Group project in TDT4215
Web-intelligence

Alessio DE ANGELIS Martin HENGSTBERGER Simon STASTNY

April 20, 2013

Contents

1 Introduction

This report was created during the Web-Intelligence (TDT4215) course in the spring semester of 2013. It contains information about the mandatory group project. The task was to, automatically suggest relevant treatments and drugs by using doctor's patient notes and an electronic handbook for pharmaceutical interventions. A java application was created using several tools as well as a lot of creativity. You can find a description of the application's architecture, the processing, the limits and the results of the project. Finally we analyze the results and collect some ideas how to improve the application.

2 System architecture

Given the fact that our main architectural drivers were short time to market and developer inexperience, we decided to use just basic facilities of chosen frameworks and tools. Spending too much time learning details of tools such as *Jena* and *Lucene* would leave with a little time to make the actual implementation.

Since the nature of most of the tasks in this software system is highly time consuming, we decided to take advantage of provided multi-threading facilities that Java offers and we implemented the software using *Java Executor Framework*. This choice made the code a bit more complicated than a code of regular single-threaded application, but we managed to reduce the execution time significantly on multi-processor machine.[1]

As stated above, we are using some tools such as *Jena* and *Lucene*. Their particular use in the application is described below in corresponding chapters. Apart from those tools, we are for example using *SAX* and *Apache Tika* for parsing tasks.[2]

Since the team work went in parallel, we decided to make unit test to test each of the unit developed. This made the testing of certain tasks possible at the time even if their testing in the main application would not be possible because of some preceding tasks missing. We used JUnit library to run the tests.[3]

2.1 Application flow

This part presents textual description of application flow. Please refer to the figure 1 for sequence diagram representation.

In the beginning, the ICD and ATC files are parsed by `ATCParser` and `ATCParser` respectively, new tasks (`ATCIndexer` and `ICDIndexer`) for indexing them are created and submitted to the executor.

Directories `/data/book/` and `data/patient_data/` are then scanned for files. For each patient case file a new `PatientCaseParser` task is submitted to the executor. Same thing happens for book files, the only difference being the parser implementation, which is `BookParser` in this case. These return collections of `MedDocument` instances.

After both indexes (ICD and ATC) are created, the application starts to take finished `MedDocument` instances from completion service and creates `CodeAssigner` tasks for them and submits them to execution.

In the `CodeAssigner` service, each sentence in the `MedDocument` is looked up in both indexes and found hits (ATC/ICD codes) are added to the document. This happens both for patient case documents and book chapters.

[1] Execution time on a machine with Intel Core i3 (M350 2.27GHz) was reduced by a factor of cca 3.
[2] http://sax.sourceforge.net/ and http://tika.apache.org/
[3] http://junit.org/

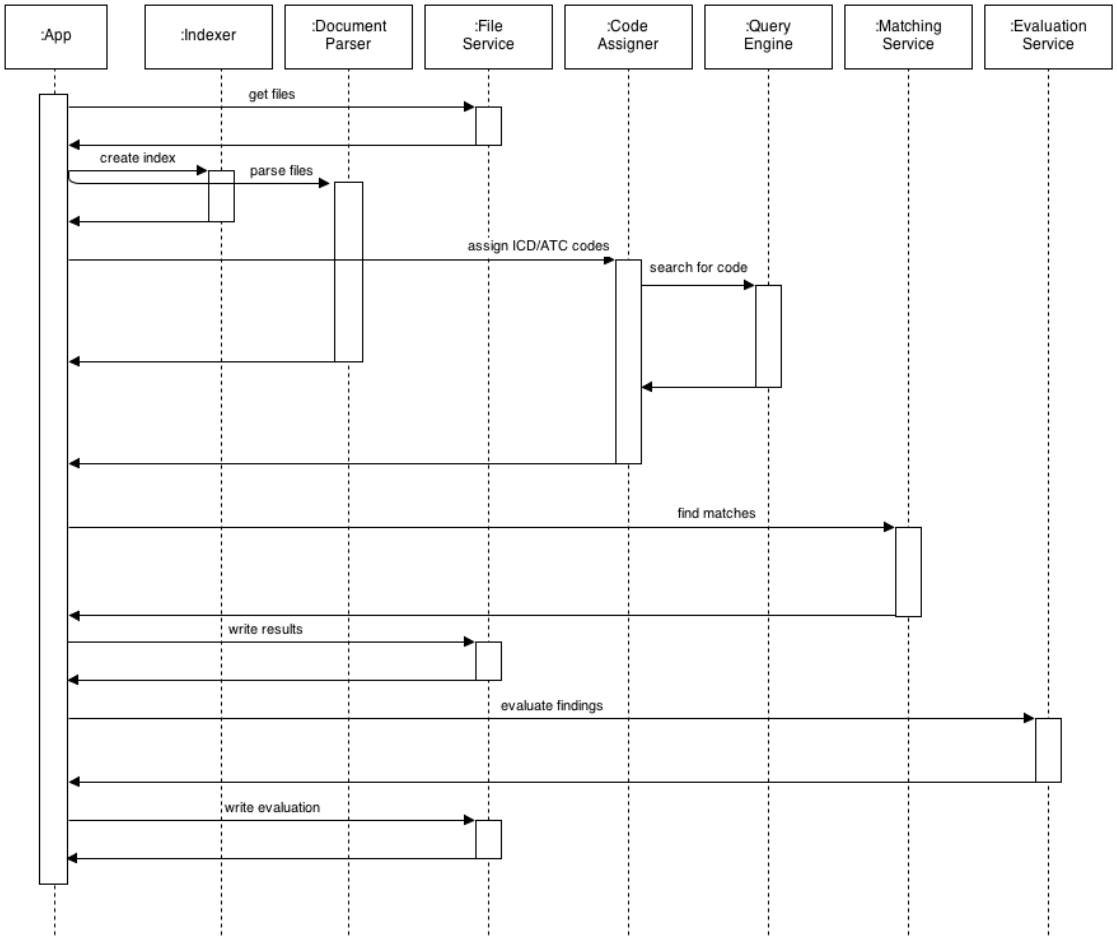

Figure 1: Sequence diagram

After all tasks from the executor are finished (i.e. after all books and patient files are parsed and all `MedDocument` instances are assigned codes), the application proceeds to matching.

An instance of `MatchingService` is made and is used to find matches between patient case documents and book chapters. This is done based on common ICD/ATC codes found in both documents. If two documents (one of each kind) have one or more codes in common, the book chapter is marked as relevant to the patient case and its ID is added to the `relevantIds` set in `MedDocument`.

After that, patient case documents are written to files in `data/output/` directory.

In the end, an `EvaluationService` is instantiated and used to evaluate the output with the gold standard. It checks all patient case documents against their obverse from the gold standard and checks if the correct chapters were found, how many wrong chapters were found and how many are missing. It writes the evaluation result into `data/output/evaluation.xml` file.

3 Components function and role

Many tasks occurring in the application share some similarities in behaviour and structure and thus we made the system very modular and easily modifiable to accommodate the small differences but large similarities. The sequence diagram shown in 1 displays a naive view over the system. Please note, that `Indexer`, `Parser` and `QueryEngine` are general names for different implementations (for example ATC files are parsed and indexed differently than ICD files, however the structure of the software component is similar).

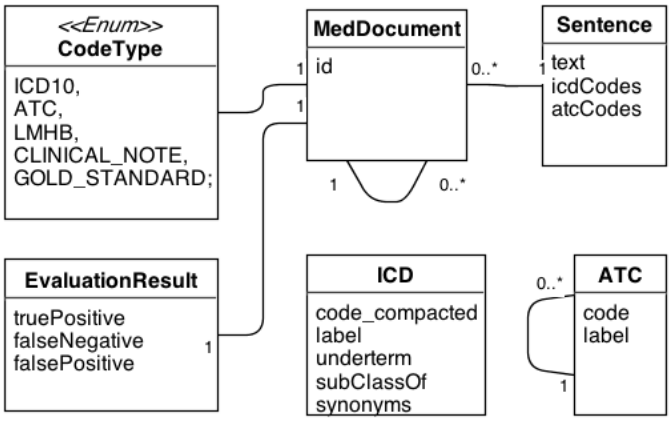

Figure 2: Class diagram

For reuse and modifiability, class `MedDocument` was created. This can hold an instance of patient case, a chapter from a book or a gold standard patient case. Since these files show similarities (have an id and contain sentences), the system treats them in the same way.

3.1 Parsing ICD and ATC files

There are several ways to parse ICD and ATC files to extract information from them: you can consider those as normal XML files, use regular expressions and so on.

We have chosen a semantic web tool, *Jena*.[4] As we can read from the website: *Jena is an API for reading, processing and writing RDF data in XML, N-triples and Turtle formats; an ontology API for handling OWL and RDFS ontologies; a rule-based inference engine for reasoning with RDF and OWL data sources; stores to allow large numbers of RDF triples to be efficiently stored on disk.*

This tool may seems to be a little bit verbose to use, but it lets us handle in a very efficient way both the ontologies, the ICD that is an OWL format and the ATC that is a turtle, without the risk of losing information from them. Moreover, we can have easily access the information about all the RFD graphs: we can keep track for example of the hierarchies, the subclasses, the synonyms and so on. All this extra information may be used to improve our query engine, doing some refinements when the pure textual search is not accurate enough.

When we parse ICDs we store them in Java beans the values of the following properties `code_compacted` (the code of the disease), `label` (description of the disease), `underterm` (extra information about the disease, not every instance has it), `synonyms` and `subClassOf`. When we parse ATCs we store them in Java beans the values of the code, the `label` and `subClassOf`.

[4] http://jena.apache.org/

3.2 Tools for indexing and querying

For indexing and querying we are using an information retrieval approach, through a bag of words model. For this task, we are using *Lucene*.[5] We read from the Website:*Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.* Lucene is scalable, powerful and accurate for using efficient search algorithms. At the beginning we were trying to create our own information retrieval engine, but we came across several problems. First of all how to deal with Norwegian language, our background is very poor since we are all international students in the group, how to stem correctly words or divide properly the *aggregates* (four, five words merged all together in one). *Lucene* is a library made by experts and their implementation of the `NorwegianAnalyzer` (with the Porter stemming algorithm), even if maybe not as accurate as the English one, has been revealed very useful for our purposes. Moreover it natively supports the vector space model for a more accurate retrieval than boolean.

We keep two different indexes, one for the ATC documents and the second for the ICD documents. Since the indexing is very fast, we keep both the indexes in ram, for performance reasons during the queries (you don't need to access the disk that is usually time consuming). The ATC documents are composed by two fields: one for the `atc_code`, that just indexed without any further string processing; the second field, `label`, contains the description of the code and is the one used for extracting the documents. This field is indexed, stored, tokenized and calculates the term vector matrix.

The ICD documents are indexed according to three fields. The first one is for the property `icd:code_compacted`, to identify uniquely the codes. The second one is for the `label:` the field has the same properties as the ATC label. The third field contains all the terms of the label plus the other information for doing query expansion, such as the synonyms and the property `icd:underterm`. Adding those extra words let have a more accurate result when we submit the query to our engine. In fact at the moment we query the ICD index looking into the terms contained in the extra field.

3.3 Parsing patient cases

Since the assignment description states that preprocessing of patient case input file is legal, we took the possibility and manually made an XML file representing the data contained in the `.doc` file provided. Parsing a `.doc` file would be a waste of time with no learning outcome. This as well gave us the opportunity to decide how the text is divided into sentences manually, and what characters to replace (we found that slashes cause problems in queries).

To parse this created XML document, we were using SAX parser with our own implementation of the `DefaultHandler` to produce `MedDocument` instances from the files read. This part of the work gets done in `PatientCaseParser`.

3.4 Parsing the gold standard file

To perform the evaluation easily, we decided to make an XML file for the gold standard file[6] and let the application read and compare the two result sets (the produced one and the one from gold standard) automatically. This file gets read by `GoldStandardParser` and data are held as instances of `MedDocument`.

[5] http://lucene.apache.org/core

[6] File is located at `/data/goldStandard.xml` in the project directory.

After the patient case files are read, assigned codes and found their relevant matches from the book, we compare the relevant matches with those from the gold standard. This gets done `EvaluationService` in the very end of the program run. Evaluation results are then stored in an XML file.[7]

3.5 Parsing the book

The headlines in the handbook identify the therapies and drug descriptions. Those are parsed into instances of `MedDocument` as well as the corresponding text body. The text is spitted into sentences for later processing. Headlines that do not start with a legal (i.e. *T3.1.1* or *L2*) therapy or drug identifier are skipped. The following text is skipped too, until a legal heading is found. There were challenges with invalid[8] HTM files. Furthermore different ways of encoding special characters were used. I.e. both `&aring;` (HTML entity) and `&#x00E5;` (Unicode) appear.

4 Limitations given in the assignment. Present theories used. Are any theories outside of the curriculum been used? Explain the reason for the selection

5 How have the theories been adapted to the assignment?

6 Summary of the results.

7 Evaluation and discussion of the results. Limitations of your evaluation results.

8 Thoughts of potential improvements.

9 Conclusions.

10 Future work

Our system is accurate but in the future some refinements could be done to improve our query engine that we didn't manage to implement due to time constraints. For example an evaluation function that gives more importance to the hierarchy of the classes, for example if a code is subClassOf another code (E10, subclass and E10-14, the superclass in ICD for example). Since we use *Jena* to read the OWL ontologies, could be done through a simple graph traversal using suitable methods offered by the APIs, or using ad hoc SPARQL queries.

We could even taken better into account the synonyms: if two codes have the same synonyms, the may refer to the same disease.

[7] File is located at `/data/output/evaluation.xml` in the project directory.
[8] http://validator.w3.org/#validate_by_upload