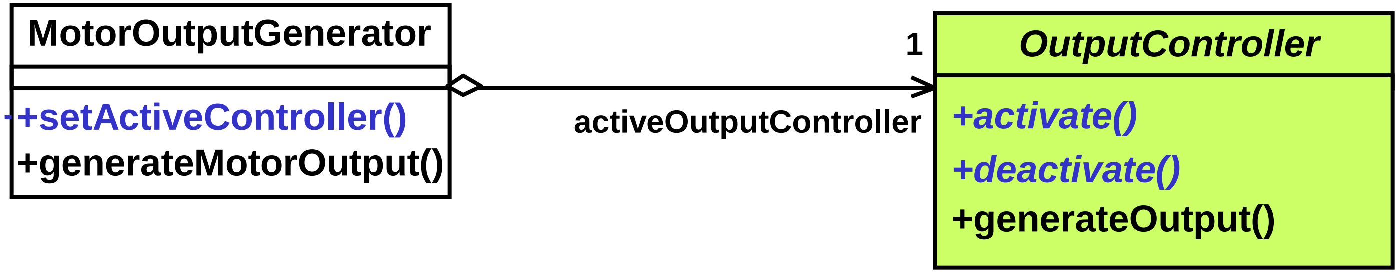
```
MotorOutputGenerator::generateMotorOutput()
{
    activeOutputController->generateOutput();  // Strategy
}
```

C++ Code Example for ‘generateOutput()’



```
OutputController::generateOutput()    // Template Method (GoF)
{
    frequency= generateF();           // pure virtual function
    voltage= generateV();             // pure virtual function
    thePwmAsic->output(frequency,voltage);
}
```

C++ Code Example for ‘setActiveController()’



```
MotorOutputGenerator::setActiveController
                                        (OutputController* newController)
{
      controllerInfo= activeOutputController->deactivate();
      activeOutputController= newController;
      activeOutputController->activate(controllerInfo);
}
```

Notice: a method for transfer of controller state information

Class Exercise

Using the example from the previous slide:

Sketch a **Main Program** with the following functionality:

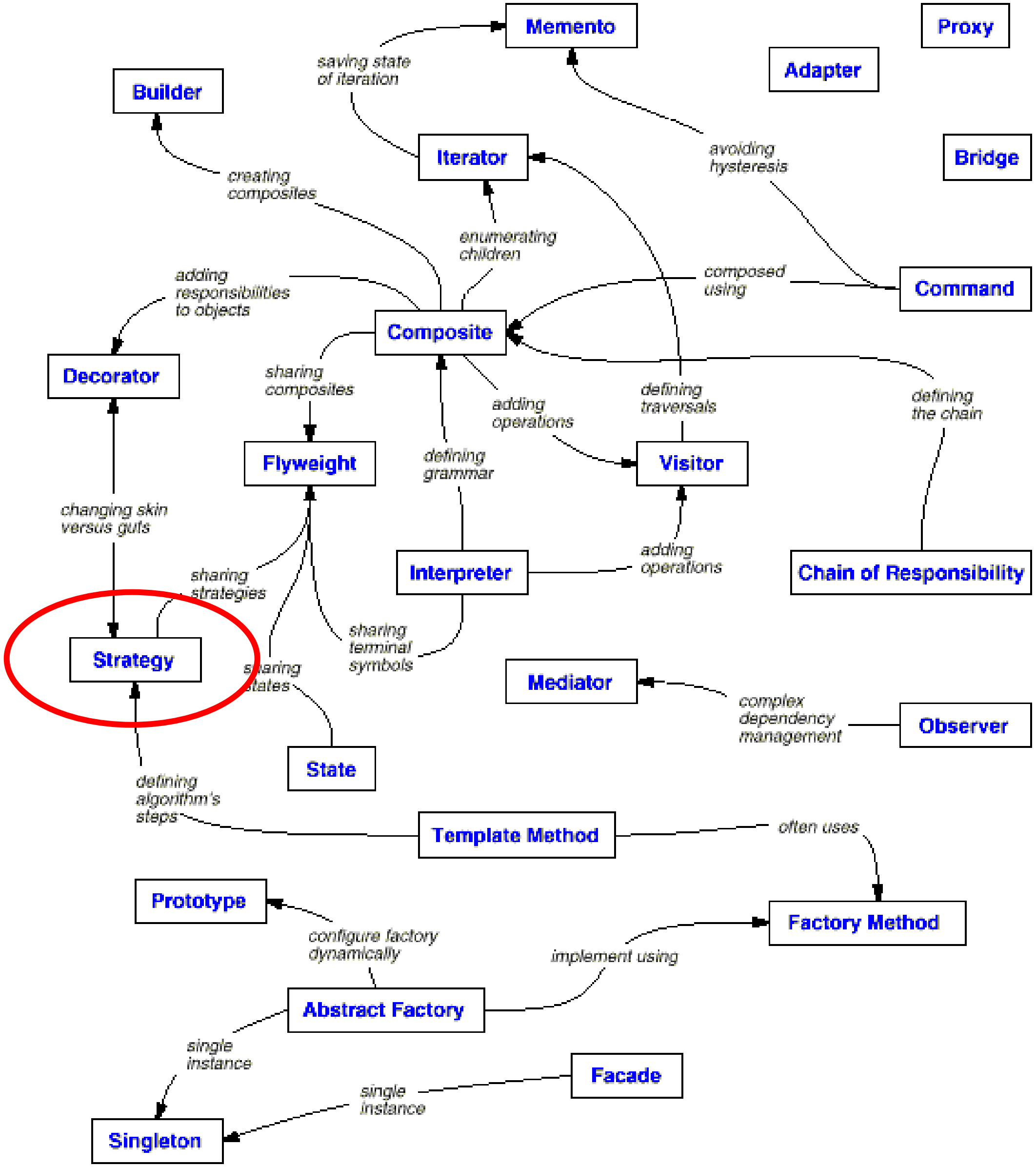
1. Instantiate a MotorOutputGenerator object using a SpeedOpenLoopController object.
2. Continuous do the following:

   2.1 write the code for activating the ”generateOutput” operation in the actual controller object every 1. ms

   2.2 test for conditionX and if true change the controller object to the ClosedLoopController object.

Summary

- Strategy
  - extremely useful



Memento
Proxy
Adapter
Builder
saving state of iteration
Iterator
avoiding hysteresis
Bridge
creating composites
enumerating children
composed using
Command
adding responsibilities to objects
Composite
Decorator
sharing composites
adding operations
defining traversals
defining the chain
Flyweight
defining grammar
Visitor
changing skin versus guts
sharing strategies
Interpreter
adding operations
Chain of Responsibility
sharing terminal symbols
Strategy
sharing states
Mediator
complex dependency management
Observer
State
defining algorithm's steps
Template Method
often uses
Prototype
Factory Method
configure factory dynamically
implement using
Abstract Factory
single instance
Facade
single instance
Singleton