Middle East Technical University
Department of Computer Engineering

-TURKUAZ PROJECT-

TIRAN SOFTWARE
FINAL DESIGN REPORT

RadeX

Tahir Bilal 1394741
Onur Deniz 1391002
Soner Kara 1395110
Mert Karadağlı 1395128
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Of Contents</td>
</tr>
<tr>
<td>1. Introduction</td>
</tr>
<tr>
<td>1.1. Motivation</td>
</tr>
<tr>
<td>1.2. Project Topic</td>
</tr>
<tr>
<td>1.3. Project Description</td>
</tr>
<tr>
<td>2. Current Status</td>
</tr>
<tr>
<td>2.1. What We Have Done So Far</td>
</tr>
<tr>
<td>2.2. Research</td>
</tr>
<tr>
<td>2.2.1. Db4o</td>
</tr>
<tr>
<td>2.2.2. WordNet</td>
</tr>
<tr>
<td>2.2.3. Jwnl</td>
</tr>
<tr>
<td>2.2.4. Snowball</td>
</tr>
<tr>
<td>3. Design Constraints</td>
</tr>
<tr>
<td>3.1. Naming and Documentation Constraints</td>
</tr>
<tr>
<td>3.2. Time Constraints</td>
</tr>
<tr>
<td>3.3. User Interface Constraints</td>
</tr>
<tr>
<td>3.4. Performance Constraints</td>
</tr>
<tr>
<td>4. Data Design</td>
</tr>
<tr>
<td>4.1. Database Tables</td>
</tr>
<tr>
<td>4.1.1. Rapor</td>
</tr>
<tr>
<td>4.1.2. Problem</td>
</tr>
<tr>
<td>4.1.3. Normal_bulgu</td>
</tr>
<tr>
<td>4.1.4. Islem</td>
</tr>
<tr>
<td>4.1.5. Ilac_tedavi</td>
</tr>
<tr>
<td>4.1.6. Kyas</td>
</tr>
<tr>
<td>4.2. Er Diagram</td>
</tr>
<tr>
<td>5. Architectural Design</td>
</tr>
<tr>
<td>5.1. Structural Modeling</td>
</tr>
<tr>
<td>5.1.1. Data Flow Diagrams</td>
</tr>
</tbody>
</table>
5.1.1.  Level-0 .......................................................................................................................... 14
5.1.2.  Level-1 "RadeX" ........................................................................................................... 14
5.1.3.  Level-2 "preprocess reports" ....................................................................................... 15
5.1.4.  Level-2 "associate semantics" ....................................................................................... 15
5.1.5.  Level-2 "learn and extract" .......................................................................................... 16
5.1.6.  Level-3 "extract data" ................................................................................................... 16
5.2.  Moduler Hieararchy .......................................................................................................... 19
5.3.  Module Explanations ........................................................................................................ 20
5.3.1.  Report Analyzer Component ...................................................................................... 20
5.3.1.1. Preprocessor Module ................................................................................................. 20
5.3.1.2. Semantic Associator Module .................................................................................... 20
5.3.1.3. Learner ...................................................................................................................... 21
5.3.1.4. Finalizer ................................................................................................................... 21
5.3.1.5. Spell Checker ............................................................................................................ 21
5.3.2. Database Searcher Component ..................................................................................... 22
5.3.2.1. Database Connector ................................................................................................ 22
5.3.2.2. ResultSet Reader ...................................................................................................... 22
5.3.2.3. Displayer .................................................................................................................. 22
5.4.  Class Modeling .................................................................................................................. 22
5.4.1. Package/Class Descriptions & Class Diagrams ............................................................... 22
5.4.1.1. "anlam" package ....................................................................................................... 22
5.4.1.2. "arac" package .......................................................................................................... 23
5.4.1.3. "database" package .................................................................................................. 25
5.4.1.4. "lexicon" package .................................................................................................... 27
5.4.1.5. "malumat" package ................................................................................................. 28
5.4.1.6. "rules" package ........................................................................................................ 30
5.4.1.7. "radex" package ...................................................................................................... 31
5.4.1.8. "yapi" package ......................................................................................................... 32
5.4.1.9. "gui" package ........................................................................................................... 34
5.5.  Behavioral Modeling ......................................................................................................... 35
5.5.1. Sequence Diagrams ....................................................................................................... 35
5.5.1.1. "Analist" Sequence Diagram .................................................................................... 35
5.5.1.2. "Sorgucu" Sequence Diagram .................................................................................. 36
5.5.1.3. "Preprocessor" Sequence Diagram ........................................................................... 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5.1.4</td>
<td>“Anlamsalliskilendir” Sequence Diagram</td>
<td>38</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Activity Diagrams</td>
<td>39</td>
</tr>
<tr>
<td>6.</td>
<td>Interface Design</td>
<td>40</td>
</tr>
<tr>
<td>6.2</td>
<td>User Interface Design</td>
<td>40</td>
</tr>
<tr>
<td>6.3</td>
<td>Method/Class Interfaces</td>
<td>42</td>
</tr>
<tr>
<td>6.3.1</td>
<td>“arac” package</td>
<td>42</td>
</tr>
<tr>
<td>6.3.2</td>
<td>“rules” package</td>
<td>45</td>
</tr>
<tr>
<td>6.3.3</td>
<td>“malumat” package</td>
<td>46</td>
</tr>
<tr>
<td>6.3.4</td>
<td>“database” package</td>
<td>47</td>
</tr>
<tr>
<td>6.3.5</td>
<td>“lexicon” package</td>
<td>48</td>
</tr>
<tr>
<td>6.3.6</td>
<td>“radex” package</td>
<td>49</td>
</tr>
<tr>
<td>6.3.7</td>
<td>“yapi” package</td>
<td>49</td>
</tr>
<tr>
<td>6.3.8</td>
<td>“database” package</td>
<td>50</td>
</tr>
<tr>
<td>6.4</td>
<td>Method/Class Interfaces</td>
<td>51</td>
</tr>
<tr>
<td>6.5</td>
<td>User Interface Design</td>
<td>52</td>
</tr>
<tr>
<td>6.6</td>
<td>Method/Class Interfaces</td>
<td>53</td>
</tr>
<tr>
<td>6.7</td>
<td>User Interface Design</td>
<td>54</td>
</tr>
<tr>
<td>6.8</td>
<td>Interface Design</td>
<td>54</td>
</tr>
<tr>
<td>6.9</td>
<td>Method/Class Interfaces</td>
<td>55</td>
</tr>
<tr>
<td>6.10</td>
<td>User Interface Design</td>
<td>56</td>
</tr>
<tr>
<td>7.</td>
<td>Procedural Design</td>
<td>49</td>
</tr>
<tr>
<td>7.1</td>
<td>PseudoCodes</td>
<td>49</td>
</tr>
<tr>
<td>7.1.1</td>
<td>“radex” package</td>
<td>49</td>
</tr>
<tr>
<td>7.1.2</td>
<td>“arac” package</td>
<td>50</td>
</tr>
<tr>
<td>7.1.3</td>
<td>“database” package</td>
<td>51</td>
</tr>
<tr>
<td>8.</td>
<td>Future Work</td>
<td>52</td>
</tr>
<tr>
<td>9.</td>
<td>Appendix</td>
<td>53</td>
</tr>
<tr>
<td>9.1</td>
<td>Gantt Chart</td>
<td>53</td>
</tr>
<tr>
<td>10.</td>
<td>References</td>
<td>54</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

1.1. MOTIVATION

The invention of computers and the advancement in data storage technologies flourished the use of electronic documents to store data. Since electronic documents have so much advantages over manuscripts or typewritten documents, this was a profound technology revolution that had a huge impact in our lives.

Yet, electronic documents don’t reveal the information they have immediately, a human being still has to read a free-text electronic document to comprehend its contents. This is a bottleneck for information identification and association of related information. To be able to maximally make use of the electronic platform, information should be easier to obtain, search and identify.

There is a great attempt in this trend in a number of ways. Semantic web technology can be given as an illustrating example. Semantic web tries to extend the web to such an extent where content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting them to find, share and integrate information more easily.

1.2. PROJECT TOPIC

Project topic is text-mining in Turkish radiology reports. Our radiology report analyzer is named RadeX, an abbreviation for Radiology Data Extractor.

1.3. PROJECT DESCRIPTION

The main purpose of our project is to extract meaningful data out of free-text radiology reports, so that the collected data can be easily manipulated and searched on demand. The clinical records and reports of patients contain much potentially useful information in free text form that is not directly searchable. By extracting useful data from clinical reports, records of patients can be held at databases, which will drastically help the diagnosis of further or future clinical problems of patients. Moreover new correlations about some illnesses or drugs such as a drug’s unnoticed side affect could be more easily discovered. Such advancement will help the medical science and diagnosis a lot.

Text-mining on Turkish radiology reports is a challenging subject since there is not much research about text-mining on Turkish texts. Additionally we will face with the complex structure of Turkish as well as hundreds of medical terms. On the other hand, the project will be very handful for academic use and it is an important research on automated medical information systems. In order to extract high quality data out of free-text reports we have to choose the right text mining techniques such as specific natural language processing and machine learning methods.
2. CURRENT STATUS

2.1. WHAT WE HAVE DONE SO FAR

We have done considerable progress in the last few months both in research and implementation.

Most progress was on semantic association part. We dealt a lot with external lexicons and started to build our own lexicon. Before looking unknown words in dictionaries we had to find stem of the words as well as English equivalence of them. We can find stem of some words using a right to left affix stripping method even without using a dictionary. Also using string matching we can do some basic spellchecking.

We completed the implementation of querying Word Net using JWNLI, so that we can find the sense of the words, which are present in Word Net.

In machine learning subject we have also done some progress. Using SVMLight and metu-sabanci corpus, we were able to do named entity recognition with an average success. On the implementation side, moreover, we have done the connection with the database and implemented basic database queries using postgresql. Lastly, our preprocessor module as well as general class structure of the project has been improved.

2.2. RESEARCH

2.2.1. DB4O

db4o (database for objects) is an open source and object oriented database for Java and .NET platforms. Object oriented databases (object databases for short) differ from their relational counterparts in many ways. In Object oriented databases, objects are stored as they are, no tedious mapping overhead of relational databases is necessary. Moreover, object oriented queries don't need to be written in SQL, just using the expressions of the underlying language (Java or C# for db4o) is sufficient.

One main inadequacy the object oriented databases have is that they are only accessible from the programs that know their structure of inner content. This makes them inconvenient for the programs that need their data to be accessible by external applications.

Nevertheless, the advantages that they possess render OO databases extremely useful for some particular applications. They are the remedy for the necessity of native persistence. In our case, our program has to access to lots of data that has nothing to do with the outside world. We have several lexicons in our program that should be accessed very frequently. Additionally considering we use machine learning methods, we need to store a great number of parameter values related to neural networks or decision trees. We can’t just use Java object serialization, since it would cause a huge memory consumption to search for a lexeme in a lexicon, via deserializing all the objects.

According to [5] some of db4o’s customers are BMW, Boeing, Bosch, Intel, Ricoh and Seagate.
2.2.2. WORDNET

WordNet is a semantic lexicon for English language. It has a concept of synset, which designates a set of one or more synonym words. Nouns, verbs, adjectives and adverbs in English are grouped into collections of synsets, each expressing a different concept. Synsets are reciprocally linked by means of conceptual and semantic relations. The resulting network of meaningful words can be navigated with a browser, besides it is also publicly available for download. WordNet offers a very powerful ontology for English, so it is a useful tool for natural processing research in English. Our project is regarding Turkish radiology reports. However there is not any freely available WordNet like semantic lexicon for Turkish. This inspired the idea of first translating a Turkish word into English and then exploiting the use of WordNet to get the part of speech tag and the sense of the word.

WordNet was created and is currently maintained at the Cognitive Science Laboratory of Princeton University. The development began at 1985, and the project received about $3 million of funding from government agencies interested in machine translation [6].

2.2.3. JWNL

JWNL stands for Java WordNet Library. It is a free of charge and open source Java API for accessing WordNet. It is well documented and requires the knowledge of a moderate number of functions and data types to make use of. Hereby, it doesn’t take much time to start coding. This project is hosted at sourceforge.net [10].

2.2.4. SNOWBALL

Although dictionary-based stemmers provide better performance, algorithmic stemmers also contribute great utility in Information Retrieval. Snowball is a string processing language in which these algorithmic stemmers can be implemented. It is easy to learn and to compile it into ANSI C or Java. Most important feature of Snowball is it is language independent.

3. DESIGN CONSTRAINTS

3.1. NAMING AND DOCUMENTATION CONSTRAINTS

Using understandable, consistent names for identifiers, proper commenting and documentation are important issues for shared implementation as well as maintenance of the project.

Identifier names for variables, constants, functions and classes should be self describing, clear and understandable. Again, for clarity purposes these names can be chosen as Turkish words, where appropriate. Since we are going to use Java, we have to obey restrictions that Java implies. Identifier names containing multiple words should be written without using underscores, for example, instead of kelime_gruplarini_bul(), we will use kelimeGruplariniBul(). If an identifier name is very long and we will use abbreviations for the first terms, like ANN Learner, NER Recognizer, etc.

Commenting is also another important issue. In the beginning of important functions we will include a small pseudo code for that function as a comment. Separate sections of code in a single file should be easily
distinguished again by using proper commenting. The language for commenting is not important, both Turkish and English can be used. The important thing is; commenting should be clear enough so that later we can remember what those functions do.

We will also start documentation of the project when we start implementation. Documents should describe every detail of the project. It should be understandable by anyone, even by those who has no idea about text mining or any other technical stuff.

3.2. TIME CONSTRAINTS

We have only six months to finish our project, therefore we have to obey our Gantt chart as much as possible in order to avoid possible delays. Preparation of the detailed design report and implementation of the first prototype should be done in a month. In the following one and a half week debugging and necessary performance tests for the prototype should be thoroughly performed.

Since we will implement more than one builds, we should take it very seriously to make the produced builds have a stable state. Each build will be made as following the other ones; it would cause a huge waste of time and effort if we come across a bug after the third or fourth build, whose root lies in the first build.

This one and a half month is very important for us since we should finalize our design and implement a demo program having all the basic characteristics of our final project.

3.3. USER INTERFACE CONSTRAINTS

We are planning to have two different user models in our project. The first will be an admin-like user, who will have the ability to upload new reports to be analyzed. The program should permit analyzing more than one report at the same time. The admin should be able to correct the wrong results of an analysis in order to prevent storing wrong information.

The second user will be a searcher. The searcher will do the search using specific criteria so that he/she will reach the desired information easily. Our user interface should provide the necessary means to implement these functionalities. Besides it shouldn’t restrict our design in a way that we have to change it.

3.4. PERFORMANCE CONSTRAINTS

In text mining every single word should be equivalent to a meaning. This can be done in two ways. The first is searching the lexicon; the other is searching in internet. Because it can take some time to connect to internet, searching the lexicon should prior. Since we chose java as language of the program we will be able to use DB4O. This will provide the program to reach the lexicon database faster.
4. DATA DESIGN

4.1. DATABASE TABLES

Our database tables’ structure had gone through some important revisions. The table named ‘Oneri’ is removed. ‘Oneri’ table was used for storing the suggestions referred in the reports. Since a suggestion could be either an operation or a medical treatment, this table requires multiple inheritance (to the tables ‘Islem’ and ‘Ilac_tedavi’). Its integrity constraints are not well-defined and they are tricky; both of the foreign keys it possesses shouldn’t be empty, but one could be empty. This situation is hard to implement in a relational database. Instead of this table, ‘Islem’ and ‘Ilac_tedavi’ tables store an additional field named ‘oneriMi’.

We added a new table ‘Kiyas’, which is explained below. Other than these changes the structure of our database design remains fundamentally the same.

4.1.1. RAPOR

This table holds the base information about an analyzed document. The main attribute of this table is to hold the unique id of the whole analyzed report. Besides, this table holds all the straightforward information existing in a report that doesn’t need to be categorized into more specific entities like problems, findings, so on.

<table>
<thead>
<tr>
<th>Rapor_no</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baslik</td>
<td>String</td>
</tr>
<tr>
<td>Tarih</td>
<td>Date</td>
</tr>
<tr>
<td>Doktorlar</td>
<td>Varchar(40)</td>
</tr>
<tr>
<td>Hasta</td>
<td>Varchar(20)</td>
</tr>
</tbody>
</table>

- `Rapor_no` is the id of the analyzed report
- `Baslik` is the heading of the report.
- `Tarih` is the date that this report was committed to paper.
- `Doktorlar` is the concatenation of the name of the doctors who wrote this report.
- `Hasta` is the name of the patient that the report was written about.

Since sustaining the names of the patients and the doctors is not very crucial for the quality of information extracted from a report, we didn’t made extra tables for storing values related to them. It would be easy to integrate in our project, if requested. The table about the patients may have fields such as name, age, sex.

4.1.2. PROBLEM

This table holds detailed information about a medical problem/abnormal finding that subsists in the document. It may be the case that the patient doesn’t suffer from the problem. We still hold information about it in this table.
**Problem_no** is the primary id of the table.

**Rapor_no** is a foreign key to the Rapor table that designates the report in which this problem subsists.

**Rapor_bolum** field holds the section of the report that this problem subsists in.

**Problem** field holds the name of this problem.

**Bolge** field holds the body part on which this problem takes place, such as 'breast'.

**Alt_bolge** field holds the more specific body part, such as 'areola of the left breast'.

**Derece** field holds the severity of the problem.

**Aciklayicilar** field is to store the descriptors of the problem concatenated by a whitespace. Most of the problems in medical reports have at least one and at most 3 descriptors. So using a separate table for this entity wouldn't be a good choice.

**Tespit_tarih** field holds the date the problem was detected.

---

### 4.1.3. NORMAL_BULGU

This table holds detailed information about a normal condition finding that doesn’t emphasize existence or nonexistence of a problem, as depicted by the sentence 'the heart size is normal'.

<table>
<thead>
<tr>
<th>Bulgu_no</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rapor_no</td>
<td>Integer</td>
</tr>
<tr>
<td>Rapor_bolum</td>
<td>Varchar(20)</td>
</tr>
<tr>
<td>Bulgu</td>
<td>Varchar(20)</td>
</tr>
<tr>
<td>Nitelik</td>
<td>Varchar(20)</td>
</tr>
<tr>
<td>Bolge</td>
<td>Varchar(20)</td>
</tr>
<tr>
<td>Kesinlik</td>
<td>Enum { Kesin, Yuksek Ihtimal, Dusuk }</td>
</tr>
<tr>
<td>Tespit_tarih</td>
<td>Date</td>
</tr>
</tbody>
</table>

**PRIMARY KEY(Bulgu_no)**

**FOREIGN KEY(Rapor_no) REFERENCES(Rapor)**

**INDEX(Rapor_no)**

**CONSTRAINT NOT NULL(Rapor_no), NOT NULL(Rapor_Bolum)**
11

- **Bulgu_no** is the primary id of the table.
- **Rapor_no** is a foreign key to the Rapor table that designates the report in which this problem subsists.
- **Rapor_bolum** field holds the section of the report that this finding subsists in.
- **Bulgu** field holds the subject of the finding, such as 'the heart size'.
- **Nitelik** field holds the predicate of the finding such as 'is normal', 'is clear', 'is natural'. The 'is's parts are just for illustration, they are not part of the value stored in Nitelik.
- **Bolge** field holds the body part that this finding is initially related such as 'heart'.
- **Kesinlik** field holds the certainty of the finding.
- **Tespit_tarih** field holds the date of the finding.

---

### 4.1.4. ISLEM

This table holds information about a technique/operation (ultrasound, chest x-ray, biopsy or radiograph) that subsists in the radiology report.

<table>
<thead>
<tr>
<th><strong>Islem_no</strong></th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rapor_no</strong></td>
<td>Integer</td>
</tr>
<tr>
<td><strong>Rapor_bolum</strong></td>
<td>Varchar(20)</td>
</tr>
<tr>
<td><strong>Islem_adi</strong></td>
<td>Varchar(20)</td>
</tr>
<tr>
<td><strong>Baslangic_tarih</strong></td>
<td>Date</td>
</tr>
<tr>
<td><strong>Sure</strong></td>
<td>Integer</td>
</tr>
<tr>
<td><strong>Bolge</strong></td>
<td>Varchar(20)</td>
</tr>
<tr>
<td><strong>Aygit</strong></td>
<td>Varchar(20)</td>
</tr>
<tr>
<td><strong>Aciklama</strong></td>
<td>Varchar(30)</td>
</tr>
<tr>
<td><strong>OneriMi</strong></td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**PRIMARY KEY(Islem_no)**

**FOREIGN KEY(Rapor_no) REFERENCES(Rapor)**

**INDEX(Rapor_no)**

**CONSTRAINT NOT NULL(Rapor_no), NOT NULL(Rapor_Bolum)**

- **Islem_no** is the primary id of the operation.
- **Rapor_no** is a foreign key to the Rapor table that designates the report in which this suggestion was proposed.
- **Rapor_bolum** field holds the section of the report that this operation subsists in.
- **Islem_adi** field holds the name of this operation.
- **Baslangic_tarih** field holds the date this operation started.
- **Sure** field holds the number of days the operation lasted, or will last approximately.
- **Bolge** field holds the body part/body organ that this operation was processed on like the 'chest' in 'chest x-ray'.
- **Aygit** field holds about the device that the operation was carried out with, like x-ray.
  - This field somewhat seems as if colliding with the **Islem_adi** field, but actually they are two different things. To illustrate in the sentence ‘An ultrasound operation to confirm that these are real cysts is required’ they both have the value ‘Ultrasound’. But the operation does not have to contain a device name, as in the sentence ‘comparison with previous [Image]
studies is suggested’ or ‘biopsy of this mass could be made without any contingent side effects’.

- **Aciklama** field holds some explanation about this operation.
- **OneriMi** field holds if this operation is a suggestion or not.

### 4.1.5. ILAC_TEDAVI

This table holds information about a medication treatment that subsists in the analyzed radiology report.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ilac_tedavi_no</td>
<td>Integer</td>
</tr>
<tr>
<td>Rapor_no</td>
<td>Integer</td>
</tr>
<tr>
<td>Rapor_bolum</td>
<td>Varchar(20)</td>
</tr>
<tr>
<td>Ilac</td>
<td>Varchar(20)</td>
</tr>
<tr>
<td>Baslangic_tarih</td>
<td>Date</td>
</tr>
<tr>
<td>Sure</td>
<td>Integer</td>
</tr>
<tr>
<td>OneriMi</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**Ilac_tedavi_no** is the primary id of the operation.

- **Rapor_no** is a foreign key to the Rapor table that designates the report in which this suggestion was proposed.
- **Rapor_bolum** field holds the section of the report that this operation subsists in.
- **Ilac** field holds the name of the drug used.
- **Baslangic_tarih** field holds the date of this treatment started.
- **Sure** field holds the number of days the operation lasted, or will last approximately.
- **OneriMi** field holds if this treatment is a suggestion or not.

### 4.1.6. KIYAS

This table stores the comparison of a problem to an early phase. Not much of the problems have comparison information associated with them, so holding this information in another table hopefully will save some space.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kiyas_no</td>
<td>Integer</td>
</tr>
<tr>
<td>Problem_no</td>
<td>Integer</td>
</tr>
<tr>
<td>Kiyas_tarih</td>
<td>Date</td>
</tr>
<tr>
<td>Degisiklik</td>
<td>Varchar(30)</td>
</tr>
</tbody>
</table>

**Kiyas_no** is the primary id of the suggestion.
- *Problem_no* is a foreign key to the Problem table that designates the problem that was compared to the information in this table.
- *Kiyas_tarih* field depicts the date of the phase of the problem that it is compared with.
- *Degisiklik* field stores the change that is referred in the report.

4.2. ER DIAGRAM

Here is the overall ER diagram of our database design. The attributes of the entities are omitted from the figure since they are explained thoroughly at the previous section.

![ER Diagram](image)

**LEGEND**

- **Entity**
- **Relation**
  - One and only one
  - Zero or more

All the types except 'Rapor' should have all their instances contained by another type. This inference shows that, actually the four remaining tables may be modeled as weak entity sets. However, the fields in the other tables do not have key-like characteristics. To illustrate, we can't introduce a key for 'Islem' table by using the
‘rapor_id’ and all of its attributes. So we must add a primary key field to ‘Islem’ table. Upon adding a primary key field to a table, there is no more need to model it as a weak entity set.

5. ARCHITECTURAL DESIGN

5.1. STRUCTURAL MODELING

5.1.1. DATA FLOW DIAGRAMS

5.1.1.1. Level-0

5.1.1.2. Level-1 “RadeX”
5.1.3. **Level-2 “preprocess reports”**

5.1.4. **Level-2 “associate semantics”**
5.1.1.5. **Level-2 “learn and extract”**

5.1.1.6. **Level-3 “extract data”**
What’s new?

**Learn and Extract**
This module is now more detailed although the details are not shown explicitly in DFD’s, we have explained them in class design. We added Level-3 for “Extract Data”.

**Tag named entities**
This module will deal with machine learning stuff by creating a knowledge base(model) using a training corpus and does the named entity recognition. It will help especially in finding sense of words which cannot be found just using semantic associator. The feature sets of machine learning used in named entity recognition will be mostly sense, POS, and affixes of words and predicate of clause acquired from semantics association.

**Template matching**
First of all, we added a new function “match with templates” in Level-3 extract data. Template matching will be a rule-based information extractor, which will take semantics of words that are already found at previous modules, and feed these words to pre-defined templates in order to collect a more meaningful formatted data. This will help to organize sentence semantics composed of senses of individual words. In other words, using template matching we will find out relationships between word-phrases.
# Data Dictionary for Data Flow Diagrams (LVL 0-1-2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Use area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free-text reports</td>
<td>Free-text with certain sections</td>
<td>Analyzer class</td>
<td>It's the input for the report analyzer module.</td>
</tr>
<tr>
<td>Formatted data</td>
<td>XML</td>
<td>Analyzer and Searcher classes</td>
<td>It's the information extracted from the report.</td>
</tr>
<tr>
<td>Query string</td>
<td>String</td>
<td>Searcher class</td>
<td>It's used to search a string on database. It may be a single word or group of words.</td>
</tr>
<tr>
<td>Query result</td>
<td>Data object, XML</td>
<td>Searcher class</td>
<td>Result of a query is stored in a Data object. Data object stores rows Then it's printed in XML format.</td>
</tr>
<tr>
<td>Parsed logical report</td>
<td>Report object</td>
<td>In all classes</td>
<td>Logical report holds sections, sentences, words and phrases of corresponding physical report.</td>
</tr>
<tr>
<td>Unknown word/term</td>
<td>String</td>
<td>snomeddeAra(), lexicondaAra()</td>
<td>It's the word sent to lexicons and SNOMED to gain information about it.</td>
</tr>
<tr>
<td>Word with semantics</td>
<td>Lexeme object</td>
<td>MedicalLexicon, SifatLexicon and YuklemLexicon classes, snomeddeara(), lexicondaAra()</td>
<td>It’s created as a result of a successful query in Lexicons, SNOMED, and TDK. It holds semantic information about words.</td>
</tr>
<tr>
<td>Current/New knowledge</td>
<td>-</td>
<td>Learner class</td>
<td>It forms the knowledge-base. It’s updated when learning algorithm learns new knowledge.</td>
</tr>
<tr>
<td>Report sections</td>
<td>RaporKisim object</td>
<td>Preprocessor class</td>
<td>It holds the sentences of the corresponding section in physical report.</td>
</tr>
<tr>
<td>List of words</td>
<td>Cumle object</td>
<td>Preprocessor class</td>
<td>It holds words of the corresponding sentence in physical report.</td>
</tr>
<tr>
<td>Words with POS</td>
<td>MyKelime object</td>
<td>Preprocessor class</td>
<td>It's created for every word in the report. If Zemberek recognizes the word, it updates the object with its POS, root and suffixes.</td>
</tr>
</tbody>
</table>
5.2. MODULER HIEARARCHY
5.3. MODULE EXPLANATIONS

RadeX will consist of two main components: Report Analyzer and Database Searcher. We are planning to separate these two modules into two different executables.

5.3.1. REPORT ANALYZER COMPONENT

This module is most complex part of the project. It basically receives free-text radiology reports, and performs some text-mining operations to extract meaningful information. It consists of four sub-modules, namely, preprocessor, semantic associator, learner and finalizer.

5.3.1.1. PREPROCESSOR MODULE

This is the first module of the program. It contains all the classes and functions to parse and process raw text. Preprocessor module creates a report object (which we call logical report throughout this report) corresponding to a raw (physical) report. Again, this module consists of different sub-modules and routines each of which fills some fields of the logical report.

Section Splitter
Here, report will be divided into main sections: Tur, Klinik Bilgi, Teknik, Bulgular, Sonuc and Doktorlar. Each section has a corresponding RaporKisim object in the logical report.

Sentence Splitter
As the name implies, this routine splits each RaporKisim object into its sentences. For each sentence, a new Cumle object is created.

Tokenizer
This routine splits sentences into tokens. Then, each word will be sent to Word Processor.

Word Processor
Here is the routine, where Zemberek comes into play. Each word will be processed using Zemberek one by one. If Zemberek recognizes the word, a corresponding MyKelime object is created. If not, these fields left empty for now.

5.3.1.2. SEMANTIC ASSOCIATOR MODULE

It is the module, where words gain meaning, i.e. their classes and ontological meanings are discovered. Adjectives and predicates are also fit into certain groups according to their meanings.

This module consists of several query modules and named entity recognizer to achieve proper classification of words.

Query Own Lexicon
This module consists of three sub-modules, Query MedicalLexicon, Query YuklemLexicon, and Query SifatLexicon. In these modules, unknown medical terms, predicates and adjectives are queried in the existent lexicons built by us. If query succeeds, a Lexeme object corresponding to that word is returned. This object contains type/class information of the word as well as Turkish and English translations.

Query SNOMED
Here unknown terms are queried in SNOMED. If query succeeds, the word gains its category and ontological place in SNOMED hierarchy.
Query TDK/Word net
This module is used when the above queries fail, i.e. the word is neither in our lexicons nor in SNOMED. Here we try to associate meaning to a word using Word Net, with the help of TDK. Details can be found on New Research section under Current Status.

Named Entity Recognizer
If we can build or find a suitable training corpus, we are planning to use this module for further classification.

5.3.1.3. LEARNER

Details of this module are not yet clear. But we are planning to use neural networks algorithms and decision tree learning methods to enhance the quality of extracted information and to reach information that cannot be extracted using other classification techniques.

SVM Learner
This module will use svm algorithms to extract new information.

Decision Tree Learner
This module will use decision trees to extract new information.

Bayesian Learner
This module will use decision naïve Bayesian classifier to extract new information.

5.3.1.4. FINALIZER

This module consists of following sub-modules.

Information Collector
After semantic associator and learner modules finish their jobs, all the extracted information will be scattered among words and sentences and even report sections. This information should be collected somehow before formatting to display. This module will start working here.

Data Formatter
This module is responsible for building XML formatted view of all collected information. Also it will construct SQL statements for insertion.

Database Connector
This module will establish the connection between Report Analyzer and the database. Then, it will send all the data with the SQL statements prepared by Data Formatter module.

5.3.1.5. SPELL CHECKER

Spell Checker
As the name implies, it consists of some sub-routines in order to check spellings of words, and also to find close/similar words to recognize unknown words.
The second main module of the program is database search module. It gives the user the ability to search the database by submitting a query. It consists of three sub-modules: Database Connector, ResultSet Reader and Displayer.

5.3.2.1. DATABASE CONNECTOR

This module will establish the connection between Database Searcher and the database. It will prepare an SQL query statement for the string entered by the user, and commit.

5.3.2.2. RESULTSET READER

Result of the query will be analyzed and an XML format will be created here.

5.3.2.3. DISPLAYER

Here, the prepared document will be displayed to the user using GUI.

5.4. CLASS MODELING

5.4.1. PACKAGE/CLASS DESCRIPTIONS & CLASS DIAGRAMS

We have divided our classes into eight packages, namely anlam, arac, database, gui, lexicon, malumat, radex, rules, yapi.

For a legend of class diagrams

- The **green** fields signify a public member.
- The **red** fields signify a private member.
- The **yellow** fields signify a protected member.
- Fields with a superscript `s` (`^s`) are static members.
- Methods with a superscript `C` (`^C`) are constructors.

5.4.1.1. “anlam” package

This package contains one interface with only one method: AnlamTipi, and two enumerations implementing AnlamTipi. This package reveals our feature selection procedure for *sense* extraction of word in a radiology report. After having analyzed the radiology reports, we came into the conclusion that all the predicates and adjectives in the reports actually can be reduced to 15-20 different ones with actual different *senses*.

Predicates in the reports are usually verbs, but there are cases when they are nouns or adjectives. In that case, the sense of the predicate is ‘medikal’ and it is found using IsimAnlamTipi.
In IsimAnlamTipi the fields Egilim, Kesinlik, Miktar, Derece, Pozisyon, Durum are reserved only for adjectives not nouns. We use the same schema for both of nouns and adjectives because most the adjectives are relational. They just refer to a noun. Some examples are renal and kidney, abdominal and stomach.

The method named elements returns all the values in the enumeration. This is useful to assign a feature these values;

5.4.1.2. “arac” package

This package contains the all tool functioned classes that we will implement in our program, namely Anlamsalliskilendirici (Semantic Associator), Preprocessor, SpellChecker, Tdk, Finalizer, NERecognizer, Chunker, WordNet, Zargan and TextNormalizer.

The words we gathered in the feature selection for Turkish radiology reports have intuitive corresponding English counterparts in WordNet. To illustrate, “body part” is a category in WordNet corresponding to Turkish “organ bolgesi”. For “hastalık” the counterpart is “health problem”. To make use of WordNet we first translate a Turkish word to English by using an external dictionary. Next we look for the synonyms and hypernyms of the English words until we come across one of our categories.
We added two important classes since initial design report, Chunker and NERecognizer. Chunker uses the methods in ‘rules’ and ‘malumat’ packages for shallow parsing. NERecognizer uses ‘malumat’ package to find named entities by machine learning methods. So these both classes store a db4o object storing their instances, and instantiate the Learner s using this instance database object. Chunker also stores the directories of its rules files.

Another new class is TextNormalizer, this class will contain the methods to convert the non-text files to text files.

5.4.1.3. “database” package

This package contains the corresponding java classes of the database tables that were explained in the section 3.1. Other than those it contains two other classes: Data and Database.

Data is an abstract class that has one abstract method named toGrid. All the database table classes are required to have a concrete implementation of this method. This method will be used as a helper to view a dataset table as a JTable.

Database class is to abstract the database related functionalities of radex. Class diagrams for the database package are below.
Database

- bagIM: boolean
- bulgular: Vector<Normal_bulg extends Data>
- datalar: Vector<Data>
- db: Connection
- dbmd: DatabaseMetaData
- exc: Exception
- host: String
- ilacTedavliler: Vector<ilac Tedavi extends Data>
- instance: Database
- isemler: Vector<Isem extends Data>
- kiyaslar: Vector<Kiyas extends Data>
- pass: String
- problemler: Vector<Problem extends Data>
- raporlar: Vector<Rapor extends Data>
- sql: Statement
- tedaviler: Vector<Rapor extends Data>
- user: String

Database()
- eklar()
- baglan(): boolean
- baglantiyiSonlandir()
- baglani(): boolean
- createTables()
- gettInstance(): Database
- gomData(n vec: Vector<Data>)
- gomIlacTedavi(n r: ilac Tedavi)
- gomIsem(in r: Isem)
- gomKiyas(n r: Kiyas)
- gomNormal_bulgur(n: Normal_bulgur)
- gomProblem(n t: Problem)
- gomRapor(n r: Rapor)
- initialize()
- queryRapor(in key: int): Vector<Vector <E>>
- queryStringAnhtar(in keyword: String): Vector<Integer>
- queryStringilac_Tedavi(in key: String): Vector<Integer>
- queryStringIsem(in key: String): Vector<Integer>
- queryStringKiyas(in key: String): Vector<Integer>
- queryStringNormal_bulgur(in key: String): Vector<Integer>
- queryStringProblem(in key: String): Vector<Integer>
- queryStringRapor(in key: String): Vector<Integer>
- setHost(in host: String)
- setPass(in pass: String)
- setUser(in user: String)
- temizle()
5.4.1.4. "lexicon" package

This package contains the data structures and classes that we use to implement our own lexicons.
All of Lexicons do have a db4o.ObjectContainer property named db. That property holds the db4o database object to store the lexicon in the file system.

Since there will be only one instance of a lexicon, they do not have a public constructor. Rather than that, they have a public method getInstance() that returns the only instance of that lexicon.

Lexeme, YuklemLexeme are the classes that the lexicons store the instances of.

Lexeme means a vocabulary entry in English. It is different than the yapi.MyKelime class in a number of ways. It doesn't need to store the suffixes of a word. The words 'ruin', 'ruined', 'ruining' have all the same Lexeme 'ruin', whereas they are different words.

5.4.1.5. "malumat" package

This package is supposed to hold classes implementing machine learning algorithms that we will use.
All the learners hold a db4o database which holds their training instances; Learners implement the java library interface Externalizable, in order to provide persistence of some of their attributes, such as the nodes of a decision tree.

Most ML algorithms require the features represented in numeric values. Hence, Feature class does not have a public constructor, it has a factory inside which assigns a unique integer to each feature.

TargetValue class is to hold the value of the classified feature (target). It isn’t just an object because there are additional fields that we should consider such as its probability.

Instance class holds either a training instance or a test instance. It holds the features, their values, the target feature and its value if it is determined.
5.4.1.6. “rules” package

This package holds the class RuleBase, which we will use for template matching.

Since there could be many more relations between words in a sentence we decided not to implement template matching rules directly in Java but in another simple language that we designed.

The methods in the RuleBase class convert these files to corresponding java files that does template matching over words in a sentence.

The generated files publicly visible will just have a match method that iterates over words and returns some data.

The generated classes should all implement an interface specifying their return types.

Here are some sample rules

```
@ N
database.Data[] (database.Bulgu[]) ret
   implements rule.DataRule
   var x<Database.Bulgu> .defaultTokenValueGetter .icerik()+' '
   ( <pos:ISIM anlam:ORGAN|ORGAN_BOLGESI> { x.bulgu = this }  
     <pos:ISIM anlam:ORGAN ekler=DeHalEki>? { x.bulgu += this }  
     <pos:SIFAT>* { x.nitelik += this }  
     <YUKLEM> { x.nitelik += this }  
     ( ret.add(x) )
   ).empty x )+

@ N
yapi.KelimeGrubu[] ret
   implements rule.DataRule
   var x<yapi.KelimeGrubu> .concatAction .kelimeler.add
     <pos:SIFAT> { x += this }  
     <pos:SIFAT anlam=ORGAN|PROBLEM>+ { x += this }  
     <pos:ISIM> { x += th
     ( ret.add(x) )
```


@ NP
database.Data[] (database.Problem[]) ret
implements rule.DataRule

    var x<Database.Problem>
    .defaultTokenValueGetter .icerik()+''
    {<
        <pos:SIFAT icerik="bilateral"><pos:ISIM anlam:HASTALIK> { x.bulgu = this }<pos:ISIM anlam:ORGAN ekler=DeHalEki>? { x.bulgu += this }
        <pos:SIFAT>* { x.nitelik += this }
        <YUKLEM> { x.nitelik += this }
        ( ret.add(x) ) .empty x
    }

We just use EBNF notation coupled with Java code.

- A rule file may consist of several rules where they all begin with the sign @
- The word next to @ sign ( N or NP ) decides whether we iterate over words or word phrases.
- The tokens between <> signs match a token (word or word phrase)
- Tokens can have regular operators *.,?,+ just like EBNF notation.
- The right part of a token between {} signs are semantic actions that are executed when a token is matched.
- The fields inside the token matcher signs are just the fields of Kelime class (like icerik, pos)
- First line after @ denotes the value that this rule returns
- The next line denotes the interface that this class implements
- There are some directives starting with a dot ('). .empty directive assigns a new object to the variable at the right hand side.

5.4.1.7. “radex” package

This package contains two classes namely Analist and Sorgucu which stands for Report Analyzer component and Database Searcher component.

At the moment all these two classes have is the main method.
5.4.1.8. “yapi” package

This package exists in order to store the structures of linguistic entities. All the classes in this package implement java.lang.Cloneable interface, and override default toString() methods. Moreover all of their member variables are private and have setters and getters.

- Kelime is not a class that we created, it is part of Zemberek. Our own class is named MyKelime which extends Kelime.
- KelimeGrubu stands for Noun Phrase.
- Cumle stands for sentence.
- Yuklem stands for predicate.
- Kesinlik stands for exactness.
- RRapor is the class that holds all the information related to an analyzed radiology report.
- RaporKisim class is to store the different sections in a radiology report in different places.
5.4.1.9. "gui" package

This package stores the classes related to our graphical user interfaces.
5.5.1. “Analist” Sequence Diagram
5.5.1.2. “Sorgucu” Sequence Diagram
5.5.1.3. “Preprocessor” Sequence Diagram
5.5.1.4. “Anlamsallıskilendir” Sequence Diagram
5.5.2. ACTIVITY DIAGRAMS

Activity Diagram for "Report Analyzer"

Activity Diagram for "Searcher"
6. INTERFACE DESIGN

6.2. USER INTERFACE DESIGN

In this section, we will describe how the interaction between users and RadeX will take place. As we described in our requirements analysis report, there will be two different user types: Report supplier (admin) and searcher.

General functionalities (i.e. both for admin and searcher) of the interface are on “Dosya” and “Yardım” menus in menu bar. “Dosya” includes “Çıkış” option which terminates the program. “Yardım” includes “Radex hakkında” option, which gives the usage and version information about radex; “Tiran software hakkında”, which gives information about developers of Radex.

Interactions between Admin and Report Analyzer component:

- In order to load report file/files or directory, user should choose “Sisteme rapor yükle” option from “Dosya” menu shown in figure B.

- As it is seen on Figure A, user can choose one or more files or a folder to be loaded.

- Opening these reports; initial states (unprocessed) of reports will be inserted into “İşlenmemiş raporlar” node of the “Dizin” of the tree view as shown in Figure B.

- Upon the user clicks “Analiz” button, the program will process the report (or reports); take the report from “İşlenmemiş raporlar” and put into “İşlem sonuçları”. The highlighted leaf in “İşlenmemiş raporlar” node will display the result (Figure C) of the chosen report.

- User may edit the text fields on desire.

- User must click “Kaydet” button to put the extracted data into database.
Figure B

Figure C
Interactions between Searcher and Program:

- User searches the keywords via text field shown in figure D.
- Program will list the compatible result as listed in figure D.

![Figure D](image)

6.3. METHOD/CLASS INTERFACES

6.3.1. “arac” PACKAGE

**static method** TDK::tdkSorgulaKelime

- **input parameters** : \( s \) – String
- **return type** : Vector<String>

- This method stores all the definitions of \( s \) in TDK in a vector and returns that vector.
- If there is no definition in TDK return value is null.
**static method** TDK :: tdkIngilizceyeCevir
- **input parameters** : s – String
- **return type** : String
  - This method returns the english translation of the Turkish word s
  - If translation is unsuccesull return value is null.

**static method** SpellChecker :: kelimeDuzelt
- **input parameters** : s – String
- **return type** : String
  - This method tries to fix a possibly erroneously spelled Turkish word
  - If fix is unsuccesull return value is the same string s.

**static method** SpellChecker :: yakinKelimele
- **input parameters** : s – String
- **return type** : Vector<String>
  - Here s is a jumble word written in Turkish medical terminology. This function tries to convert it to the actual English or Latin equivalent.
  - It returns a vector of possible retouched words indexed in the order of their contingency.

**static method** SpellChecker :: kokBul
- **input parameters** : s – String
- **return type** : Vector<String>
  - This method tries to find the root of the jumble word or correctly converted word s.
  - It returns a vector of possible roots indexed in the order of their contingency.

**method** Preprocessor :: preprocess
- **input parameters** : None
- **return type** : None
  - This method does preprocessing of the report rapor.
method AnlamsalIîiskilendirici :: ilîiskilendir
  - input parameters: None
  - return type: None

• This method does semantic association of the report rapor.

method Finalizer :: databaseTablolariniOlusturVeDon
  - input parameters: None
  - return type: Vector<database.Data>

• This method constructs the database tables for the parameter passed to its constructor. Next, it returns those tables in the form of a vector.

method WordNet :: getPos
  - input parameters: s - String
  - return type: String

• This method returns the pos tag of the English word s using Word Net.

method WordNet :: getSense
  - input parameters: s - String
  - return type: String

• This method returns the sense of the English word s using Word Net.

method Chunker::chunkUsingRules
  - input parameters: r - RRapor
  - return type: None

• This method finds noun phrases in the given Rrapor r via template matching.

method Chunker::chunkUsingML
  - input parameters: r - RRapor
  - return type: None

• This method finds noun phrases in the given Rrapor r using machine learning methods.
method NErecognizer::NErecognizeUsingRules

- input parameters : r: RRrapor
- return type : None

- This method uses template matching in order to find named entities.

method NErecognizer::NErecognizeUsingML

- input parameters : r: RRrapor
- return type : None

- This method uses machine learning in order to find named entities.

6.3.2. “rules” PACKAGE

method RuleBase::generateRule

- input parameters : in: File, out: File
- return type : None

- This method translates a rule file into java code.

method RuleBase::compileRule

- input parameters : in: File, out: File
- return type : None

- This method compiles a java code into class file.

method RuleBase::generateAllRules

- input parameters : sourcedir: File, j_sourcedir: File
- return type : None

- This method calls generateRule for each file in a given directory sourcedir.

method RuleBase::compileAllRules

- input parameters : sourcedir: File, j_sourcedir: File
- return type : None

- This method calls compileRule for each file in a given directory sourcedir.
method RuleBase::loadRuleBinaries
  - input parameters: ruleVec – Vector, dir – File
  - return type: None

- This method creates instances of the classes in the given directory dir using reflection and returns them via ruleVec

---

6.3.3. “malumat” PACKAGE

method AbstractLearner::addInstance
  - input parameters: i - Instance
  - return type: None

- This method adds an instance to the container of Learner.

abstract method AbstractLearner::classify
  - input parameters: i - Instance
  - return type: TargetValue

- This method classifies the given Instance i and returns the value the found target.

method Feature::createFeature
  - input parameters: name – String, values – Object[ ]
  - return type: Feature

- This method creates a new Feature with given name and values if it’s not already exist, otherwise it returns null.

method Feature::getFeature
  - input parameters: fname – String
  - return type: Feature

- This method retrieves the feature with the given fname. It returns null if not found.
method Feature::isValueOf

- **input parameters**: o – Object, i – int
- **return type**: boolean

- This method returns whether the value and numeric of the feature matches or not.

6.3.4. “database” PACKAGE

abstract method Data :: toGrid

- **input parameters**: none
- **return type**: Vector<Object>

- This method returns a vector of all the members of the implementer class.

- This method is to be used as a helper to display the Data in a JTable.

method Database :: gomData

- **input parameters**: v – Vector<Data>
- **return type**: None

- This method inserts all the element Datatas of v to this class.

method Database :: temizle

- **input parameters**: None
- **return type**: None

- This method empties all the vectors in this Database class holding any Data.

method Database :: queryRapor

- **input parameters**: no – int
- **return type**: Vector<Vector>

- This method queries the physical database for all the Data related to the Rapor having rapor_no no and returns all the Data in a vector.

method Database :: queryStringAnahtar

- **input parameters**: key - String
- **return type**: Vector<Integer>
• This method queries the physical database for all the Data having some field containing the keyword key. Next it returns the vector of raporID containing the key.

6.3.5. “lexicon” PACKAGE

**method** Lexicon :: init

- **input parameters**: none
- **return type**: none

• This method reads the lexicon file from the file system and does some necessary initialization.

**method** Lexicon :: put

- **input parameters**: l - Lexeme
- **return type**: none

• This method inserts the Lexeme l to this lexicon.

**method** Lexicon :: getTurkce

- **input parameters**: s - String
- **return type**: Lexeme

• This method searches the lexicon and returns the lexeme having its turkce field equal to s.
• If the lexeme does not exist in the lexicon, return value is null

**method** Lexicon :: getIngilizce

- **input parameters**: s - String
- **return type**: Lexeme

• This method searches the lexicon and returns the lexeme having its ingilizce field equal to s.
• If the lexeme does not exist in the lexicon, return value is null

**method** Lexicon :: getInstance

- **input parameters**: None
- **return type**: Lexicon

• This method returns the only instance of this lexicon.
(Actually the abstract Lexicon class doesn't have this method, but since all of its subtypes have this method it is written as if it is part of the Lexicon class.)

6.3.6. "radex" PACKAGE

**method** Sorgucu :: main

- **input parameters**: String []
- **return type**: None

- This function is the entry point of our Database Querier component.

**method** Analist :: main

- **input parameters**: String []
- **return type**: None

- This function is the entry point of our Report Analyzer/Information Extractor component.

6.3.7. "yapi" PACKAGE

The classes in this package don't have any public methods, but just getters and setters for all of their variables which are depicted by the class diagrams a section 4.2.2.7. Additionally they all override the default toString and clone methods, to facilitate debugging and illustration purposes.

Since all these methods are native to Java they don't require additional explanation.

7. PROCEDURAL DESIGN

7.1. PSEUDOCODES

Here are pseudo codes of some important functions.

7.1.1. "radex" PACKAGE

```java
Analist :: main ()
    l <- Lexicon Instance
    d <- Database Instance
    initialize l
    bagl an d
```
display GUI
for each report files
    create Rapor object rapor from path
    create Preprocessor object : p[r]
    preprocess rapor
    create AnlamsalIliskilendirici object  ai from rapor
    iliskilendir rapor
    display rapor
    get editedresult from GUI
    learnFrom editedresult
    send result to d
end
end

Sorgucu :: main()
    d <- Database instance
    baglan d
    display GUI
    for each string object s that user enters;
        query s from d
        get result
        display result
    end
end

7.1.2. “arac” PACKAGE

Preprocessor :: preprocess()
    kisimleriAyir islenmemisrapor
    foreach raporkisim:RaporKisim;
        cumleleriAyir
        for each cumle:Cumle
            kelimeleriAyir
            for each kelime:MyKelime
                check zemberek
                if kelime found in zemberek
                    kelime:MyKelime <- kelime:Zemberek.Kelime
            end
        end
    end
end

AnlamsalIliskilendir :: iliskilendir()
    for each kelime:MyKelime
        look up l1:MedicalLexicon for kelime
        look up l2:SifatLexicon for kelime
        look up l3:YuklemLexicon for kelime
        if( kelime not found on lexicons )
            kelimeler <- yakinKelimeler( kelime )
        lookup WordNet for kelimeler
        if( kelime not found on WordNet )
            ingilizcekelime <- tdkIngilizceyeCevir( kelime )
        lookupWordNET ingilizcekelime
        convert wordnet word to lexicon entry
        add new entry to lexicon
    end
public void Database::aktar()
    if isBagli
        continue
    else
        baglan

    for each member of Database class
        prepare the SQL statements
        commit SQL statements
end

public Vector<Data> Database::queryAnahtar( String keyword )
    d <- Database instance
    if d.isBagli
        continue
    else
        d.baglan

    create a vector V of Data
    for each table in Database
        prepare the SQL query statement
        ResultSet RS = commit the SQL statement
        If RS is not null
            Create a Data object D corresponding to table
            Read rows of RS
            Fill the members of D
            Add D to V
    Return V
end

public Vector<Data> Database::queryRapor ( int key )
    d <- Database instance
    if d.isBagli
        continue
    else
        d.baglan

    create a vector V of Data
    prepare the SQL statement for key search
    ResultSet RS = commit the SQL statement
    If RS is not null
        Create a Data object D corresponding to table
        Read rows of RS
        Fill the members of D
    end
    return V
end
8. FUTURE WORK

After prototype, we will deal with building a template recognition module, which will be helpful in extracting more meaningful and formatted data. We are planning to read text mining and machine learning books during the holiday. Then, in the second semester we will start coding accordingly.

On the semantic association side, we will continue to improve lexicons both with our own effort and with finding new medical dictionaries, and continue with extracting new rules for template matching.

Finally, we are planning to build a better GUI and improve our searcher module. Documentation and build testing are also among our plans.
### 9. APPENDIX

#### 9.1. GANTT CHART

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Dates</th>
<th>Start</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposal</td>
<td>All</td>
<td>28.9.07</td>
<td>05.10.07</td>
</tr>
<tr>
<td>Gathering requirements</td>
<td>All</td>
<td>30.9.07</td>
<td>02.11.07</td>
</tr>
<tr>
<td>Literature survey</td>
<td>All</td>
<td>06.10.07</td>
<td>31.10.07</td>
</tr>
<tr>
<td>Platform decisions</td>
<td>All</td>
<td>10.10.07</td>
<td>21.10.07</td>
</tr>
<tr>
<td>Milestone (Req.)</td>
<td></td>
<td>02.11.07</td>
<td></td>
</tr>
<tr>
<td><em>-Design-</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General architecture</td>
<td>All</td>
<td>05.11.07</td>
<td>20.11.07</td>
</tr>
<tr>
<td>Design of Components</td>
<td>All</td>
<td>09.11.07</td>
<td>30.11.07</td>
</tr>
<tr>
<td>Milestone (Inst. Design)</td>
<td></td>
<td>30.11.07</td>
<td></td>
</tr>
<tr>
<td><em>-Detailed Design-</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detailed architecture</td>
<td>All</td>
<td>02.12.07</td>
<td>12.12.07</td>
</tr>
<tr>
<td>User Interface design</td>
<td>Onur, Senor</td>
<td>12.12.07</td>
<td>15.12.07</td>
</tr>
<tr>
<td>Function/Class design</td>
<td>All</td>
<td>12.12.07</td>
<td>10.01.08</td>
</tr>
<tr>
<td>Database design</td>
<td>Tahir, Met</td>
<td>12.12.07</td>
<td>30.12.07</td>
</tr>
<tr>
<td>Milestone (Final Design)</td>
<td></td>
<td>11.01.08</td>
<td></td>
</tr>
<tr>
<td><em>-Research-</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLP</td>
<td>Senor, Met</td>
<td>09.10.07</td>
<td>05.01.07</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>Tahir, Onur</td>
<td>08.10.07</td>
<td>25.03.08</td>
</tr>
<tr>
<td>Ontology</td>
<td>Senor, Onur</td>
<td>18.10.07</td>
<td>17.01.06</td>
</tr>
<tr>
<td>Ext. dictionary conn.</td>
<td>Tahir</td>
<td>05.11.07</td>
<td>30.11.07</td>
</tr>
<tr>
<td>DfNIS</td>
<td>Tahir</td>
<td>19.10.07</td>
<td>06.04.06</td>
</tr>
<tr>
<td><em>-Implementation-</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text Preprocessor</td>
<td>Senor</td>
<td>15.10.07</td>
<td>27.01.08</td>
</tr>
<tr>
<td>Morphologic analyzer</td>
<td>Met</td>
<td>20.10.07</td>
<td>15.02.08</td>
</tr>
<tr>
<td>Spell checker</td>
<td>Tahir, Senor</td>
<td>02.12.07</td>
<td>01.03.08</td>
</tr>
<tr>
<td>Building lexicon</td>
<td>All</td>
<td>21.11.07</td>
<td>10.04.08</td>
</tr>
<tr>
<td>Semantic associator</td>
<td>Tahir, Onur</td>
<td>25.11.07</td>
<td>19.04.08</td>
</tr>
<tr>
<td>First learning module</td>
<td>Tahir</td>
<td>01.12.07</td>
<td>07.01.08</td>
</tr>
<tr>
<td>Template Matching</td>
<td>Tahir, Senor</td>
<td>31.12.07</td>
<td>17.03.08</td>
</tr>
<tr>
<td>User Interface</td>
<td>Onur</td>
<td>09.12.07</td>
<td>14.12.07</td>
</tr>
<tr>
<td>Database connector</td>
<td>Met, Tahir</td>
<td>09.12.07</td>
<td>20.12.07</td>
</tr>
<tr>
<td>Prepares first prototype</td>
<td>All</td>
<td>20.10.07</td>
<td>07.01.06</td>
</tr>
<tr>
<td>Prototype testing</td>
<td>All</td>
<td>07.01.06</td>
<td>10.01.08</td>
</tr>
<tr>
<td>Milestone (Prot. demo)</td>
<td></td>
<td>19.01.08</td>
<td></td>
</tr>
<tr>
<td>Second build</td>
<td>All</td>
<td>30.01.08</td>
<td>03.03.08</td>
</tr>
<tr>
<td>2nd Build Testing</td>
<td>All</td>
<td>03.01.08</td>
<td>10.03.08</td>
</tr>
<tr>
<td>Third build</td>
<td>All</td>
<td>03.03.08</td>
<td>17.03.08</td>
</tr>
<tr>
<td>3rd Build Testing</td>
<td>All</td>
<td>17.03.08</td>
<td>24.03.08</td>
</tr>
<tr>
<td>Fourth build</td>
<td>All</td>
<td>17.03.08</td>
<td>30.03.08</td>
</tr>
<tr>
<td>4th Build Testing</td>
<td>All</td>
<td>30.03.08</td>
<td>06.04.08</td>
</tr>
<tr>
<td>Integrated builds</td>
<td>All</td>
<td>30.04.08</td>
<td>17.04.08</td>
</tr>
<tr>
<td>Builds Testing</td>
<td>All</td>
<td>17.04.08</td>
<td>30.04.08</td>
</tr>
<tr>
<td><em>-Final Release-</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prepare final version</td>
<td>All</td>
<td>24.04.08</td>
<td>07.05.08</td>
</tr>
<tr>
<td>Testing &amp; Debugging</td>
<td>All</td>
<td>07.05.08</td>
<td>26.05.08</td>
</tr>
<tr>
<td>Documentation</td>
<td>All</td>
<td>24.04.09</td>
<td>27.05.08</td>
</tr>
<tr>
<td>Milestone (Final Demo)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Schaum’s Outlines Software Engineering, David Gustafson
3. Component Oriented Software Engineering, Ali H. Doğru
6. db4o, http://www.db4o.com/
11. The Text Mining Handbook, Ronen Feldman / James Sanger
14. C4.5 Programs For Machine Learning, J.Ross Quinlan