TI2806 Contextproject

Emergent Architecture
Group HI4

May 22 2015

1 Introduction

This document describes how the architecture of the system looks like that is being built during the context project health informatics group 4 in study year 2015. An high level overview of the designed system is given and explained in chapter 2. Chapter 3 contains a glossary. This first chapter is meant to state the design goals of this project.

1a Design Goals

When designing the product we take the following design goals in consideration:

Generic
We try to make the program as generic as possible, so that the program can be used for different forms of sequential data analysis in different disciplines, not only for the research that must be done for ADMIRE.

Modular
Our goal is to split the program into different modules. The different modules must be as loosely connected as possible so that for example changes within modules would not affect the GUI and vica versa.

Quality of product
We aim for the highest quality of the product. To achieve such a high quality we are going to build a good architecture for our program so that the code is easily maintained. We write automatic test cases, so that if we make an enhancement to our program it automatically checks that we have not broken another part of the code. We aim for a minimum line coverage of 75%. We have coupled a continuous integration server to our VCS system in order to make sure that the code on the master branch always compiles and passes all the tests.

User friendliness
Another goal is to create an easy to use interface for the user, which is partly dependent on the overall quality of the program. We think that the modularity helps to guide the user in a work flow that works well for sequential data analysis.

Performance
The data analysis should be performed within a reasonable period of time with a limit of one minute. When the results are calculated, the output should be shown directly on screen to ensure that the users don't experience long loading times. This means that the program needs to have a responsive design. Also, a form of error management including error prevention and error correction (with error messages) will be implemented in all layers of the architecture.

Scalability and flexibility
The program will be able to process large sets of data with sizes in the gigabyte range. It can handle different types of files and different delimiters.

Use of Design Patterns
We want to implement our application following the Model-View-Controller (MVC) pattern to separate the backend logic from the frontend that represents the program. This enables us to divide domain objects from the GUI elements to keep the code cleaner and the system more maintainable.

2 Software Architecture Views

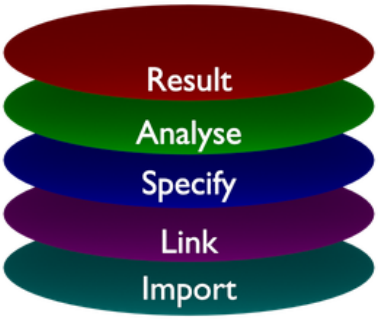
This chapter describes what our software architecture looks like. First the subsystems are identified and explained. Then the software to hardware mapping is explained, followed by how we store persistent data. Lastly concurrency between information is explained.

EAD: It is unclear how the components in your architecture design match your code. Where is the MVC model? How do the classes in the ”model” package fit into your architecture?

2a Subsystem Decomposition

The GUI is split up in five different tabs: Import, Link, Specify, Analyse and Result. These tabs represent the different subsystems of the software.

The main goal of this layered structure is to use the data of the inner layer as the input for the next layer. This way it is easy to separate all the modules and make it modular. All subsystems perform a specific task to evaluate the data one step further.



The innermost layer is the import layer. In this layer the data files are specified and read. Using this module the user can specify how the files should be parsed and stored in the program. Then the specified input is checked whether it is valid and the files are read and stored in a simple data structure.

The data that is read from the files is given to the next layer, the link layer. In this layer different files are linked together to form one sequential data structure. This sequential data structure can then be used to perform the data analysis on. The operations that need to be done on the data is specified in the next layer.

In the specify layer the user specifies what operations need to be done on the data. This is done using a scripting language that is developed during the project. This layer is able to parse the language into instructions for the next layer. These instructions are entered as in a pipeline: The output of one instruction is the input for the next instruction.

The next layer is the most crucial layer. In this layer, the analyse layer, the instructions are actually executed. This is done in pipeline like fashion. Each instruction, which is parsed in the previous layer, operates on the data sequentially. After the last instruction is executed, the resulting data is send to the final layer - the result layer.

This last layer is used to visualise and store the resulting data. Using this last layer it is possible to specify how to store files and where to store them. When this layer has finished executing, all transformed data will be available on the location specified by the user.

2b Relation between the architecture and design goals

The reason why we have chosen for a layered architecture is to make it easy to divide the code into modules and make it modular. The modularity will also help us to make the program user friendly. It will make us able to guide the user through the five steps of data analysis.

The innermost layer is the layer which is assigned to handle the scalability and flexibility of the program. It should be constructed in such a way that it is able to read in large data files of different kinds.
The link layer makes sure that the data can be analysed in a sequential order. This will improve the performance of the program since a sequential datastructure can be sorted more quickly. This is very important when the data has to be accessed randomly, for example when constraining the data on a particular value.
The specify layer needs to be able to parse general transformations that can be performed on sequential data. This will make the program generic enough to be used by researchers from all disciplines.
The analyse layer, which executes the transformations using pipelining, will play an important role in the performance of the program. By executing them sequentially, this will result in a fast execution of the analysis and generation of output.

The result layer is the layer which delivers the resulting data and information to the user interface for

display. This layer is responsible for the responsiveness of the program and thereby the user friendliness of the interface. Another task of this layer is to be able to save the results in any format and with any delimiter, specified by the user. This ensures that the program is generic with output.

2c Hardware/software Mapping

This subchapter describes how the software for this project is mapped on hardware. In this project a program for a single computer is designed. The program also runs as one process. However, the process can consist of multiple threads.

2d Persistent data management

For this application it is not required to store information persistently. The output of the program is stored as a text file. The format of the output is so that it can be entered into a statics tool of choosing.

Another idea is to store the state of the processing, for when the processing was not finished. If the state of the program can be stored in XML for example, the user can pick up their work where they left off.

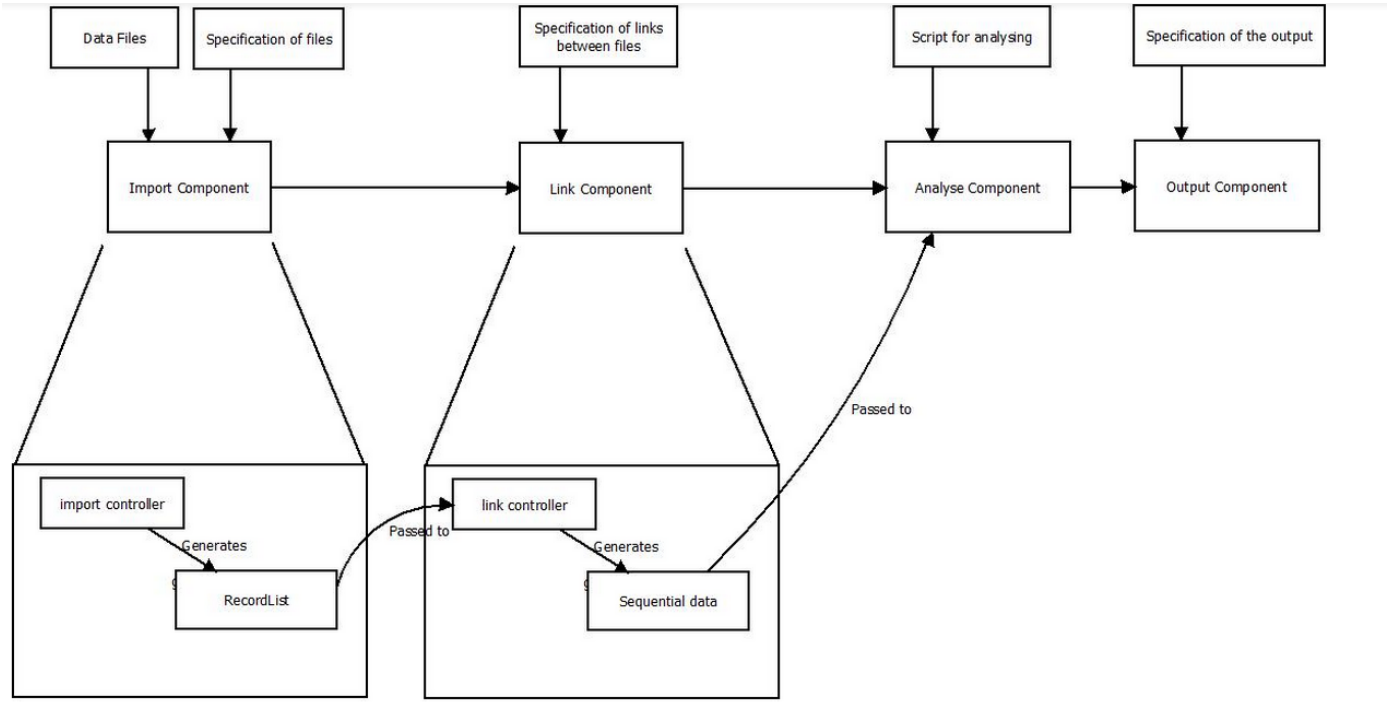
Lastly it would be useful to store the scripts written in the scripting language when a script is needed more than once. This can also be done in textfiles or a custom file format for this language.

2e Concurrency

This subchapter is meant to describe how concurrency issues are solved. Because it is not possible that the program is used by multiple people or processes at the same time, there are no concurrency issues.

2f Diagram of the architecture

Here is a diagram of our layered architecture and the connection between the front-end and the back-end:



As you can notice, the components present the layers described in the Subsystem Decomposition section of the first chapter. This diagram also shows the connection between the frond-end and back-end of the system. The import component collaborates with an import controller which controls the view of the import tab in the GUI. This is where the user specifies which data files he or she wants to import and where a RecordList is generated of the data. After this the list of records is passed to the link controller which is connected to the link component. The link controller controls the view of the link tab and transforms the data into sequential data. The RecordList gets transformed into a TreeSet to improve the performance of the data analysis.

The sequential data object will be passed to the analysis component which holds the specify and analyse layers. It will will take the scripting language as input and parse it into transformations for the analyse layer. This will enable the system to perform the transformations in a pipelining sense.

The results will be passed onto the output component which is responsible for the display of the output on screen. This is also where the user specifies the format in which the output file needs to be saved.

Major external technologies

Model View Controller

We decided to implement our project following the Model View Controller (MVC) pattern. The most important advantage of MVC is that it separates logic from your views. It's really helpful to use an architecture that utilizes a controller if there is logic required that doesn't necessarily fit into a model. In our case we are using an import controller which handles the data types of the imported data, which doesn't necessarily fit into one of the used models. The advantages below convinced us to use MVC:

- First of all, MVC makes the code more clean and maintainable. It enables us to keep a good overview of the code.
- Because of the separation of concerns, the model and controller code could be reintegrated in other systems such as a web app, a desk app, a service without much effort.
- MVC enables us as a team to work in parallel. As an individual programmer you would probably have a different approach for the implementation but when working in a team, you will first need to discuss and agree on the structure of the code. With MVC the responsibilities of the developers can be easily divided and assigned.

From a Model View Controller perspective, the FXML file that contains the description of the user interface is the view. The controller exists of Java classes, which are declared as the controller for the FXML file. They contain the application logic of the system. The model consists of Java objects that you connect to the view through the controller.

Alternatives to MVC

Other architectural patterns that could be used in our situation are for example Presentation-abstraction-control, Model View Presenter, and Model View ViewModel. These patterns are interaction-oriented and similar to MVC.

An important difference with Presentation-abstraction-control (PAC) is the abstraction component. PAC retrieves and processes the data with the Abstraction component and makes a visual presentation of the data (a template actually) with the Presentation component. The Presentation and Abstraction components never speak to each other. This communication and control flow between these components are all handled by the Control component. This is also the reason why the PAC doesn't suit our system. PAC is only useful when you aren't calling your data store directly from your display layer, which is actually what we want to do in our system. The user needs to be able to import files in the user interface.

Model View Presenter (MVP) and Model View ViewModel (MVVM) are derivations of the MVC pattern. In MVP the controller has been replaced by a Presentation component to which all presentation logic is pushed. In MVVM this is pushed to the ViewModel. These components are responsible for exposing methods and handling all UI events by receiving input from users via the View, then process the user's data with the help of Model and passing the results back to the View. Unlike View and Controller, View and Presenter or ViewModel are completely decoupled from each other and their communication is handled through the interface. These patterns have a clean separation of the View and Model and the amount of data is reduced because of the passive View but this means that there is less encapsulation

and more work to do as the developer has to do all the data binding himself. We preferred MVC above these patterns because we wanted our View to process the input partially before passing it to the next layer. This is needed for the script editor.

FXML

We use FXML to provide the structure for the user interface separate from the application logic of our code. This enables us to build an interface that uses Java components without the need to worry about fetching and filling in the data. In comparison with alternatives to FXML, the scene graph in FXML is more transparent. This enables us to build and maintain a testable interface. It is also a compiled language so you do not need to recompile the code every time you want to see the changes. The content of the files will be localized as they are read so they will be automatically updated. This means that you don't have to manually update every element of your interface. Also, it's suitable for our project since FXML works with any Java Virtual Machine.

3 Glossary

Layered structure - Also called a Multilayered Architecture; a software architecture that uses layers for allocating the different responsibilities of the program.

Master branch - The version of our software application that is always ready for deployment.