Aarhus School of Engineering

Rolling Road GUI

Architecture

May 26, 2016

Contents

Introduction 1

This document is one of three, one for each of the following subjects: Requirements, architecture and implementation of the Rolling Road GUI. Within this document the overall architecture and software design considerations will be documented for the Rolling Road GUI Application. In addition to architecture, used technologies and protocol definitions will also be included.

Technologies 2

Each technology used to develop the Rolling Road GUI have been listed below, each was chosen based on the requirements, price and usability.

One of the main non-functional requirements was the use of Window 8 & 10 platforms, therefor an obvious choice was to use C# in corporation with WPF since it allows fast development.

2.1 .Net

.Net Version 4.6.1 was used since it was the newest available at the time of development and offers increased performance in WPF-applications.

2.2 Windows Presentation Foundation

WPF (Windows Presentation Foundation) Was used to develop the front-end, it's packaged with Visual Studio, making it easy to setup and developing in.

2.3 Dynamic Data Display

D3 (Dynamic Data Display) is a WPF-Compatible plotter/graph library, used for plotting the incoming data. It's a bit outdated, but it has features such as zooming and screenshots. Making it one of the better free plotting tools for WPF.

2.4 Prism

Prism is a collection of pattens such as the DelegateCommand often used in WPF applications using MVVM.

2.5 Entity framework

EF6 (Entity Framework 6) was to used to abstract away from the data source, enabling the use of a database later in the process if a change in requirements occur.

2.6 NUnit

NUnit 2.* was used as the unit-test framework.

2.7 NSubstitute

NSubstitute was used as a mock framework when unit-testing.

Design and Architecture 3

3.1 Backend

The backend needed to be able to handle multiple front-ends and a different data-storage option such as a database. These features needs to be available if for example User Story 8 was to be implemented, since the user story requires both a change in data-storage and user-interface.

An architecture matching these requirements are the Onion-Architecture. It was therefor used for overall architecture, this allows multiple front-ends while also abstacting away from the data, making it easy to switch to a database or other datasource.

The core namespaces can be seen in figure 3.1 on the following page, excluding testing and frontend namespaces.

(Core) Domain Model

The very center of the 'Onion', it should not reference anything other than itself. This is where all the models are defined, each model should also have an property called **Id** or ClassName**Id** so that the entity framework knows what to use as a key.

(Core) Domain Services

Is the 1st ring, it must only reference the Domain Model and nothing else. This is where all the interfaces to the Domain Model is, such as the Repository Interface.

(Core) Application Services

This is the outermost ring in the core, and is allowed to reference any core-libraries (Domain -Model and -Services).

(Infrastructure) DataAccess

Since it's outside the core, it's allowed to reference any libraries in the core. This is also where the ApplicationContext and Database Access can be defined.

Outer most ring

This is where the frontends and testing frameworks will be, they can use any internal or external library.

3.2 WPF-Application (Frontend)

The frontend is in the outermost layer of the Onion-Architecture, and in this case it's called the 'Presentation Layer'. For the front-end architecture/design MVVM(Model-

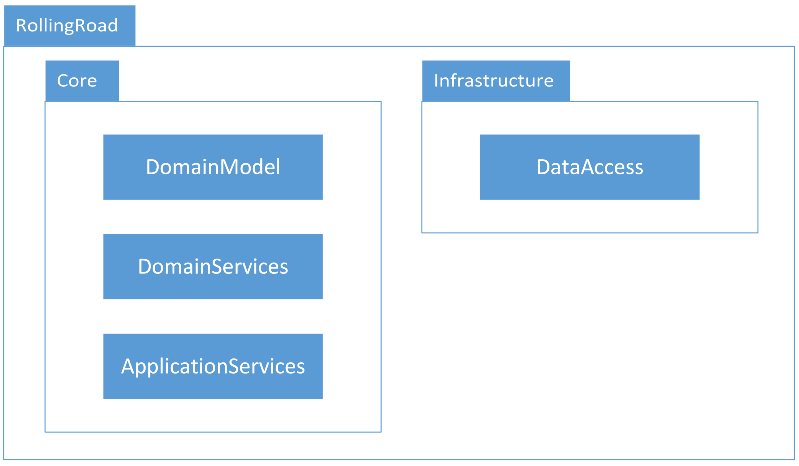

Figure 3.1. Main namespaces

View-ViewModel) was used, since it's what the WPF-library is build upon and have good support for. It also abstracts the view-code from the viewmodel, enabling testing of the viewmodel.

3.3 Testing layers

In the solution there are two testing projects, one for the core and one for the WPF-Application. It's separated in case we need to remove the WPF-Application at some point, making it easy to differentiate between what tests the core and what test the WPF-Application.

Protocols & Fileformats 4

4.1 DF4RR

DF4RR (Data File for RollingRoad) is a data storage format used to store collected data from RollingRoad, the file format is based on CSV[1][1]. But instead of ',' as a separator, ';' is used. The files encoding must be UTF8[2].

Note, the decimal mark used must be dots.

4.1.1 Header

First row, first column must be a cell[2] containing the case-insensitive string "shell eco marathon".

The following columns in the first row must be filled with the datatype-names saved in the file.

Second row, first column can contain a description about the datafile, but is not required. The following columns must be filled with the datatype-units for the datatype-name above, the number of names and units must match.

Examples of the first and seconds row can be seen section 4.1.3

4.1.2 Data

After the header, data will be on a new row for each new reading. Each row will start with an empty cell that can be used for anything. The following cells must contain data, and the number of cells filled with data must match the number of datatypes described in the header.

4.1.3 Example

```
SHELL ECO MARATHON;Time;Torque;Power
Test file 2;Seconds;Nm;Watt
;0.1;0;0
;0.2;0;20
;0.3;2;50
```

[1]Comma Separated Values

[2]A cell is the content between two separators

4.2 TC4MC

TC4MC (Torque Control File for MotorController) Is the data storage format used to store torque information about a given track. This enables a more accurate simulation on the rolling road.

The format is based on the DF4RR-Format with a few changes:

- Instead of "shell eco marathon" header in the first row, first coloum, it must be "eco shell marathon torque".
- The two data coloums "Torque" and "Distance" must be present.

Bibliography

[1] Csv unofficial standard. URL `https://tools.ietf.org/html/rfc4180`.

[2] w3schools.com.