CSCI 253

Object Oriented Design:
Strategy Pattern
George Blankenship

Overview

Creational Patterns
- Singleton
- Abstract factory
- Factory Method
- Prototype
- Builder

Structural Patterns
- Composite
- Façade
- Proxy
- Flyweight
- Adapter
- Bridge
- Decorator

Behavioral Patterns
- Chain of Respons.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

The Strategy Pattern: The Problem

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

car
registrnr:

person
age:

- Different line-breaking algorithms in an editor
- Different kinds of memory allocation algorithms for collection classes
- Different algorithms for checking valid input in different types of dialog boxes

Strategy Pattern George Blankenship 4

HL7 Message Strategy Example

- Build HL7 messages
  - Each message has a unique form
  - Encapsulate the message construction in message classes
- Root class “GenerateMessage”
- Derived classes “GenerateX…XMessage”
  - generateMessage(…) is the algorithm for individual message
  - Normally object is accessed as a “GenerateMessage” object, not as “GenerateX…XMessage” object
  - Objective of the strategy is to generate the unique string that is a message

Strategy Pattern George Blankenship 5

FSM Strategy Example

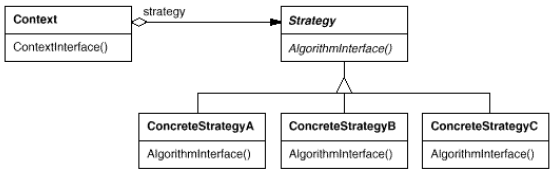
- Kernel of a application is an FSM
  - Application is able to perform multiple simultaneous FSMs
  - Application may have one of many FSMs active
- Root class “Statemachine”
- Derived classes “FSMclient”, “FSMserver”, “FSMvaDataFlow”, “FSMvendorDataFlow”, and “FSMwaitVendorConnection” define possible execution algorithm
  - Execution algorithm is embedded in xqt() method of state machine

Strategy Pattern George Blankenship 6

The Strategy Pattern Applicability

- Many different classes differ only in their behavior; using Strategy provides a way to configure a class with one of many behaviors
- Many different variants of an algorithm are needed; using Strategy the variants can be implemented as a class hierarchy of algorithms
- The algorithms use data that clients should not know about; using Strategy avoids exposing complex, algorithm specific data

The Strategy Pattern: Structure



The Strategy Pattern: Participants

- Strategy
  - Declares an interface common to all supported algorithms
  - Context uses this interface to call the algorithm defined by a ConcreteStrategy
- ConcreteStrategy subclasses
  - Each subclass implements the algorithm using the Strategy interface
- Context
  - Is configured with a ConcreteStrategy object
  - May define an interface that lets Strategy access its data

The Strategy Pattern: Collaborations

- A Context forwards requests from its clients to its Strategy
- Strategy and Context interact to implement the chosen algorithm.
  - A Context may pass all the data required by the algorithm to the Strategy when calling the algorithm.
  - Context may pass itself as an argument to Strategy operations so that these can call back on the Context as required.
- Clients usually create and pass a ConcreteStrategy object to the Context; thereafter clients interact with the Context exclusively

Strategy Pattern George Blankenship 10

The Strategy Pattern: Consequences

- Hierarchies of Strategy classes define a family of algorithms or behaviors to reuse. Inheritance can factor out common functionality of the algorithms
- Is alternative to subclassing, I.e. using a hierarchy of Context classes to implement the variations in behavior. The behavior is not hard-wired in the context so the algorithm can vary independently of the Context
- Is an alternative to using conditional statements for selecting desired behavior
- Can offers a choice of implementations for the same behavior (e.g. space and time tradeoffs)

Strategy Pattern George Blankenship 11

The Strategy Pattern: Concerns

- Clients must be aware of different Strategies. The clients must understand the difference amongst what is offered to be able to select the appropriate one. Therefor the pattern should only be used when the variation in algorithm (implementation) is relevant for the client.
- There is some communication overhead between Strategy and Context. Strategy defines a (general) interface. Many of the simpler algorithms will not need all the information that is passed.
- The number of objects in the application increases. Sometimes this overhead can be reduced when ConcreteStrategies can be implemented by stateless objects that can be shared (cfr. The flyweight pattern)

Strategy Pattern George Blankenship 12

The Strategy Pattern: Implementation

- Defining the strategy and contextinterfaces
  - Let context pass data to strategy operations
    - Keeps context and startegy decoupled
    - Context might pass data that strategy does not need
  - Let context pass itself to strategy operations and strategy can then request data from context
    - Couples context and strategy more closely
    - Context must have more elaborate interface
- Strategies as template parameters
  - In C++ templates can be used to configure a context with a strategy
  - Only applicable when strategy can be selected at compile time and does not need to change at run time

Strategy Pattern George Blankenship 13