Emergent Architecture Design

Paul Bakker, pbakker3, 4326091
Robin Borst, robinborst, 4291972
Matthijs de Groot, mawdegroot, 4171683
Julian Hols, jthols, 4247930
Jan van Rosmalen, javanrosmalen, 4318609

May 15, 2015

Contents

1 Introduction

To implement our product we will have different design goals so we can satisfy the needs of the customer. These are described here below. In section 2 we will describe the software architecture views of our product.

1.1 Design Goals

1.1.1 Usability

The most important design goal that we have is usability. It is important that the user can execute the program really efficiently and in a rapid order. That is the reason that we want to make the program executable via the command line. This allows the user to script multiple calls to the program to get the data from multiple patients. Because the customer has indicated that he knows how the command line call works, we think this is the best implementation to execute the program. This enhances the usability for the analyst as he can process a lot of data really quickly.

1.1.2 Modularity

The modularity design goal means that we will define the individual components well, which leads to better maintenance. Also because we work with five people it is beneficial to have good modularity as we can each work on an individual component. And good modularity also helps us greatly in realizing our next design goal.

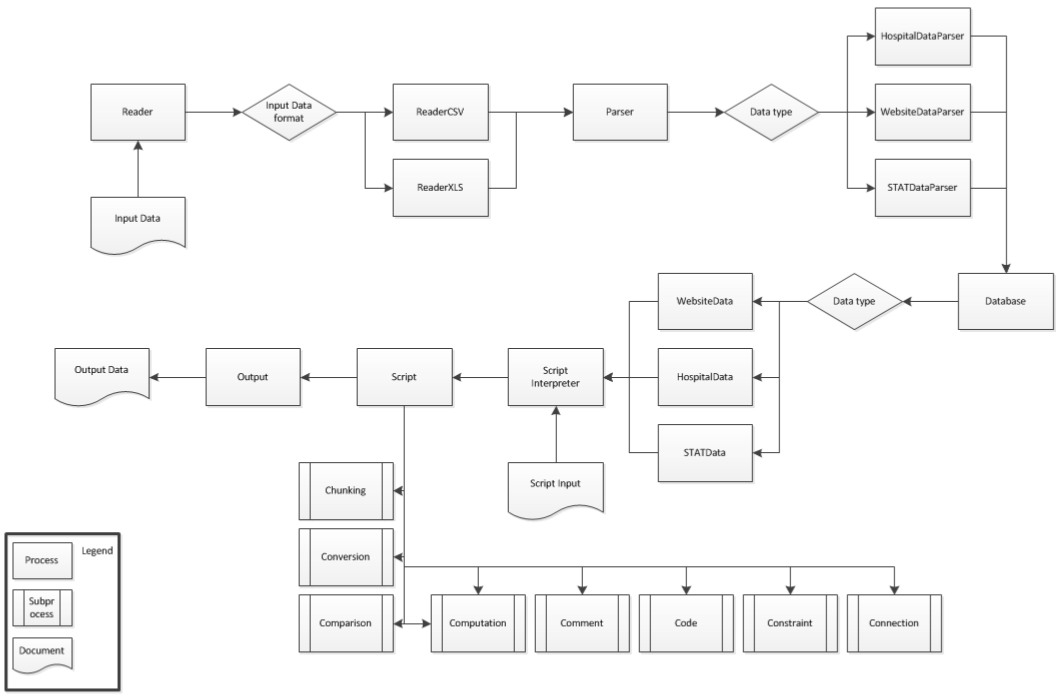
1.1.3 Re-usability

The re-usability design goal means that we will be able to reuse almost all of the code for other customers. We could for example reuse most of the code to easily get the same functionality for other self-monitoring projects.

1.1.4 Compatibility

The compatibility design goal means that the software is able to operate with different file formats outputted by other software products. We will use this to accept multiple file formats for the import and export. The exported files then can be used by different statistics analysis programs.

2 Software Architecture Views



2.1 Subsystem decomposition

We will divide our product into different components, which we will describe below.

- The Main class:
  The Main class contains the main method which is the first method that will be called when the user runs the program. Main will parse the command line arguments. The user can pass arguments to the program using a command line or a script. Main will also start the graphical user interface (gui) and receive the data the user inputs in the gui. After the user has input all the required data, the main method will execute the script file supplied by the user.

- mainUI class:
  The mainUI class contains all the code for the graphical user interface (gui). It allows the user to specify the input files in a more user friendly way than the command line. The gui asks the user to specify the script file, an output directory and a list of data files and the corresponding settings file for the data file. The Main class will initialize the mainUI and will also read the input data when the user is finished.

- DataFile class:
  The DataFile class provides an easy way of storing information about the data file and it's corresponding settings file. The dataFile will create

the correct Reader and Parser for the file and will read the settings file using XMLReader.

- Settings class:
  The Settings class is created by the XMLReader. It contains information on how to parse the file and specifies the column names and data types for each column. It is mainly used by the Parser class.

- XMLReader class:
  The XMLReader will be used by DataFile class to read a settings file in xml format. It outputs a Settings object.

- Reader abstract class:
  The reader will read the data file and provides an easy way for the Parser to read the data file. The reader allows us to add support for different file formats such as csv and xls without having to write a different Parser for each file format.

- Parser class:
  The parser will read data row by row from the Reader interface. For each file it parses it receives a Settings object containing information on how to parse this file. The settings object specifies the column names and the Value type of the data the column holds. The parser will create the correct Value object and add it to a Record. The parse class will output all Records after parsing into a Table data structure.

- Value abstract class:
  The Value interface allows us to easily add data types such as NumberValue, DateValue and StringValue. The Parser will create the correct Value object that is specified for each column in the settings file.

- Table class:
  The Table class holds a table of Record objects. A record object stores a row of data in a HashMap. This way data and columns are easily added or removed from the record. The table class is created by the Parser. It is used as an input in data processing commands such as filtering and chuncking. The data processes will output an updated table.

- ScriptInterpreter class:
  The ScriptInterpreter class will read the script file and execute all the corresponding sequential data operations. It will output a data-structure predefined by us.

- Exporter class:
  The Exporter class will generate a text/csv file from a Table object. The user will be able to select the delimiter to use to separate items.