Software Engineering – Assignment 1
MSc Software Development
Eoghan Malone - 2227056M

Assumptions

- The model should be of reasonable magnitude to demonstrate my understanding but not overtly complex. The model has been kept lightweight.
- It is assumed that both design patterns and architecture should be considered in the solution as it was stipulated that we would need to complete these lessons in order to answer the questions fully.

Solution 1 – Reduce coupling

Unlike cohesion, where operations can be grouped or split out into more normalised classes, coupling is unavoidable. However, it can be reduced significantly by the use of the Observer pattern and standard loose-coupling that interface implementation provides.

The observer pattern is ideal for reducing coupling in situations where there is a link between something that updates something else when there is a change. In this case, when the document is changed by the `DocumentInterface`, the observable class updates the view by monitoring for change from the `SetChanged()` operation.

A further tier of loose-coupling is implemented via the `DocumentInterface` interface. All editing operations are implemented via this interface and the `Document` class will never know of their existence. Further classes could be coupled to this interface and this would not affect the Document class. It only needs to be aware of the `DocumentInterface`, not what it does and how it does it.

The entire implementation is not far off a complete MVC implementation; in fact the Observer pattern forms the basis of the MVC architecture which will be discussed in solution 3.

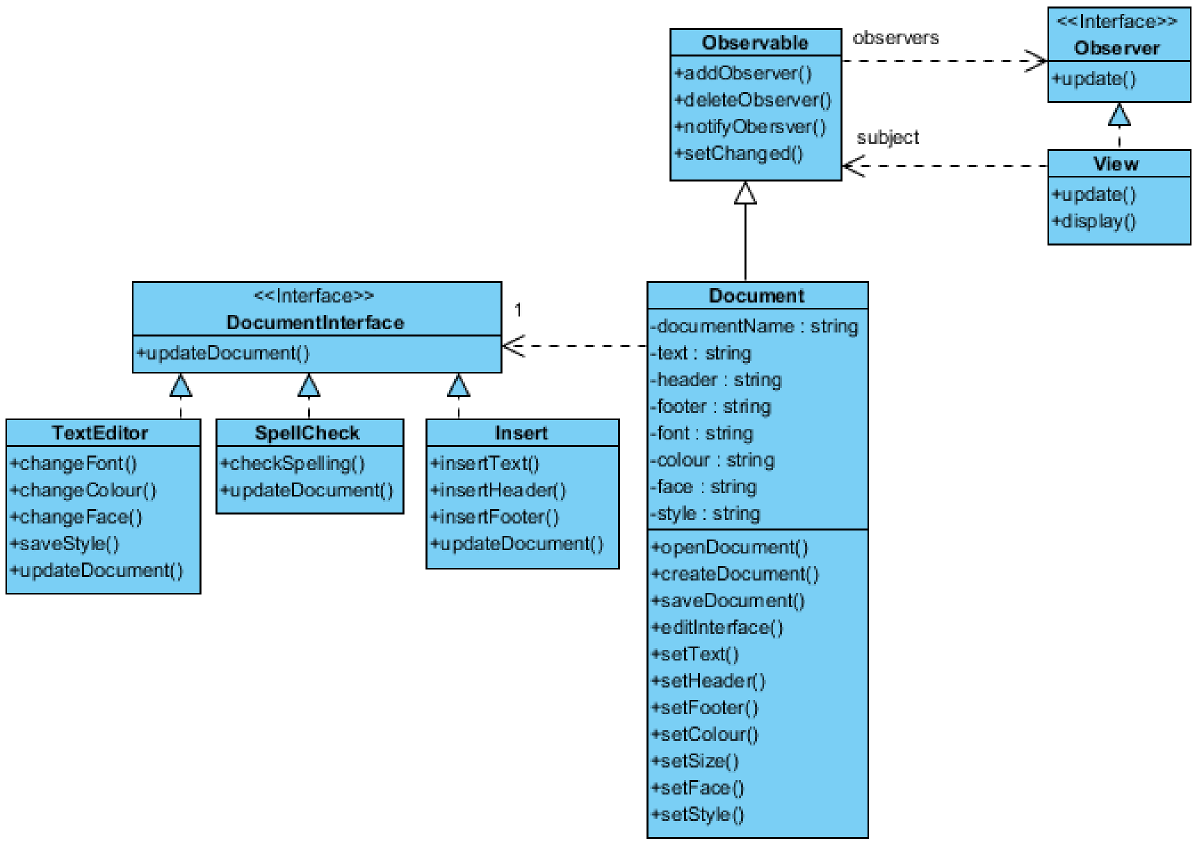

Figure 1 – Reducing coupling via the observer pattern and an interface implementation

Solution 2 – Increase cohesion

Increasing cohesion is a subjective task, and can be done in many different ways. The primary purpose of the façade pattern is to provide an easy interface for programmers. I have chosen this pattern as it contrasts sharply from the previous task. The façade pattern *absolutely increases coupling*, more so than any other pattern. However, it can be implemented in a way that *increases cohesion* at class level by ensuring that each class has exactly the amount of operations required to perform its task and no more. The pattern makes use of the principle of least knowledge as the client will use the `WordProcessorFacade` class to access all of the applications operations. This makes it ideal to present each class on its own directly connected to the façade and establishing any dependencies each class needs with other classes.

The singleton pattern has been employed to ensure that the façade can only ever be called once; otherwise the implementation will lose its cohesive nature through repetition.

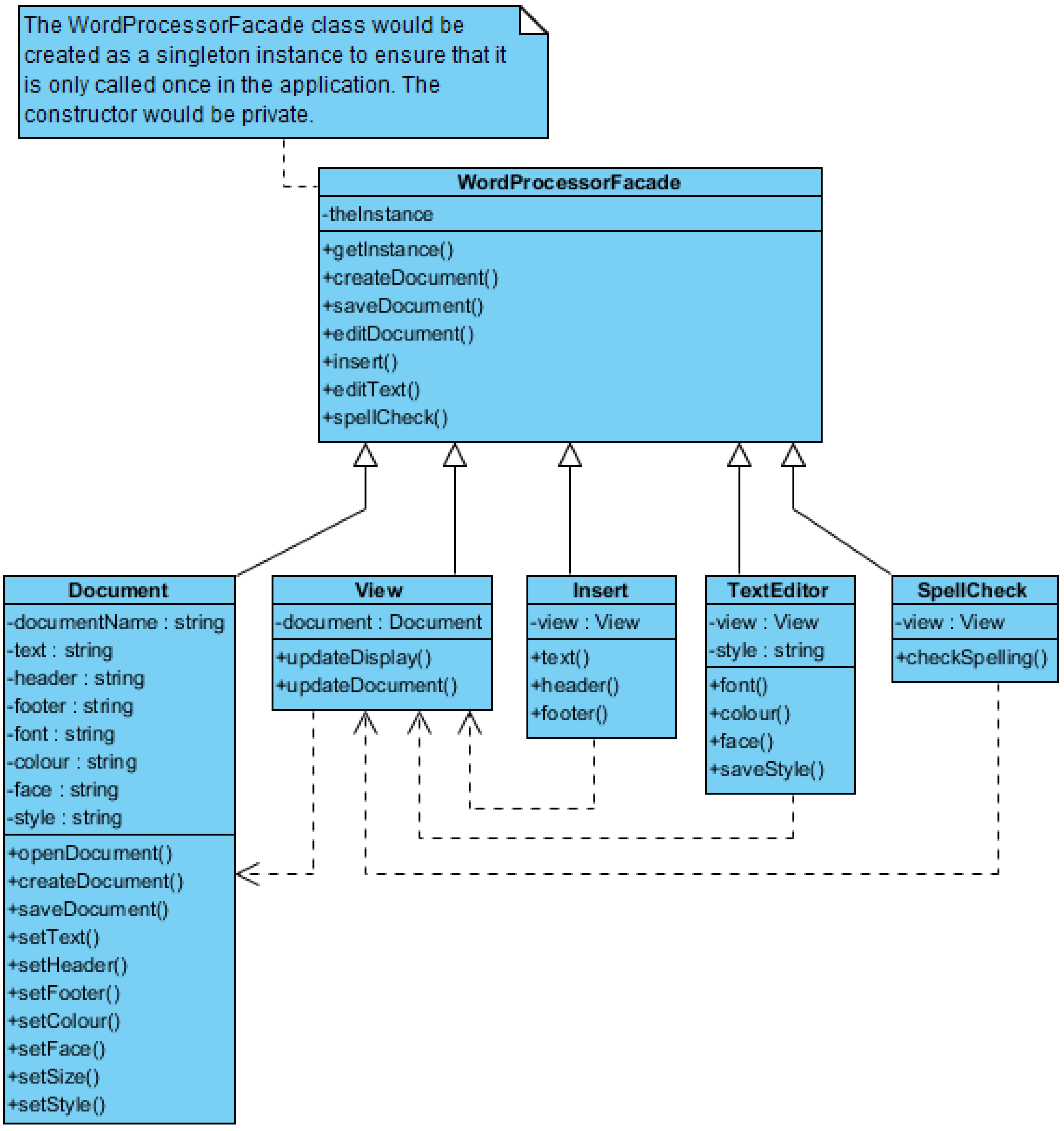

Figure 2 - Facade implementation to increase cohesion

Solution 3 – Ideal compromise between coupling and cohesion

For the 3rd solution I have utilised the MVC architecture which makes use of the divide and conquer principle as each layer of the framework can be developed independently. Cohesion is increased at class and operation level as the business logic, view and control componests are separated. As MVC architecture separates concerns into 3 components, this makes the task of both decoupling and increasing cohesion easier as the complexity of each component is reduced. While there is still an element of coupling, the coupling between the complex functions consists of the communications between the View, Controller and Model [Document in this case]. The notification of the changes to the model to the view [notify] would be provided by the `Observer` pattern. I have omitted this from the diagram to simplify as it has been discussed and demonstrated in solution 1.

As this architecture advocates the divide and conquer principle, it makes testing easier as each component can be tested separately and each component can be developed independently of the other components. This can only be achieved through an architecture that reduces both coupling, and increases cohesion. This why the MVC architecture is a good fit for this problem.

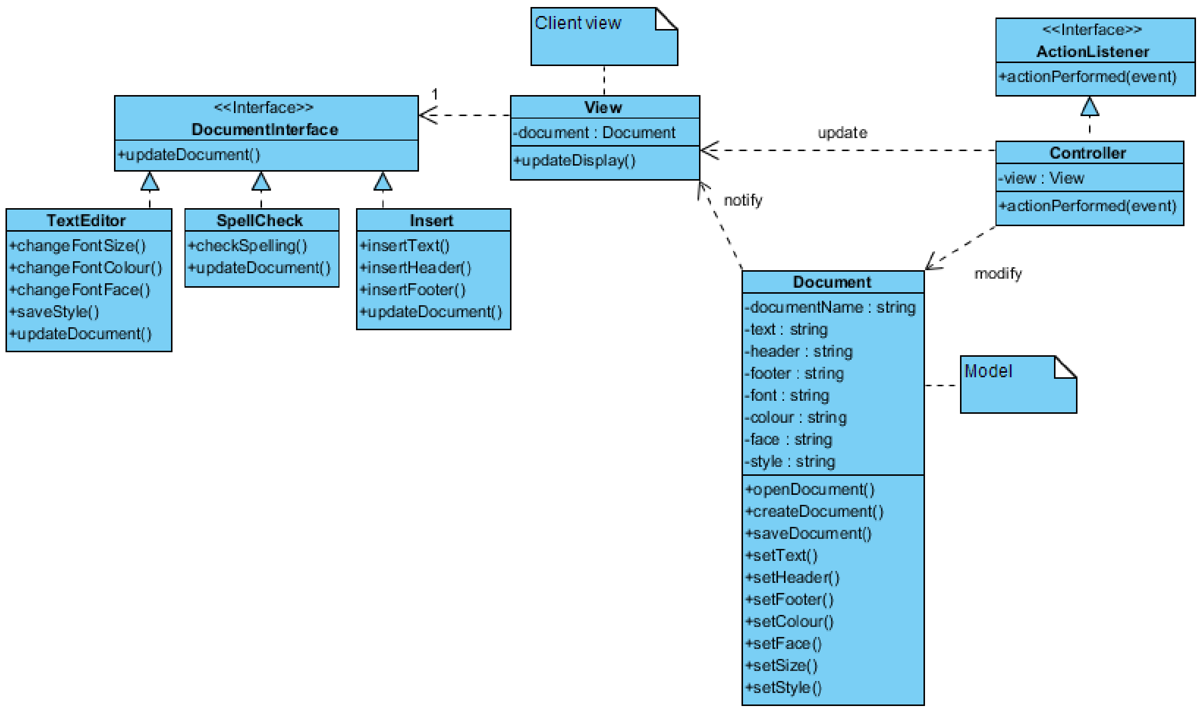

Figure 3 - Model View Controller Architecture [MVC]