Emergent Architecture Design

Group Health informatics-2

Rick Proost – 4173619
Pascal Remeijsen – 4286243
Wim Spaargaren – 4178068
Arnout vd Knaap – 4223969
Jelmer de Boer - 4223152

Work in progress

Contents

1. Introduction

This document provides sketches and ideas of the design of the system that is going to be built during the Context Project Health Informatics. The architecture of the system is explained in the form of high level components of the system.

1.1 Design goals

The following design goals will be maintained throughout the project:

- Availability
  By Using the Scrum method to built the system, we make sure that each week a working product is ready for the client. This way the client will have the ability to see and use the system each week. This way, if there's a feature the client doesn't like or wants differently, the changes can be made as soon as possible. The product will also be developed as a standalone application this way it isn't dependent of external factors and can always be used. This is needed so researchers can analyse their data at any time.

- Manageability
  The system will be provided with a command line which researchers can enter SQL-queries to retrieve data in additional ways that are not yet implemented. For larger modifications and additions to the system that require the code to be extended or modified, all code will be well documented and commented. This enables developers who were not involved with the initial development to work on the system.

- Performance
  Since the system is a standalone application with a local database it will have a high performance ratio. This means that the connection between the database and the interfaces should be fast enough to ensure that the user don't experience long loading times or loading times at all when displaying the results of the data files.

- Reliability
  The system should not crash more than once every hundred uses. We will strive to make sure the system will crash as little as possible, but reliability is not something that is absolutely required for this system.

- Scalability
  Since the program interprets data dynamically the product can be distributed to analyse different sorts of data files. So it could also be used to analyse data from for example diabetes patients.

- Securability
  The system is a standalone application which doesn't connect to the internet. This means that all patient data records will only be kept at the user's computer.

2.Software architecture views

This chapter describes the architecture of the system. The architecture is split into some subsystems. These subsystems will work together to create the research tool that we need to deliver.

2.1 Subsystem decomposition (sub-systems and dependencies between them)

- Parsers
  Our program contains a collection of Parsers that are designed to dynamically read data from xml, excel and txt files. This data also contains the necessary information to put this in a database.

- Database
  The database receives the data from the current files, parsed in the Parsers. This is needed for the queries that will be run over the database.

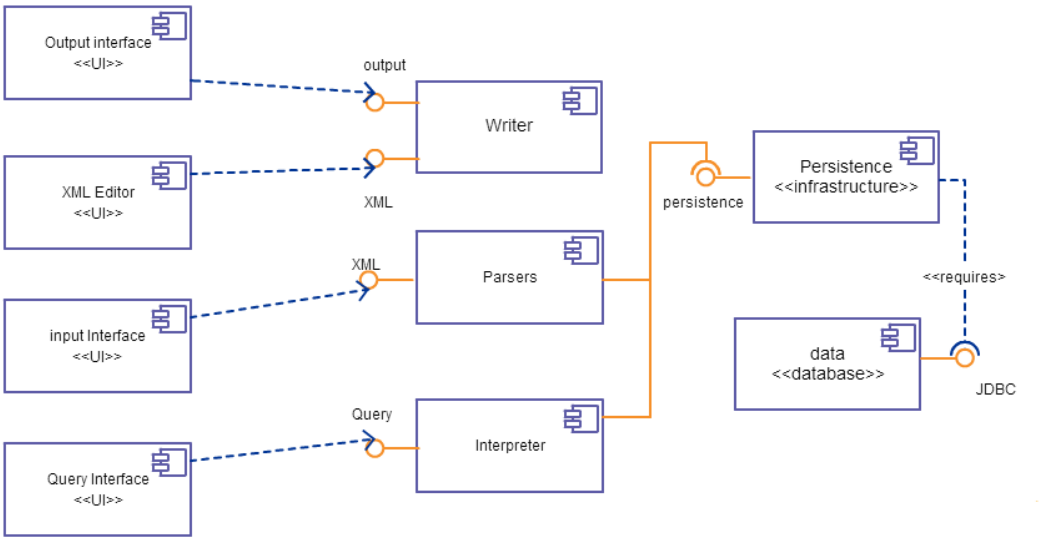
- Interpreter
  The interpreter parses the queries the user puts in the GUI. These queries will be parsed and given to the right methods in all the 8 analysis classes.

- GUI
  The GUI will handle the input from the user. This can be used to set all the wanted settings for the current run and ask queries. Settings like input files, queries, outputfilename etc.

- XML editor
  The XML editor is an interface where an xml file can be specified and created. The xml file is parsed after that by the parser component.

Below a component diagram, flow chart and database structure.

Component diagram:



Step-by-step:

We have four different kinds of interfaces for users, this is to keep the product user-friendly. The output and xml editor use the writer component to write data to files. In the XML editor you can specify an xml file according to the data the user wants to analyze.

The output interface gives options on how to write the analyzed data to a file, for example what kind of delimiters the user wants.

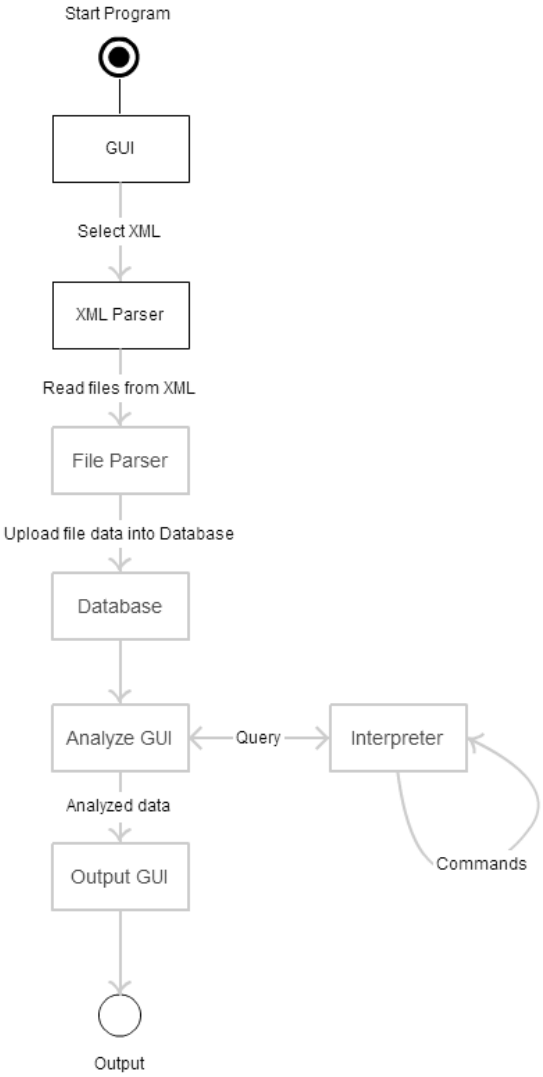
The input interface offers options to select an xml or open the xml editor, from there the product parses the selected data.

Parsers in the parser component will save all data parsed in the persistent data structure.

The query interface works with the interpreter component, in the query interface you can specify queries to analyze the data. After the query is being sent to the interpreter, the interpreter component will select what method to run, what will analyze the data in the persistent infrastructure.

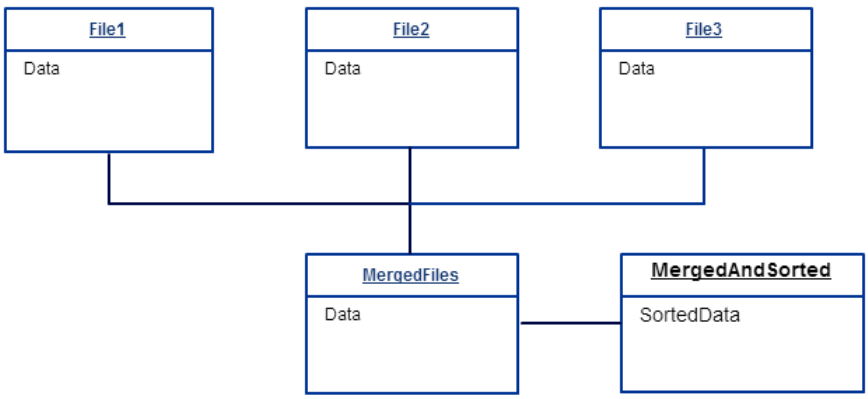
This analyzed data will be available in the output interface, which will sent it with the offered options to the writer.

Flow-chart for standard program run:



.

Database structure:



Database:
The database structure consists of tables for every file, and a table for the files merged together and a view where they are sorted on date.
It first creates tables for every file, this because we want to keep the column types of every table as generic as possible. In the xml editor we have a requirement for every file that needs to be read to have atleast one column with a date. Then in each file to be read there is one date type has to be checked to be merged on. This is to keep the data sorted on the desired date from a file.
After this it is easy to merge all files and sort them, constraints can be executed easily with SQL queries. This is basically the reason we chose for a database, there is already a language that performs quick analysis over data. We can use this to develop features the user needs.

Chunking:
We create a chunk of every line in the sorted view in the database, so we create a sorted list of java objects in our program. It's easier to add codes/comment/connections in a java object then in the database itself. Every sequential data analysis method can be performed over chunks, so creating a chunk of every line is a good way to store the data.

Chunk:
Line: Id pointer to data.
Code: String for a code of the chunk.
Comment:String for a comment of the chunk.
Pointer: HashMap of chunks this chunk points to.

This is everything you need for the sequential data analysis for now
<<More to be added>>

We thought of storing code/comment/pointers

Interface:

2.2 Hardware/software mapping (mapping of sub-systems to processes and computers, communication between computers)

The software will be only for analysts. There is a single application, which will create a temporary database. There will be no communication between computers or other hardware.
We use an overarching design for our data management. The user delivers us the tables with the data they want to process. We take those tables and store them in different tables in the database. This makes it easy for the user to reference specific data.

2.3 Persistent data management (file/ database, database design)

There is not a lot of persistent data used in this program. The only persistent data is de xml file with the settings, that the user can input.

2.4 Concurrency (processes, shared resources, communication between processes, deadlocks prevention)

Our system doesn't make use of shared resources or communication between processes, because it's a standalone application. Since we have a local database we have no real deadlock treats. But to make sure nothing will go wrong we make sure the system can only run one query at a time.

3. Glossary

Scrum - An agile software development framework which has a working product at the end of every week.

Standalone application - An application that does not load any external module, library or program.

GUI - Graphical User Interface

XML - Extensible Markup Language, XML) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable.

SQL -Structured Query Language. A language designed for managing data held in a relational database management system. (Database).