Fontys ICT

STATE PATTERN

Design patterns

Jan-Niklas Schneider, Georgiana Manolache
10-11-2016

Contents

1 Introduction

The goal of this document is to give an overview of the state pattern by giving an example implementation which features a simplified industrial 3D print process. Furthermore, reusability, extensibility, and maintainability of this pattern are elaborated. Also, the implementation, its unit test and graphical user interface (GUI) are reviewed.

2 State pattern

The state pattern is a software design pattern which describes an object-oriented state machine. The pattern has two application areas. Firstly, an object which must change its behavior at run-time depending on a state. Secondly, the pattern is utilized in applications that uses a significant amount of case statements that vector flow based on the applications state (SourceMaking, 2016).

The figure below depicts an UML diagram of the state pattern.

Initially, a *Context* class is created which is a single point of entry. *Context* holds a *State* and is able to either set or get a state.

Secondly, a *State* interface is defined which sets the actions of its derived classes.

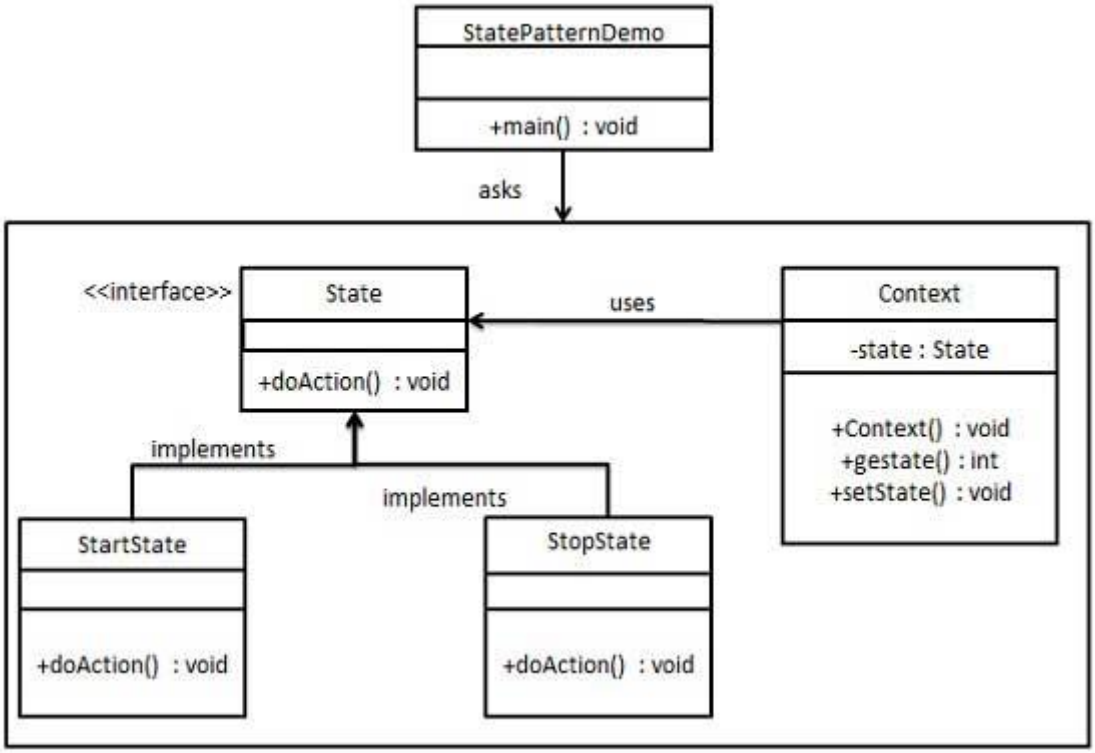


Figure 2-1: UML diagram of state pattern (TutorialsPoint, 2016)

3 Implementation

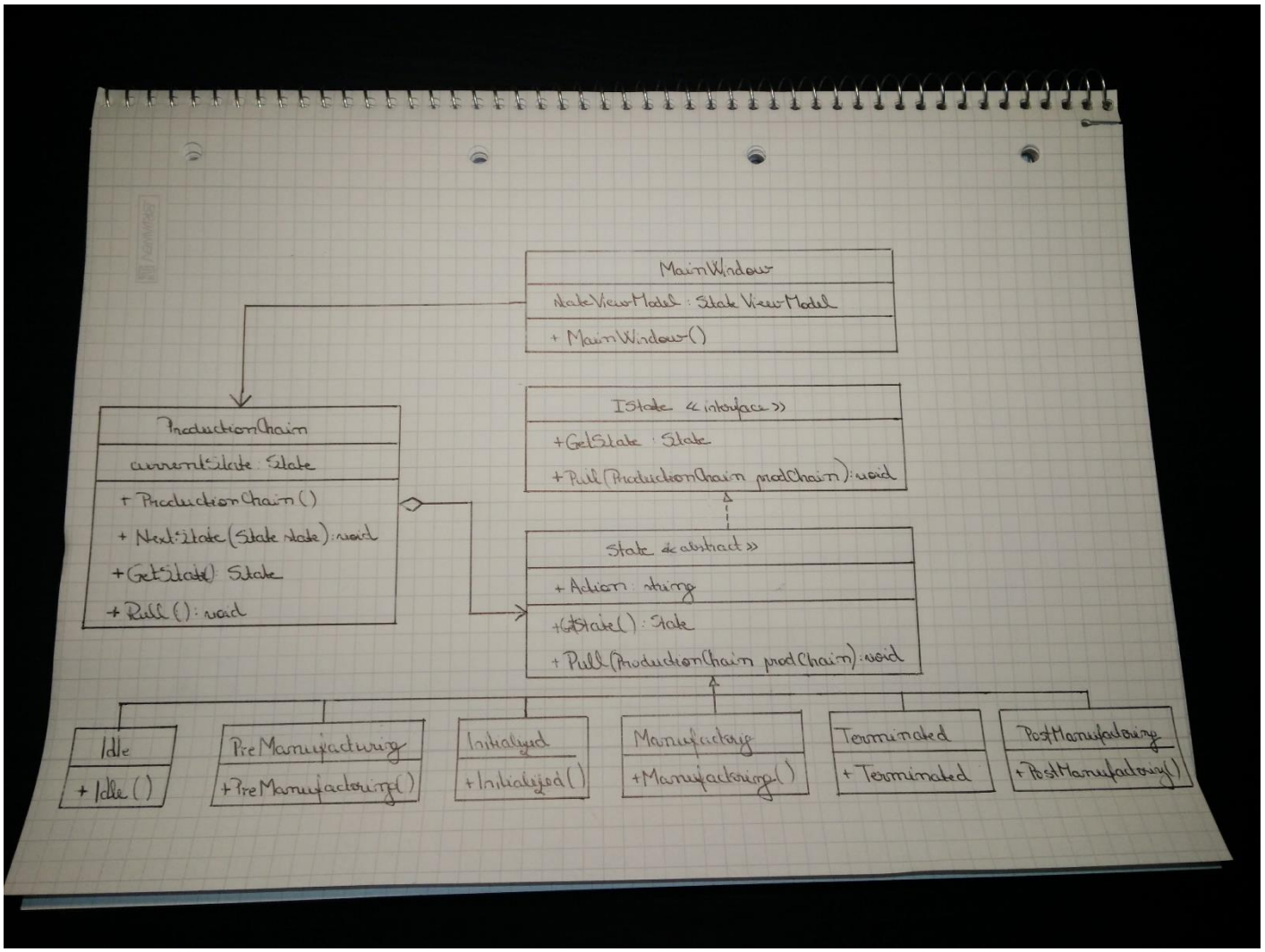
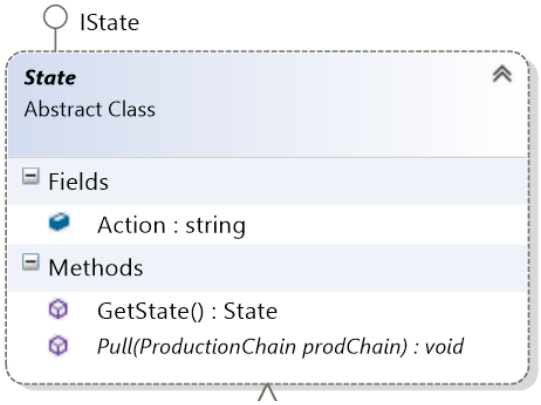


Figure 3-1: UML diagram of the state pattern

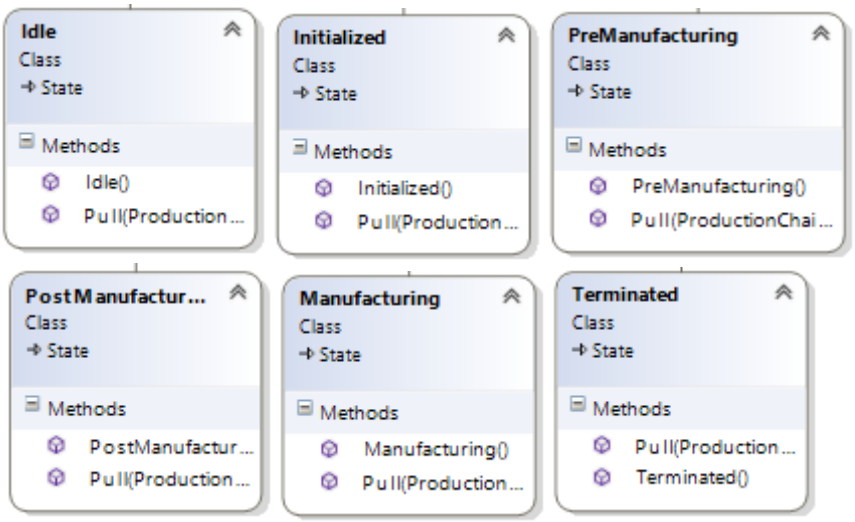
3.1 Explanation of classes

1. *IState* is an interface with two methods *GetState()* and *Pull()*. These methods define the concrete states for all derived classes.

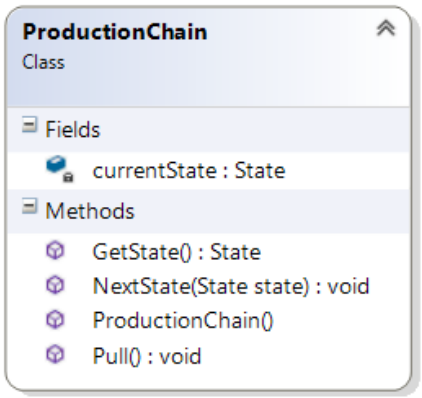
2. *State* is an abstract class which inherits from *IState*, hence, it implements the interfaces methods, namely *GetState()* and *Pull()*. Additonally, it holds a string *Action*.



3. The classes depicted below inherit from *State* and describe concrete states of the state pattern.



4. *ProductionChain* is the context class of the state pattern. It holds the *currentState* and has three methods *GetState(), NextState(),* and *Pull().*



3.2 Features

The application has a simple and straightforward user interface. Users are presented a simplified, fully automated industrial 3D printer which demonstrates the state pattern by showing a print process. The print begins with initializing the machine, followed by premanufacturing, manufacturing, and postmanufacturing. Then the machine terminates itself. During the print the machine can be emergency stopped.

4 Design choices

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

The **reusability** of the pattern is decent. While the context and interface are mostly reusable for other applications (with the ease of refactoring), the concrete states are very dependent on the application's use case, therefore, won't be reusable in another context.

In terms of **maintainability** the pattern is difficult to maintain. The classes derived from *State* are tightly coupled to its neighbors which results in dependencies between the sub classes. This decreases maintainability significantly and has drawbacks for unit tests since tight coupling will result in more complex test cases.

The state pattern shows low **extensibility**. As mentioned in the previous section, there is a high dependency on sub classes of *State* which proves the extensibility of this pattern to be difficult. The "state machine" represents a closed logic in which adding additional logic is burdensome due to the coupling of the classes.

With respect to maintainability, reusability, and extensibility, the Model–view–viewmodel (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.

5 Graphical User Interface

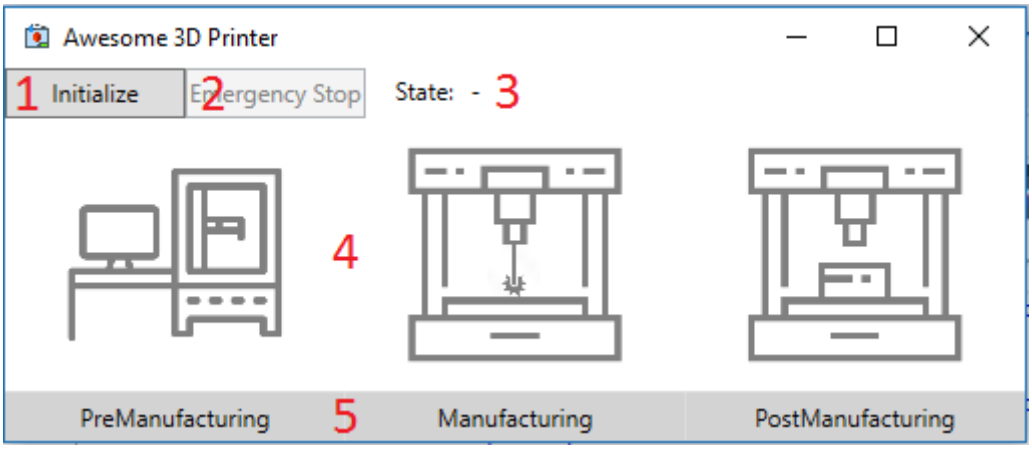


Figure 5-1: Graphical user interface

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

1. Initializes the 3D printer and starts the printing process.
2. During the printing process the printer can be stopped via an "Emergency Stop" which is indicated on panel 5 through red color.
3. The label indicates the current state of the state machine.
4. This horizontal panel shows together with 5 the current progress. The image of the current state will light up when in the respective state.
5. A horizontal panel which is similar to 4 shows current progress by lighting up green. Both 4 and 5 run always in parallel.

6 Unit tests

To assert correct behavior of the state machine, one unit test has been created, namely *StatesValidOrder_Test*. The test starts the state machine and verifies if the correct order of states has been cycled through. In order to compare the *State* objects, the class *StateComparer* was utilized. Subsequently, the test ran successful.

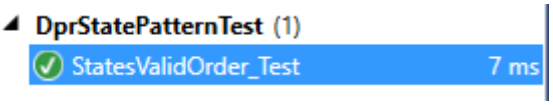


Figure 6-1: Unit test result

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

SourceMaking. (2016, October). *State*. Retrieved from SourceMaking.com: https://sourcemaking.com/design_patterns/state

TutorialsPoint. (2016, October). *Design Patterns - State Pattern*. Retrieved from TutorialsPoint.com: https://www.tutorialspoint.com/design_pattern/state_pattern.htm