İzmir University of Economics

2021-2022 Spring

SE311 Project

OS Modeling

Project #2

Team Members

Berkay Sarıtaş

Onur Ural

Arif Batuhan Yıldırımoğlu

Mustafa Dilek

Thought Process

1. File System

1.1 Patterns

Abstract Factory Pattern, Composite Pattern, Singleton Pattern

1.2 Reason

The reason for choosing Abstract Factory Pattern was to encapsulate the creation of file system elements. Since our program was supposed to be capable of implementing different types of these file systems, Abstract Factory would be a great fit. For Composite Pattern, main reason was the structure of the file system that was needed to be created. In this case, files would be individual objects and directories would be compositions of objects. Lastly, the reason for the Singleton Pattern was to create a class that can manage the file system of the OS.

1.3 Participant Mappings

Abstract Factory – FileSystem

Concrete Factory – LinuxFileSystem, BSDFileSystem, NTFileSystem

Abstract Product – File, Directory

Concrete Product – LinuxFile, BSDFile, NTFile, LinuxDirectory, BSDDirectory, NTDirectory

Component – FileSystemElement

Leaf – File

Composite - Directory

Singleton - FileSystemManager

2. Input/Output API

2.1 Patterns

Adapter Pattern

2.2 Reason

The main reason for choosing Adapter Pattern was to be able to convert language DP's interface of writing to files. With the Adapter Pattern every Input/Output API of the file systems could use the language DP's interface for writing data to files.

2.3 Participant Mappings

Target – DpIO

Adaptee – LinuxFileSystemIO, BsdFileSystemIO, NtFileSystemIO

Adapter – LinuxIOAdapter, BsdIOAdapter, NtIOAdapter

3. Application-Device Interruption

3.1 Patterns

Observer Pattern, Singleton Pattern

3.2 Reason

The reason for choosing observer pattern was the relationship between devices and applications. Since applications needed to be notified whenever a device interrupted the system, observer pattern would fit perfectly. For singleton pattern, main reason was to create a class that can manage applications.

3.3 Participant Mappings

Subject – Device

Concrete Subject – Network Port, Keyboard, Mouse

Observer – Observer

Concrete Observer – Application

Singleton - ApplicationManager

4. OS Reset

4.1 Patterns

Template Pattern, Command Pattern, Singleton Pattern

4.2 Reason

The main reason for choosing the Template Pattern was to be able to define an operation for resetting devices inside of a parent class, so that other sub classes can redefine some of its steps without changing the whole structure of this

operation. For the Command Pattern, reason of choosing it was to decouple these reset operations from their devices, so that they can get encapsulated.

Lastly, the reason for choosing Singleton Pattern was to create an OS reset manager class.

4.3 Participant Mappings

Abstract Class – OsDevice

Concrete Class – CPU, HardDisk, IODevice

Command – Command

Concrete Command – DeviceResetCommand

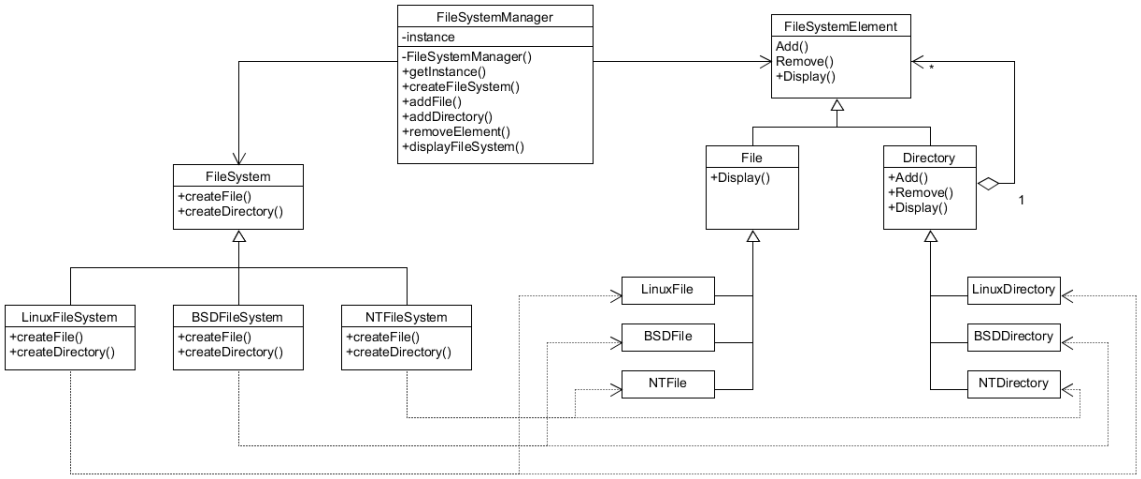
Receiver – OsDevice

Invoker – OsResetManager

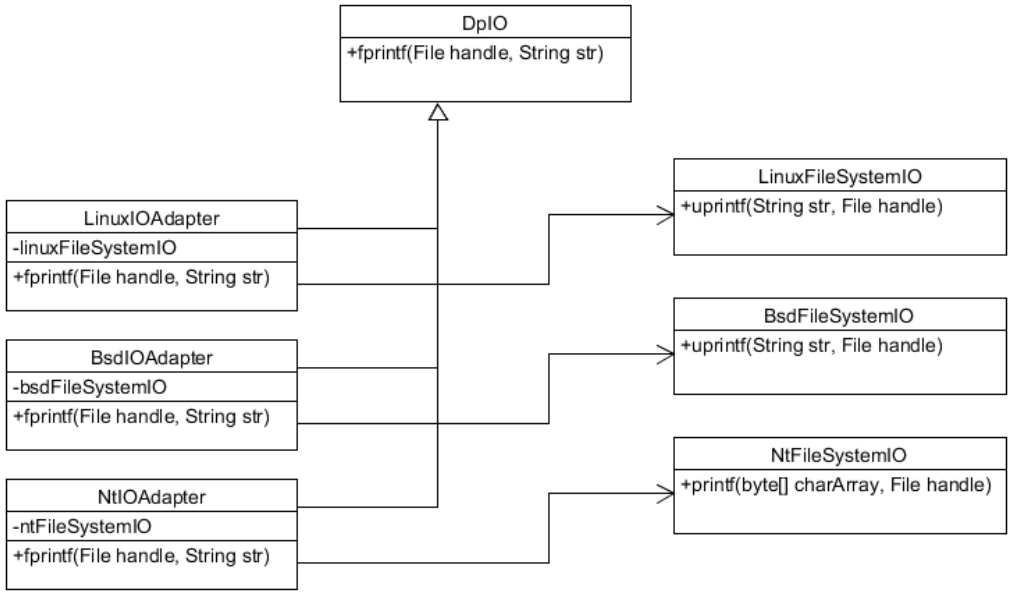
Singleton - OsResetManager

UML Diagrams

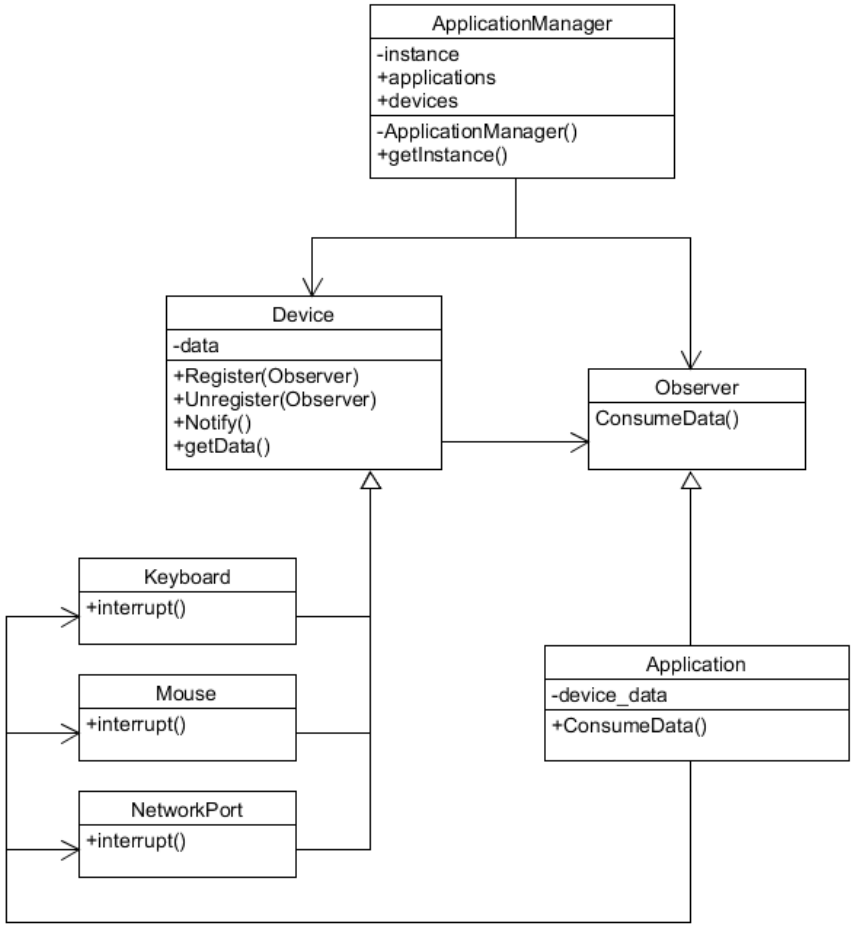
1. File System



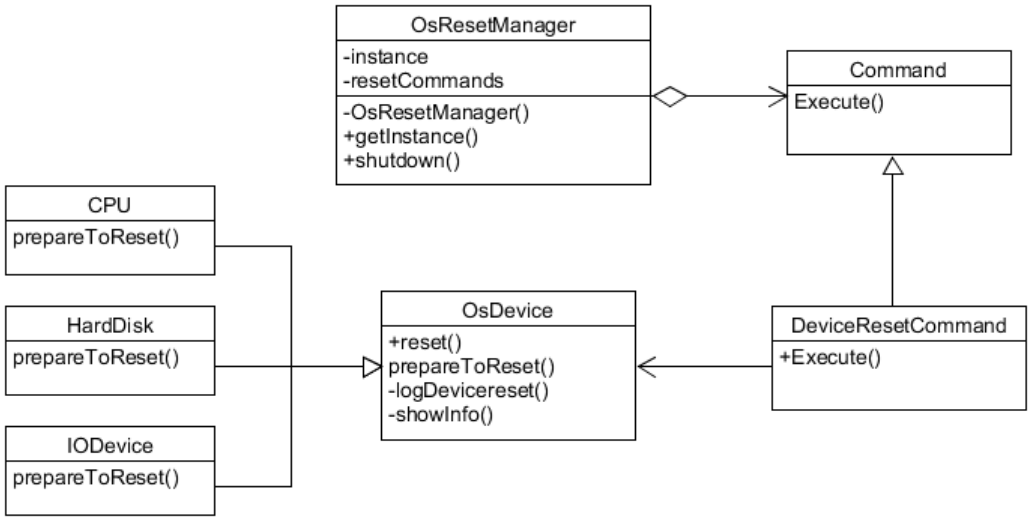
2. Input/Output API



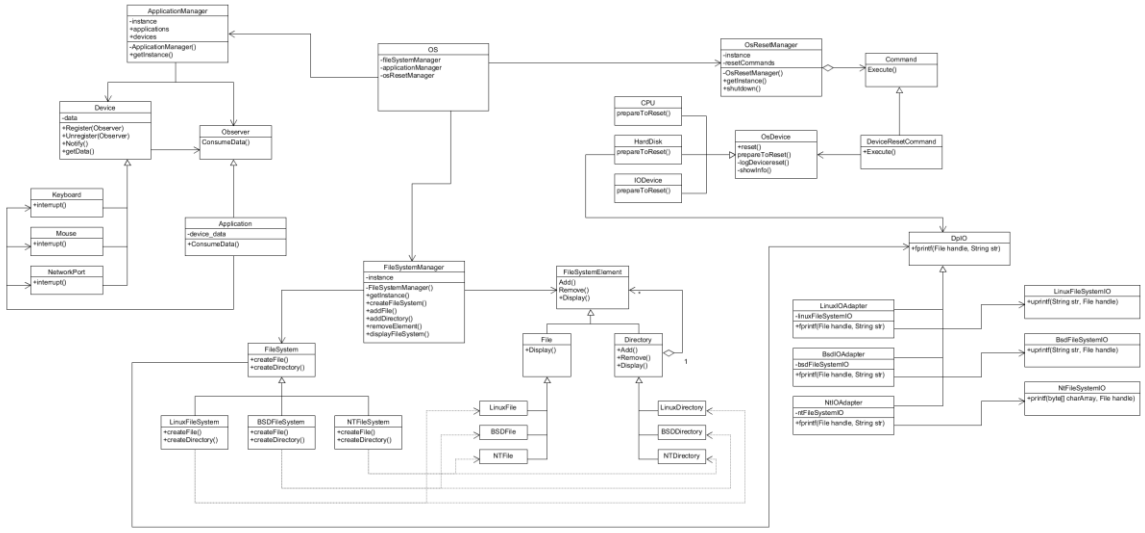
3. Application-Device Interruption


ApplicationManager
-instance
+applications
+devices
-ApplicationManager()
+getInstance()
Device
-data
+Register(Observer)
+Unregister(Observer)
+Notify()
+getData()
Observer
ConsumeData()
Keyboard
+interrupt()
Mouse
+interrupt()
NetworkPort
+interrupt()
Application
-device_data
+ConsumeData()

4. OS Reset


OsResetManager
-instance
-resetCommands
-OsResetManager()
+getInstance()
+shutdown()
Command
Execute()
CPU
prepareToReset()
HardDisk
prepareToReset()
IODevice
prepareToReset()
OsDevice
+reset()
prepareToReset()
-logDevicereset()
-showInfo()
DeviceResetCommand
+Execute()

5. OS Model



Explanation of Implementation

1. File System

1.1 FileSystemElement

This class is a super class for component classes from composite pattern. It has Add, Remove and Display abstract methods.

1.2 FileSystem

This class is the parent class of the factory classes. It has createFile and createDirectory abstract methods. It also has an Input/Output API object reference to handle writing to files.

1.3 LinuxFileSystem

This is a sub class of the FileSystem class. It implements abstract methods for Linux file system elements.

1.4 BSDFileSystem

This is a sub class of the FileSystem class. It implements abstract methods for BSD file system elements.

1.5 NTFileSystem

This is a sub class of the FileSystem class. It implements abstract methods for NT file system elements.

1.6 File

This is the parent class of the file product classes. It is also the leaf class of the composite pattern. It keeps its name and extension as String variables. It implements Add and Remove methods to do nothing since it is the leaf class. It implements the Display method to display its own name. It also keeps a data variable as String, so that it would be possible to write data into files.

1.7 LinuxFile

This is a sub class of the File class. It is a concrete product of the Linux file system.

1.8 BSDFile

This is a sub class of the File class. It is a concrete product of the BSD file system.

1.9 NTFile

This is a sub class of the File class. It is a concrete product of the NT file system.

1.10 Directory

This is the parent class of the directory product classes. It is also the composite class of the composite pattern. It keeps a list of file system elements. It implements Add and Remove methods for adding to this list of elements. It implements the Display method to make every element in the list display their names.

1.11 LinuxDirectory

This is a sub class of the Directory class. It is a concrete product of the Linux file system.

1.12 BSDDirectory

This is a sub class of the Directory class. It is a concrete product of the BSD file system.

1.13 NTDirectory

This is a sub class of the Directory class. It is a concrete product of the NT file system.

1.14 FileSystemManager

This class is a Singleton for managing the file system of the OS. It has a getInstance method to get the instance of the class. It has a createFileSystem method that takes any type of file system as a parameter. It also has addFile, addDirectory, removeElement and displayFileSystem methods to manage the file system.

2. Input/Output API

2.1 DpIO

This is the target class that the other Input/Output classes needs to be adapted to. It has a fprintf method to write into a file.

2.2 LinuxFileSystemIO

This is the I/O API class for the Linux file system. It has a uprintf method to write into a file.

2.3 BsdFileSystemIO

This is the I/O API class for the BSD file system. It has a uprintf method to write into a file.

2.4 NtFileSystemIO

This is the I/O API class for the NT file system. It has a printf method to write into a file.

2.5 LinuxIOAdapter

This is the adapter class for the LinuxFileSystemIO class. It is a sub class of the DpIO class. It overrides the fprintf method by using the LinuxFileSystemIO's method instead.

2.6 BsdIOAdapter

This is the adapter class for the BsdFileSystemIO class. It is a sub class of the DpIO class. It overrides the fprintf method by using the BsdFileSystemIO's method instead.

2.7 NtIOAdapter

This is the adapter class for the NtFileSystemIO class. It is a sub class of the DpIO class. It overrides the fprintf method by using the NtFileSystemIO's method instead.

3. Application-Device Interruption

3.1 Device

This class represents the Subject class. It is a base class for all other device classes. Device class keeps a data variable that represents the input that will be given when it interrupts the system. Register method is used for attaching

observers to this subject by adding them to a list. On the other hand, Unregister method does the opposite; it detaches observers by removing them from the list. Notify method is used for finding the most prioritized application and notifying that application for it to consume the input data. Lastly, the Interrupt abstract method is used for getting an input from the device and calling the Notify method. This method is implemented in the child classes.

3.2 NetworkPort

This class is a child class of the Device class. It implements the Interrupt method to get the data that is in the network port and call the Notify method.

3.3 Keyboard

This class is a child class of the Device class. It implements the Interrupt method to get an input from the keyboard and call the Notify method.

3.4 Mouse

This class is a child class of the Device class. It implements the Interrupt method to get a click input from the mouse and call the Notify method.

3.5 Observer

This is the interface for observer classes to implement. It has a ConsumeData method.

3.6 Application

This class represents the Concrete Observer class. It observes the devices to get an input data when any device interrupts the program. ConsumeData method is used for getting a notification of input data. It gets the device that interrupted the program and consumes its input data.

3.7 ApplicationManager

This class is a Singleton for managing applications. It has a getInstance method to get the instance of the class. It also holds the applications and devices in lists.

4. OS Reset

4.1 OsDevice

This is the super class of the OS devices. It has a reset method that has ordered methods inside of it. It has a showInfo method to diplay the information of the device before resetting. It has a prepareToReset method for reset operations. This method will be implemented in sub classes. It also has a logDeviceReset method that is called after a reset to log the reset operation.

4.2 CPU

This is a sub class of the OsDevice class. It implements the prepareToReset method by terminating all processes.

4.3 HardDisk

This is a sub class of the OsDevice class. It implements the prepareToReset method by writing the buffer data inside of a file and closing all files.

4.4 IODevice

This is a sub class of the OsDevice class. It implements the prepareToReset method by disconnecting the IO device.

4.5 Command

This is the interface for the command classes. It has an Execute method for commands to implement.

4.6 DeviceResetCommand

This is the concrete command class. It keeps an object reference of an OsDevice class. It implements the Execute method by calling the OsDevice's reset method.

4.7 OsResetManager

This is the Singleton class for managing OS reset operations. It has a getInstance method to get the instance of the class. It has a list of commands and its set operation. It also has a shutdown method that executes these reset commands one by one.

5. Conclusion

In this document, thought process of choosing patterns, UML diagrams of these patterns and explanation of pattern implementations were shown and explained.