Third Year Project 2013/2014

Report 4: Design Patterns

A Sudoku Solver Using Constraint Satisfaction (& Other Techniques)

Thomas Paul Clarke

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Dave Cohen

Design patterns provide a structure for a program, which is done by giving a concept or design of a program to solve a problem. A design pattern does not specify code but is used to simply a complicated problem by spreading it into defined classes or methods, which are predetermined by the pattern. There are huge varieties of patterns and many different designs for the same problems. This is due to the few limitations that are in place, as it just needs to work for the programmer and fix the problem, which often helps to make the code more understandable with a set layout.

Each pattern will define a structure. It will have to be designed (often) with links to other classes and objects with their inheritance defined to the programmer. Therefore it is clear what will have to be made, which will make it easier. They are designed so that if they do specify an object or class they will be open to change. They will not need to be set exactly because they are defined due to each program being build for a different purpose.

Some of the design patterns that I have researched are linked to what I'll be needing within the Sudoku program. There will be several design patterns incorporated within the program. They will all be for their own individual reasons and playing on their unique strengths.

The reasons the program will need some design patterns are:

There is a user interface, that will contain the buttons, and this will be linked to the rest of the program. It is necessary to have a design pattern that will allow a separation of the user interface and its event listeners, which allow it to be handled independently to the other program functions.

This is possible by using the MVC design pattern, which stands for model, view & controller. These are all classes that will act with each other and break up the user interface part of the program. Acting upon the user inputs the event handlers (that are placed within the view) pass the input to the controller that (depending on the input) pass it to the model. This is the gateway to the rest of the program, which is the way that the design pattern handles requests from the interface like a button. The view is used to display the interface and contains the code that is linked to that like the event listeners. The controller is used to request information from either the view or the model, which is able to run the program and do the necessary calculations that the user requires.

Sudoku is a puzzle that requires 81 instances of an object. In this case each cell within the grid will have to be created and this will represent the numbers that are input

by the user. It is necessary for each of these objects to be created exactly the same so they can be handled the same throughout the whole program.

There are options of design patterns to use due to this being a very common issue. There will need to be some debate about which is most likely to benefit, the program or the programming difficulty. There are ways that these design patterns are able to work together.

The flyweight pattern creates new objects but ones that differ from each other. Similar objects are not created as they do not need to be. The pattern allows similar objects to be used instead, for example within a Sudoku grid at the beginning of the game many of the cells are not given a value until the user sets one. This means at some point only a few objects will be set with values that differ from others. Due to this change there is also the factor that there will be only 10 possible values, null-9, which allows the pattern to use each number as a way to create the cells by just storing the position of each cell. It benefits the program because it is more effective at reducing the number of objects created and decreasing the memory needed, which also increases better performance[1].

Flyweight is designed by having a reference to every object that could be considered that is the flyweight object. For example, in the Sudoku puzzle there are only 10 variations of what each cell can be. This means that the flyweight will have those 10 different objects stored within a hash map and the only data needing to be stored is each cell that equals those objects. [2]

Alternatively the Composite design pattern essentially groups objects and treats them in the same way as one would be treated. This could benefit the handling of the objects within the system as there could be many and a lot with the same values. This allows there to be a group that can change, for example, rows and columns can be grouped or objects with the same values, which allows them to be checked against constraints.

The process of this pattern is to put the objects into a tree where it can be added and removed from that group by the simple methods. This however, does limit the functions, but then allows there to be a link between the objects and their values. With this process comes the ability to put the cells into trees.

Sudoku uses constraints on the objects and this needs to be implemented at different times. Often it would be effective to have a design pattern that would allow the implementation of the algorithms, which are essential. Each constraint could be done in a different method and there can be a runner method that allows the object to only use the constraints that it needs to instead of run them all.

Strategy pattern allows the program to use a variety of independent algorithms that can be performed to the object and it does not matter which. It allows a structure that each object can be handled by the desired methods that could be helpful in the project to restrict unnecessary checks taking place, which could reduce the time taken to complete constraints.

---

[1] http://www.tutorialspoint.com/design_pattern/flyweight_pattern.htm
[2] http://en.wikipedia.org/wiki/Flyweight_pattern

This pattern gives the program increased flexibility and allows each cell to be treated independently with the algorithms that it needs. It would also be beneficial to have a system that does split up by checking for the constraints within the program. This is because I have noticed within the concept programs the code can become cluttered when there are several constraints.

Within the Sudoku there are many data structures that are used in several ways to allow the program to function correctly and with the fastest methods to do so. However, on these data structures there will need to be a significant amount of logistics [3] performed on each. It would be beneficial to have a pattern that would allow the separation of the data structure from the logic so it would be independent.

Visitor pattern does this by separating the objects logic and structure by splitting their classes and allowing them to be called when needed, so that they are independent and each object will be able to handle the logic that is needed at the correct time. To do this the classes are created for each algorithm and then the object is required to apply the logic if it is needed.

Within the program, as a Sudoku solver, this could be beneficial to limit and monitor what is being called and how each cell is being influenced. This is especially during the design stage to really see what is happening having separations will make the program simpler.

There have been a lot of ways to handle and manipulate an object but there are design patterns that allow easy and effective ways to create objects. It is possible to incorporate a design pattern that create objects but incorporate more effective ways to create an object.

For example the singleton pattern creates a class that can create one object every time it is invoked, which allows the process to be very quick and easy. Meaning that the creating of an object is simply calling the method. It is easy to then limit the number of objects as the amount of objects depends on how many times the class is invoked.

To link this to the Sudoku program it will allow the correct number of cells to be created, allowing other design patterns to be incorporated into this could be necessary in terms of creating the objects by the flyweight pattern. For example, when it creates its objects it could use a singleton, meaning it is possible to get the benefits from both patterns.

Design patterns give a massive wide range of functions and help throughout a project from the backend of the program to the GUI; this really can push a programmer to try new patterns to improve their programs. They can be used repeatedly within projects in the future as each pattern can be used for many different uses and reasons.

However there are problems with design patterns. The obvious is that there are so many, or that for each problem there will be a pattern that is linked and can be used for that situation. The problem here is that you have to find the right pattern, which can be quite labour intensive as it requires research or previous knowledge of the patterns. Although, when working within a team each person will have their own experience with patterns and their way of programming due to patterns soon becoming a regular way to

[3] http://www.codeproject.com/Articles/186185/Visitor-Design-Pattern

program in terms of structure if it helps. Within a group there can be too many ideas for design patterns that could actually harm the programs performance instead of help it.

Overall I will try to incorporate most, if not all, of the design patterns that I have described above. They will help the process of coding by making it understandable as well as help the performance within the system by making it as effective as possible.