PRA Major Coursework Project Report

Extension of the Shark Tracker Application

Introduction:

This report will focus on the Shark Tracker application, and extensions which could potentially be made to it in order to improve its features, making it more accessible to both a more casual user, as well as someone using it for a more dedicated purpose.

To allow proper analysis of these proposed changes, a number of analytical techniques shall be applied. These will take the form of a Domain model, a Hierarchical task analysis, Virtual window models and finally showing the Global navigation structure.

A proposed extension is the ability to add information pertaining to new Sharks directly into the application. This will allow instantaneous updates to the data stream, without having to rely on the external API.

Domain Model:

This will describe the different parts of the application and how they are related.

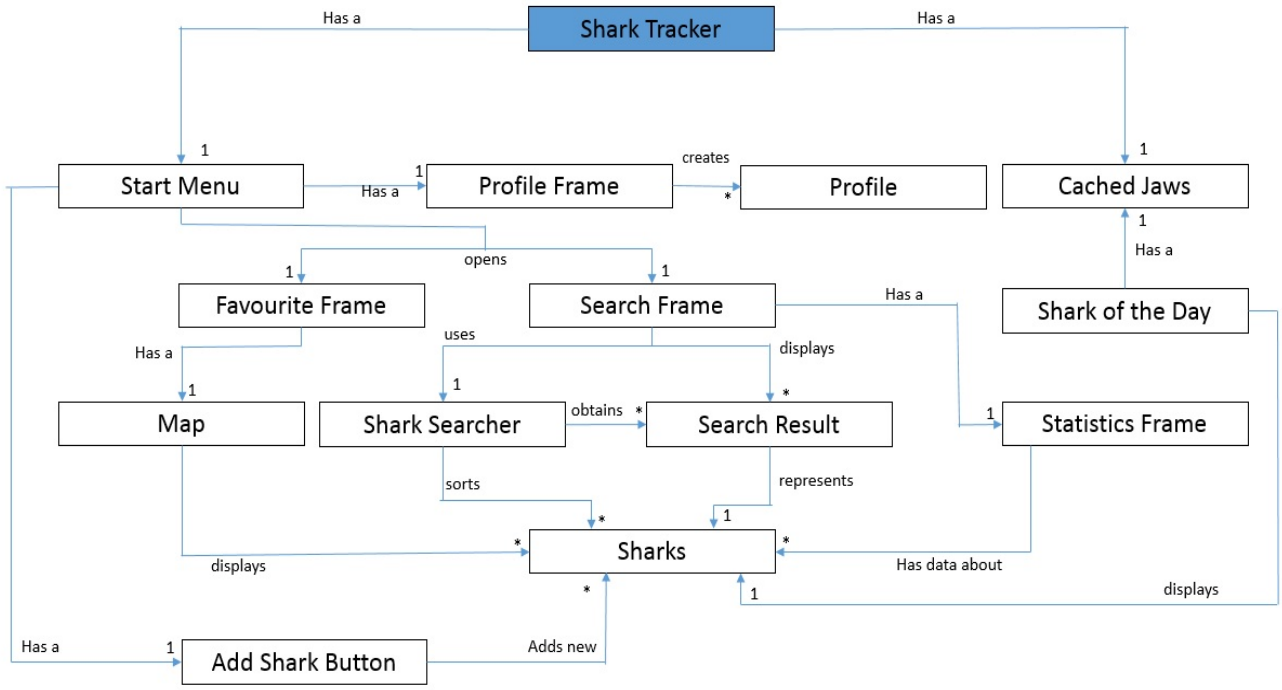

Figure 1: The domain model of the Shark Tracker Application

The domain of this application involves Sharks, and information about them. The main functionality of the application is to search for sharks and the ability to keep track of favourite sharks, if required.

Figure 1 shows a representation of the main concepts that were required of the application, including the aforementioned extension. The arrows demonstrate how each entity is related, while the 1 and * symbols denote the cardinality of the relationship.

As there may be more than one user, the ability to create more than one user profile is available. Different users may have different desires, which may require the ability to add new shark data to the application as and when they wish. It is therefore logical to include the ability to add new sharks, in the domain in question.

Other features included in this domain are: the caching of data from the Jaws API - represented by the Cached jaws entity, a novelty Shark of the day feature, statistics about certain sharks and even a map to show the geographical location of favourite sharks. All of these features are provided, and therefore satisfy the requirements of the application.

One downside to this technique is that it requires an additional development step. This could lead to delays further down the design pipeline, as well as costing additional resources.

Hierarchical Task Analysis:

This analysis represents the order that tasks, and their subtasks, need to be completed during execution of the application. This particular rendition will deal with the starting of the program, navigating to the Add Shark option, adding one, and then exiting the program. This involves identifying tasks and any subtasks they may entail, until there is no more tasks to be undertaken.

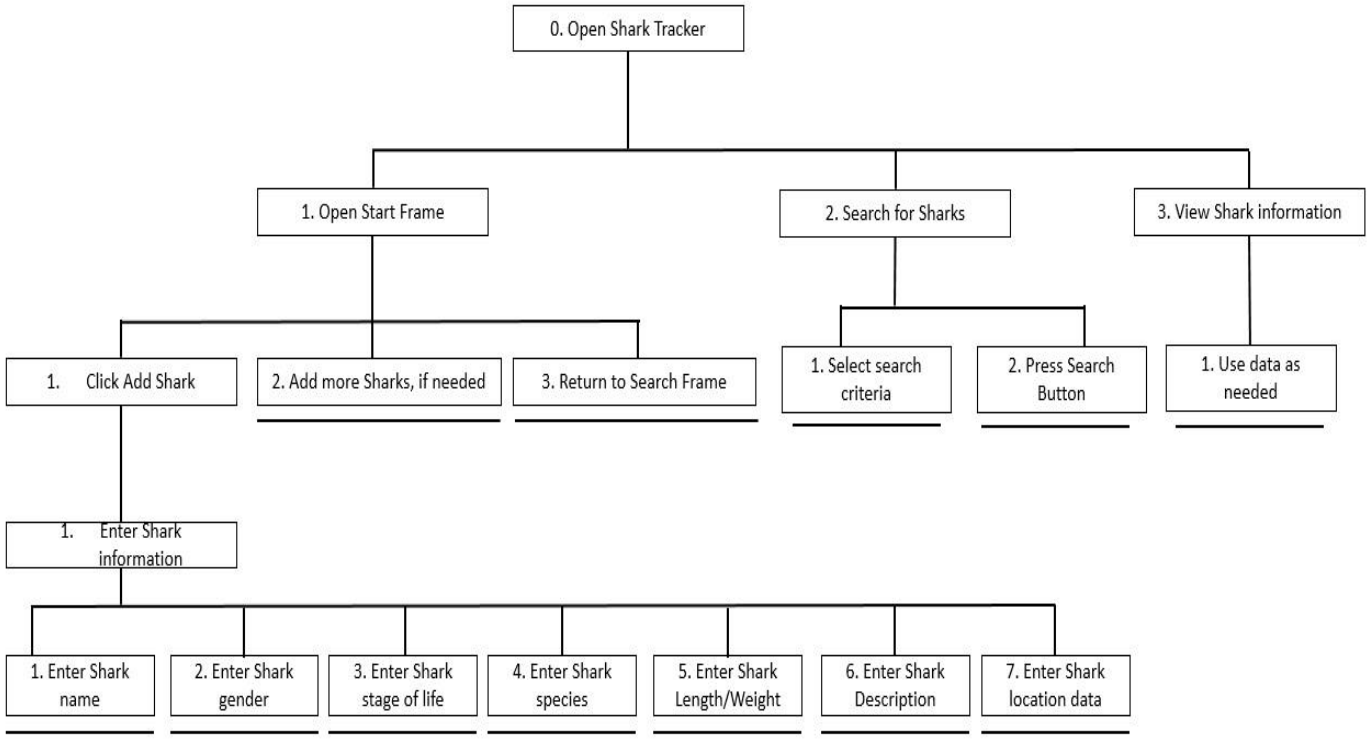

Figure 2: The Hierarchical task analysis of the Shark Tracker application

The following are plans based on the hierarchical task analysis:

Plan 1: Do 1, 1.1.1, 1.1.1.1, 1.1.1.2, 1.1.1.3, 1.1.1.4, 1.1.1.5, 1.1.1.6, 1.1.1.7, 1.2, 1.3, 2.1, 2.1, 2.2, 3.1
Plan 2: Do 1, 1.1.1, 1.1.1.1, 1.1.1.2, 1.1.1.3, 1.1.1.4, 1.1.1.5, 1.1.1.6, 1.1.1.7, 1.3, 2.1, 2.2, 3.1
Plan 3: Do 1, 1.1.1, 1.1.1.1, 1.1.1.2, 1.1.1.3, 1.1.1.4, 1.1.1.5, 1.1.1.6, 1.1.1.7, 1.3

Here, Plan 1 entails opening the search frame, adding a shark with all relevant information, adding more sharks as needed, before returning to the search frame. A shark search is then initiated, and the resulting data is manipulated as the user wishes.

Plan 2 involves adding a new shark to the data stream as before, but only a single shark is added. A search then takes place, and the consequent data is viewable.

Plan 3 is essentially adding a new shark, then returning to the search frame, with no further action taking place.

I constructed these plans after calculating potential courses of action the user may take, and then doing the required hierarchical task analysis. The 3 plans listed are not exhaustive, but are probably the 3 most common sequence of actions that the user will do. The analysis here is relevant to the requirements of the application, and fulfils the need set forth by the extension of the program.

A potential drawback of this technique is that all plans cannot be accounted for. A user may find an order of doing things which are not taken into consideration, which may lead to oversights in design.

Virtual Window Models:
These models will allow a visual representation of which components come together to form different screens. Each different screen/frame will ideally deal with a set of similar tasks, and provide a look at how the user will interact with the program application itself.

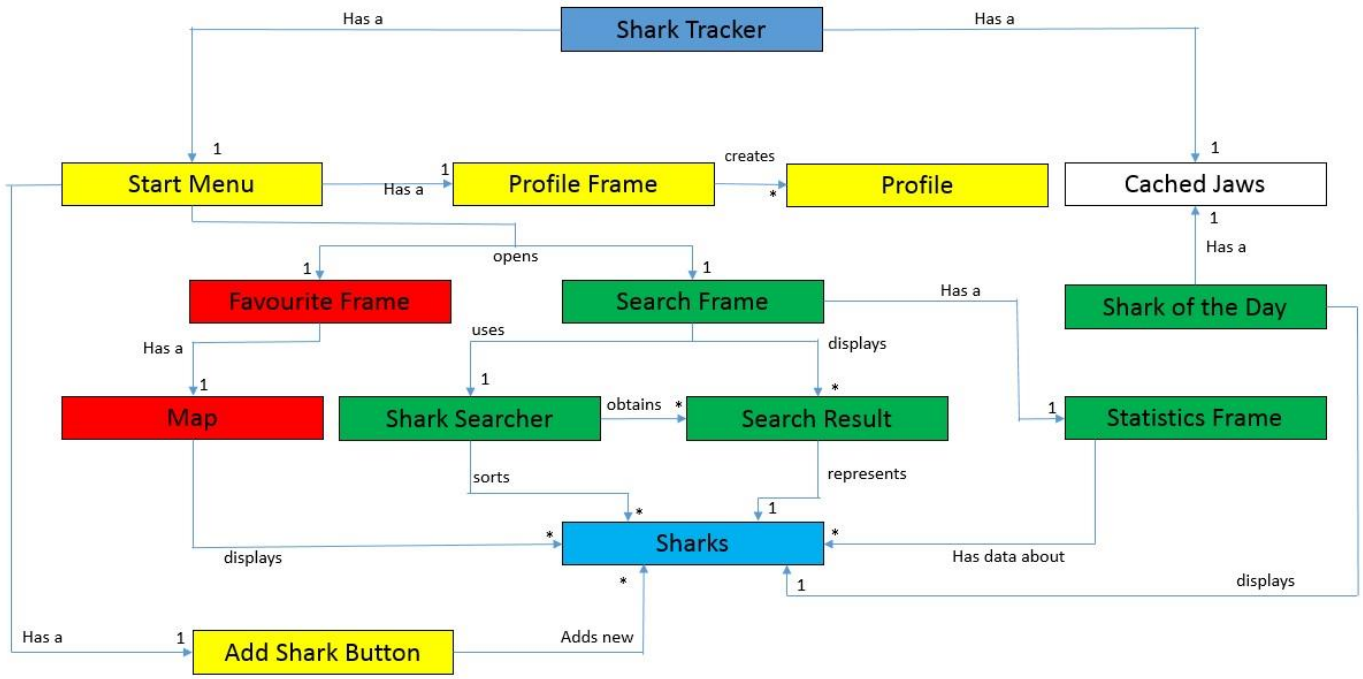

Figure 3: The Virtual window models of the Shark Tracker application

Key:

- Yellow entities refer to those that are present on the Start Menu, the first window seen when the application is launched.
- Green entities are those found on/in the Search Frame, or accessed from it.
- Red entities are to do with the Favourites frame, which is accessible from the Start menu.
- The Shark entity is used by all of the previous windows, and is a program-wide item.

This model was generated based on the interaction of the entities within the domain, and builds upon the domain model. Using a basic execution of the application as an example, a typical series of events would be the following:

Upon starting the program, the user would be greeted with the start menu window (yellow), and given the option to either search for sharks, review favourites and to add new sharks.
After clicking on Add shark, the user will be brought to a new window, where they will be able to input all the information required to generate a new shark object, and then given the option to return to the Start menu. This new frame would be simple to implement, consisting of a few JLabel, JButton and JTextField components.

If the user decides to then perform a search, they can click on the Search button, which will open the Search frame. Many other features of the application can be accessed from within this window, and it serves as a hub, of sorts, for the entire program. All actions and data relating to the searching of a shark may be found on this menu.

The Favourites frame contains a list of all favourite sharks, as well as their distance from the user.
A map is also provided to show the geographical location of each shark. If the user wishes, they may click on a shark in the list, which will then redirect them back to the Search frame, and all relevant information about that shark will then be displayed. All aspects regarding favourite sharks are found in this section of the virtual windows.

A drawback of this analytical technique is the fact it doesn't produce any executable code, it's just a model. This requires time, effort and money – all of which might hinder the development process.

Basically, every component is grouped with others that exhibit similar behaviour.
This entire process is user friendly, and the extra windows required by the extension fit seamlessly into the pre-existing program. This fulfils the requirements set forth in the domain analysis.

Global Navigation Structure:
This section will show how the entire program interacts, describing each section and how they are interconnected. This will illustrate the process of starting the application, and how the user navigates from one part of the program to another.

This process is heavily based on the virtual model windows, and the behaviour shown therein has been expanded upon to deliver the analysis of the global navigation structure.

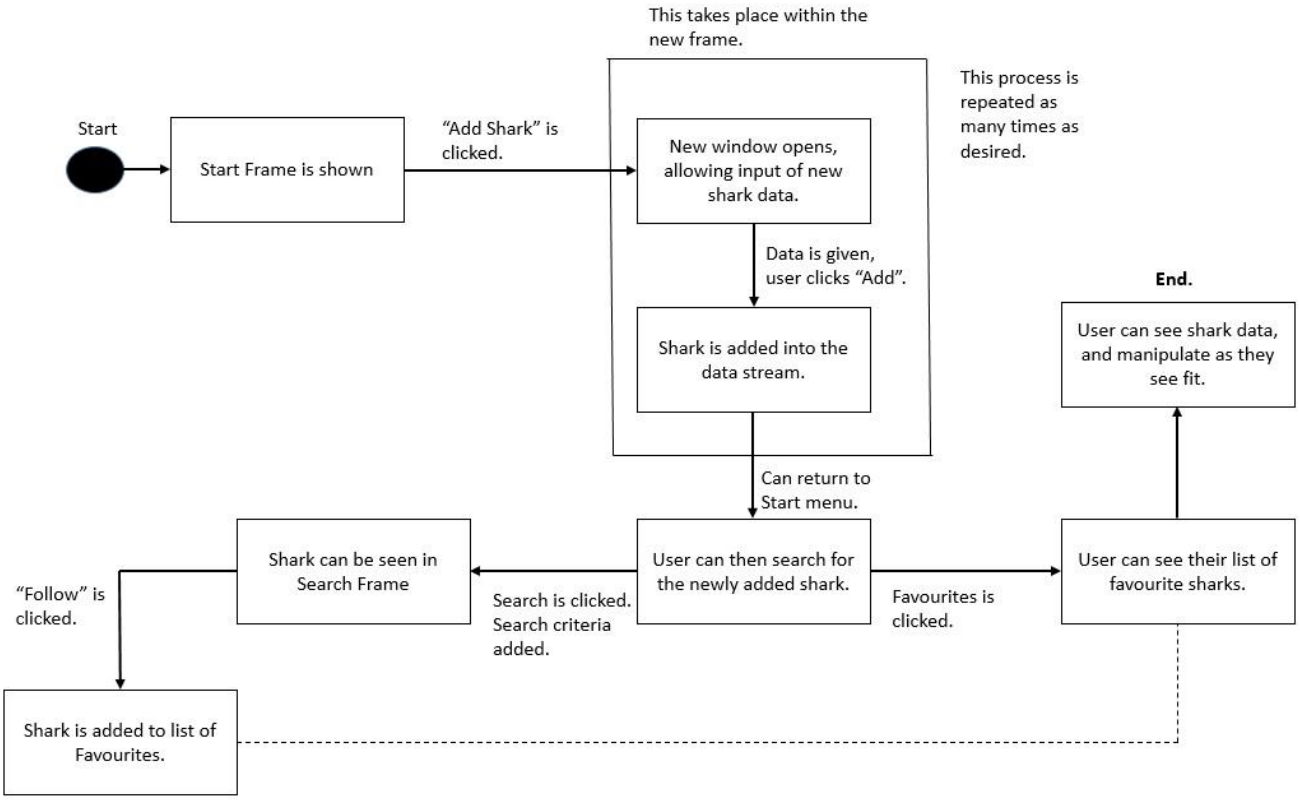

Figure 4: The Global navigation structure of the extension to the Shark Tracker application.

This structure has been realised after expanding on the analysis of the virtual model windows. There is a clear flow of action, starting from the execution of the program, the adding of sharks, searching & adding the sharks to favourites, and then terminating the application.

Each major change in state of the program is shown in a rectangle, and the action which causes this change is shown in text, accompanied by an arrow. The implementation of the planned extension is found in a new frame, for the majority. As shown, moving from one aspect to the program to another is fairly linear and easy to comprehend, something which is very desirable to the end user.

The implementation doesn't interfere with any pre-existing systems, and meets all requirements needed to complete an extension such as this.

For purposes of making the program safer and more clear, confirmation prompts will be provided on the "add shark" window. This gives another layer of authentication, and will help to prevent any undesirable inputs.

However, some user actions cannot be predicted, which may lead to the end condition never being achieved. This is a potential limitation for this technique, which cannot be helped.

To conclude, this global navigation structure provides a clear, succinct implementation of the extensions requirements, where each step is easy to achieve and program flow is obvious.

Summary and Conclusion:
This report aimed to provide a comprehensive, yet brief, overview of the process of implementing an extension to a pre-existing application, while referencing and taking into account the findings of four different analytical techniques.

I have recognised the importance of taking the needs of the user into account, and through different design approaches, construct a piece of software that is both easy to implement, navigate, and most importantly use. By taking the team to do the appropriate research, it can be made certain that the proposed extension is actually useful and perhaps allows even further expansion thereafter.

However, I have also realised that all of these techniques are not without their flaws and limitations. Each technique requires an additional development step, and does not produce any executable code. This effectively leads to in an increase in time, effort and money spent on the procedures.

With regards to actual implementation of the features discussed, it would be a mostly trivial affair. The bulk of the new code would be contained into a new class, with references to it being made in the Start menu. The resulting (newly created) shark object would then have to be stored too.
As previously mentioned, most of the new class would consist of a new frame, JLabels and JTextFields, as well as a few JButtons. A Layout Manager would ensure an aesthetic look to the new frame.

To finish, I have found the usefulness of these four techniques to aid greatly in the creation of a new facet of the application, despite their potential individual drawbacks. While making sure to remain within the confines of the guidelines laid out, each aspect of the extension came together without fail, and it is with these considerations in mind that I apply the processes to any design decisions I make in future.