Fontys ICT

STRATEGY PATTERN

Design patterns

Jan-Niklas Schneider, Georgiana Manolache
9-6-2016

Table of Contents

1 Introduction

The goal of this document is to give an overview over the design pattern strategy pattern by giving an example implementation which simulates an operating system's disk scheduling procedure. Furthermore, reusability, extensibility, and maintainability of this pattern are elaborated. Also, the implementation, its unit test and graphical user interface (GUI) are reviewed.

2 Strategy Pattern

The strategy pattern is a software design pattern which allows changing the behavior or algorithm of an application in runtime.

The strong points of the pattern are the ability to capture abstraction in an interface and encapsulate its implementation in derived classes. Adding a context, a layer between the strategy and main interface, allows interchangeability between strategies in runtime.

3 Implementation

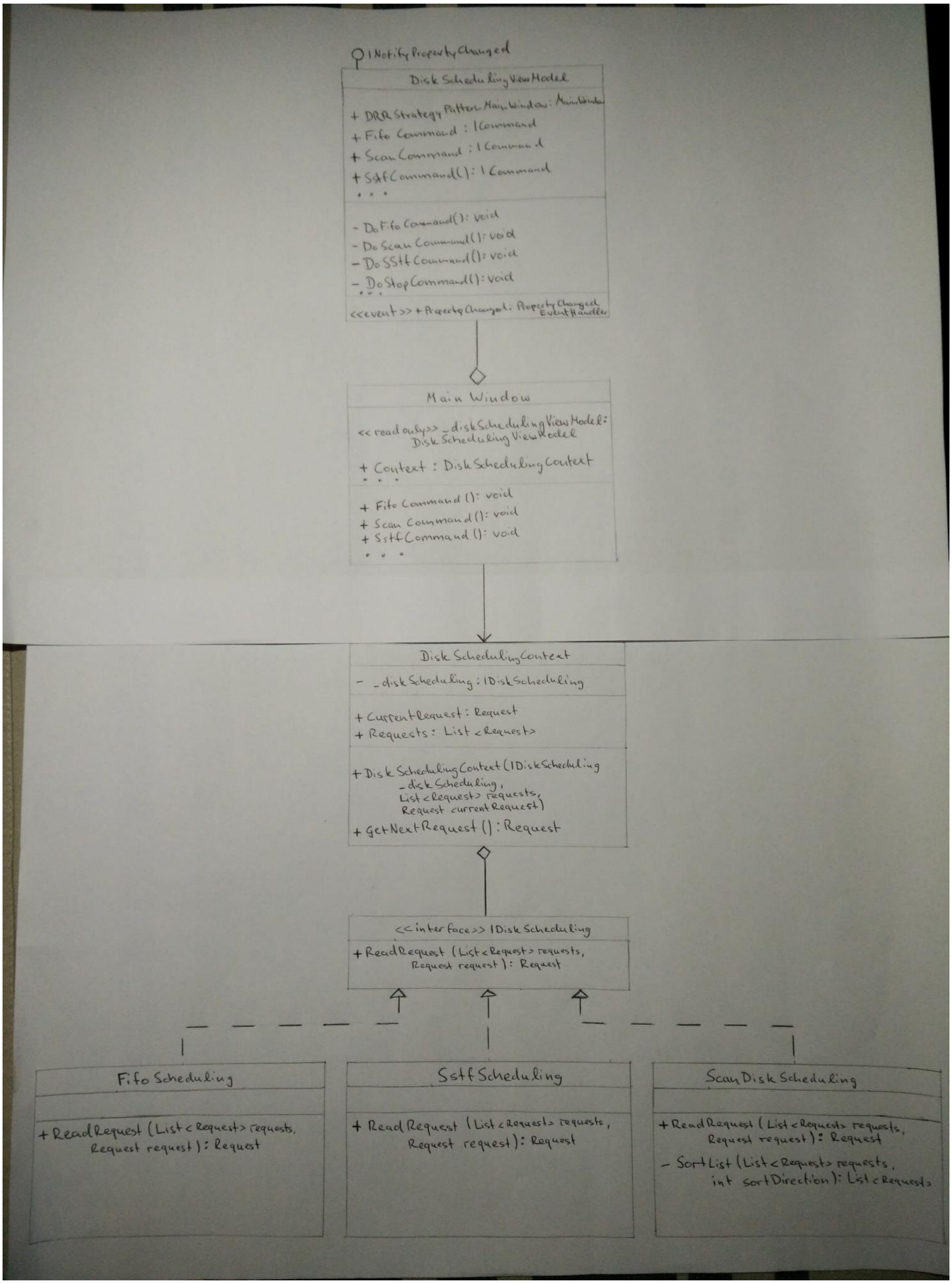

IPropertyChanged
IDiskSchedulingViewModel
+ DRRStrategyPattern.MainWindow: MainWindow
+ FifoCommand : ICommand
+ ScanCommand : ICommand
+ SstfCommand(): ICommand
. . .
- DoFifoCommand(): void
- DoScanCommand(): void
- DoSStfCommand(): void
- DoStopCommand(): void
. . .
<<event>> + PropertyChanged: PropertyChanged EventHandler
Main Window
<<readonly>> _diskSchedulingViewModel: DiskSchedulingViewModel
+ Context : DiskSchedulingContext
. . .
+ FifoCommand (): void
+ ScanCommand(): void
+ SstfCommand (): void
. . .
DiskSchedulingContext
- _diskScheduling : IDiskScheduling
+ CurrentRequest: Request
+ Requests: List<Request>
+ DiskSchedulingContext(IDiskScheduling _diskScheduling, List<Request> requests, Request currentRequest)
+ GetNextRequest (): Request
<<interface>> IDiskScheduling
+ ReadRequest (List<Request> requests, Request request): Request
FifoScheduling
+ ReadRequest (List<Request> requests, Request request): Request
SstfScheduling
+ ReadRequest (List<Request> requests, Request request): Request
ScanDiskScheduling
+ ReadRequest (List<Requests> requests, Request request): Request
- SortList (List<Requests> requests, int sortDirection): List<Request>

FIGURE 1: CLASS DIAGRAM

The figure above depicts a class diagram of the implementation of a WPF application that simulates a simplified operating system's disk scheduling.

Three disk scheduling algorithms, namely *SCAN Disk Scheduling*, *Shortest Seek Time First (SSTF) Disk Scheduling,* and *First-Come First-Served (FIFO) Disk Scheduling,* have been realized in the application using the strategy pattern. As drawn out in chapter 2, the strategy pattern holds an interface which captures abstraction which is found in *IDiskScheduling*. Derived from this interface the disk scheduling algorithms are built in encapsulated units. These units are defined in the application as *FifoScheduling, SstfScheduling,* and, *ScanDiskScheduling*.

In order to complete the strategy pattern, a context is required which adds a layer between implemented strategies and main interface. In the class diagram the context is shown in *DiskSchedulingContext* which fulfills the previous statement. Additionally, this class has the ability to change strategies during runtime and manipulate data presented in the main interface.

In addition to the implementation of the strategy pattern, the Model–view–viewmodel architectural pattern (MVVM) has been utilized. Despite the influence in the application it has minor implication to the strategy pattern. Thus, it is shown only partially in the class diagram to retain a full overview of the present classes but leaving focus on the design pattern. This also holds for the next subchapter which explains classes of the pattern itself but leaving out MVVM or the *MainWindow* class.

3.1 Explanation of classes

This subchapter gives a descriptive explanation of the strategy patterns implementation, such as methods, properties or fields.

3.1.1 IDiskScheduling

IDiskScheduling is the single and main interface from which strategies are derived.

IDiskScheduling		
Type	Definition	Explanation
Method	`+ ReadRequest(List<Request> requests, Request request) : Request`	The interfaces method which strategies derive from

3.1.2 FifoScheduling

Implementation of *IDiskScheduling*.

FifoScheduling		
Type	Definition	Explanation
Method	`+ ReadRequest(List<Request> requests, Request request) : Request`	Returns the first item in the list.

3.1.3 SstfScheduling

Implementation of *IDiskScheduling*.

SstfScheduling		
Type	Definition	Explanation

Method	`+ ReadRequest(List<Request> requests, Request request) : Request`	Finds the smallest difference between *request* and any list item and returns it.

3.1.4 ScanDiskScheduling

Implementation of *IDiskScheduling*.

ScanDiskScheduling		
Type	Definition	Explanation
Method	`+ ReadRequest(List<Request> requests, Request request) : Request`	Initially sorts the list ascending. Find next largest request if not avail. Next smallest request.
Method	`- SortList(List<Request> requests, int sortDirection) : List<Request>`	Sorts the list ascending if sortDirection = 1 or descending sortDirection = -1 and returns it. Runs in a thread and waits for 5 seconds for the sorting task to finish.

3.1.5 DiskSchedulingContext

Layer between Strategies and *MainWindow*.

ScanDiskScheduling		
Type	Definition	Explanation
Property	`+ Requests : List<Request>`	Holds read requests.
Property	`+ CurrentRequest : Request`	Current request which is being processed.
Field	`<<readonly>> _diskScheduling : IDiskScheduling`	Selected disk scheduling strategy.
Method	`+ GetNextRequest() : Request`	Returns the next read reqest based on strategy.

3.2 Features

- Display scheduling process in resizable graph
- Requests can be created manually or randomly
- Switch disk scheduling strategies in runtime
- Reset current running disk scheduling
- Stop disk scheduling
- Output of processed requests is shown
- Final graph can be exported as a png-file

4 Design choices

The implementation of the assignment has been done with regard to reusability, extensibility, and maintainability.

The strategy pattern has been designed with the goal of the points named above. If we inspect the solution, the strategies are separated units and can be reused in any other implementation. Additionally, code is kept clean and can be extended if necessary. Therefore, maintaining the application is time efficient.

Furthermore, the code was designed to reduce runtime, for instance, conditional statements check for null items and return if true.

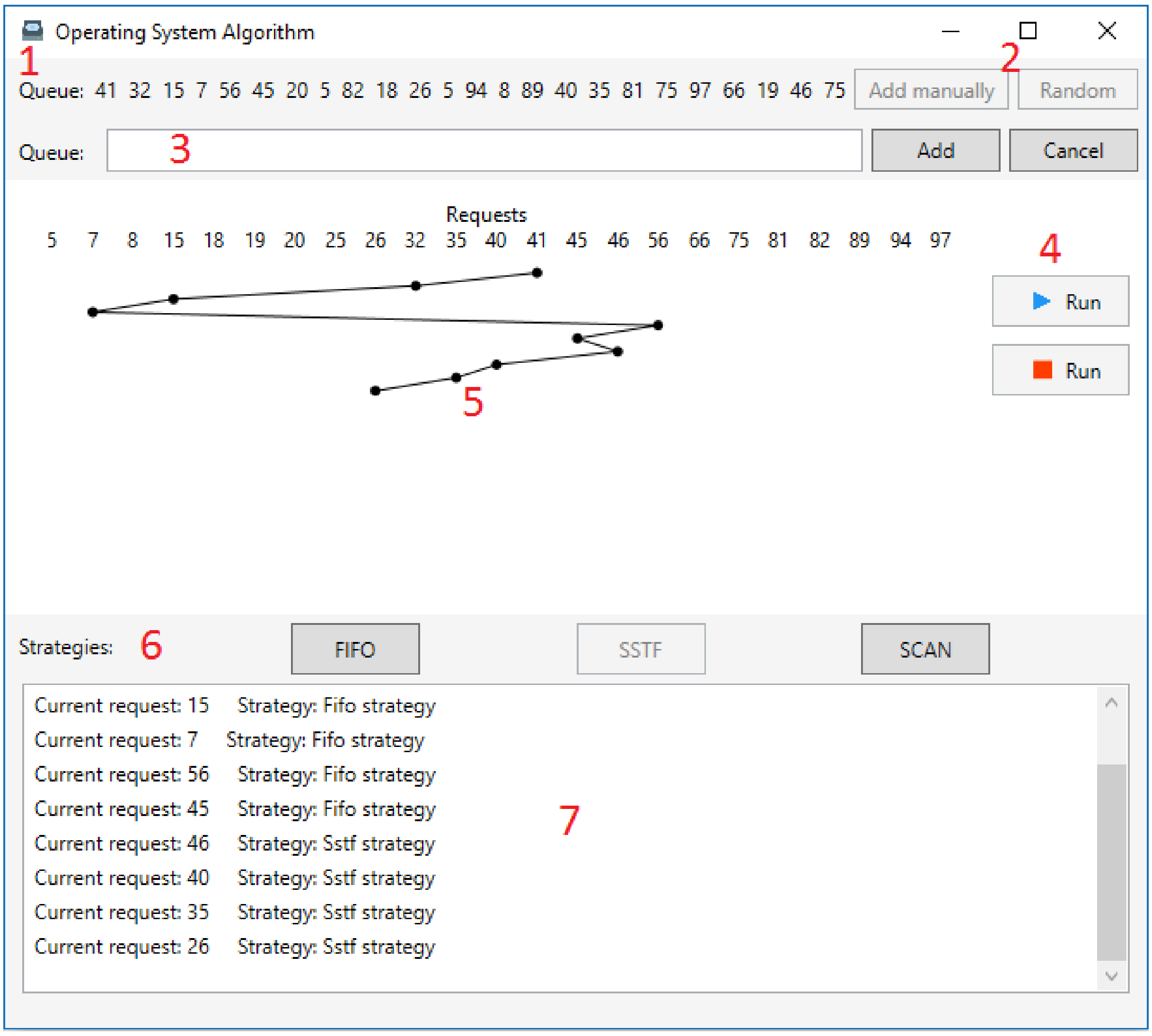
Strategies have been implemented in such a way that unit test are very easily created. Manipulation of the read request list is handled in the context, hence, the strategy is an isolated unit and unit tests do not depend on the context. As a result, repeated code is reduced as well.

The context describes the link between the UI and the strategies. In the *DiskSchedulingContext* all necessary manipulation of the request list is handled, moreover, the class holds the ability to change between a strategy in runtime.

With respect to maintainability, reusability, and extensibility, MVVM has been applied which decouples the UI from the business logic of the application. Thus, developer-designer workflow is applicable and the application is easier to test, maintain, and evolve, to name a few of the benefits of the MVVM pattern.

The UI has an easy and straight forward approach (user friendly). Tool tips are available at each control (buttons, textbox) to support the user. The disk scheduling strategies are showcased as a graph representation to provide a better understanding of the scheduling pattern. A list at the bottom is meant to indicate extra information during the runtime.

5 Graphical user interface



Queue: 41 32 15 7 56 45 20 5 82 18 26 5 94 8 89 40 35 81 75 97 66 19 46 75

```
Current request: 15    Strategy: Fifo strategy
Current request: 7    Strategy: Fifo strategy
Current request: 56    Strategy: Fifo strategy
Current request: 45    Strategy: Fifo strategy
Current request: 46    Strategy: Sstf strategy
Current request: 40    Strategy: Sstf strategy
Current request: 35    Strategy: Sstf strategy
Current request: 26    Strategy: Sstf strategy
```

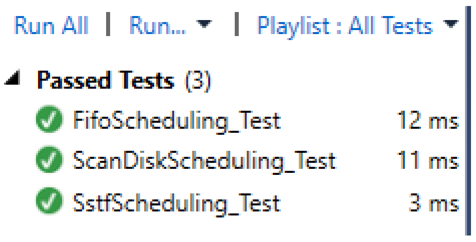
FIGURE 2: USER INTERFACE OF THE APPLICATION

The above figure depicts the user interface where red numbers indicate functionality or controls. More precisely these are:

1. Queue of read requests.
2. Queue items can be added manually or generated randomly with 25 items.
3. If in 2 *Add manually* was chosen the input field indicated in 3 will appear. Items can be entered by entering multiple numbers separated either by comma or space.
4. Controls *Start* and *Stop:*
   a. Start: start processing read requests.
   b. Stop: stop processing read requests.
5. In runtime a graph displays how read request are being processed. On top labels indicate the read request.
6. The horizontal panel holds three buttons that allows changing the strategy between *First in, First out (FIFO), Shortest seek time first (SSTF),* or *Scan Disk Scheduling (SCAN).* Changing can strategies can be done anytime.
7. After requests have been processed, they are in the output field.

6 Unit tests

For each implemented strategy unit tests have been defined to assert correct behavior. Three test methods can be found in *DprStrategyPatternTest*, namely *FifoScheduling_Test, ScanDiskScheduling_Test,* and *SstfScheduling_Test*. The tests compare a hardcoded list to expected values utilizing a *RequestComparer.cs*. Consequently, all test ran successfully.



```
Passed Tests (3)
  FifoScheduling_Test        12 ms
  ScanDiskScheduling_Test    11 ms
  SstfScheduling_Test         3 ms
```

FIGURE 3: SUCCESSFUL UNIT TESTS

7 References

MSDN Microsoft. (2016, September 04). *The MVVM Pattern*. Retrieved from MSDN Microsoft: https://msdn.microsoft.com/en-us/library/hh848246.aspx?f=255&MSPPError=-2147217396

OODesign. (2016, September 04). *Strategy*. Retrieved from OODesign: http://www.oodesign.com/strategy-pattern.html

Source Making. (2016, September 04). *Strategy*. Retrieved from Source Making: https://sourcemaking.com/design_patterns/strategy

Tutorials point. (2016, September 04). *Design Patterns - Strategy Pattern*. Retrieved from Tutorials point: http://www.tutorialspoint.com/design_pattern/strategy_pattern.htm