Middle East Technical University

Department of Computer Engineering

-TURKUAZ PROJECT-

TIRAN SOFTWARE
INITIAL DESIGN REPORT

RadeX

Tahir Bilal *1394741*
Onur Deniz *1391002*
Soner Kara *1395110*
Mert Karadağlı *1395128*

TABLE OF CONTENTS

1. INTRODUCTION 5

1.1. Motivation 5

1.2. Project Topic 5

1.3. Project Description 5

2. CURRENT STATUS 6

2.1. What We Have Done So Far 6

2.2. New Research 6

2.2.1. Db4o 6

2.2.2. WordNet 6

2.2.3. JWNL 7

3. DESIGN CONSTRAINTS 7

3.1. Naming and Documentation constraints 7

3.2. Time Constraints 7

3.3. User Interface Constraints 8

3.4. Performance Constraints 8

4. DATA DESIGN 8

4.1. Database Tables 8

4.1.1. Rapor 8

4.1.2. Problem 9

4.1.3. Normal_bulgu 10

4.1.4. Islem 10

4.1.5. Ilac_tedavi 11

4.1.6. Kiyas 12

4.2. Er Diagram 13

5. ARCHITECTURAL DESIGN 14

5.1. Structural Modeling 14

5.1.1. Data Flow Diagrams 14

5.1.1.1. Level-0 14

5.1.1.2. Level-1 “RadeX” 14

5.1.1.3. Level-2 “preprocess reports” 15
5.1.1.4. Level-2 “associate semantics” 15
5.1.1.5. Level-2 “learn and extract” 16
5.1.2. Data Dictionary 17
5.2. Moduler Hiearchy 18
5.3. Module Explanations 19
5.3.1. Report Analyzer Component 19
5.3.1.1. PREPROCESSOR MODULE 19
5.3.1.2. SEMANTIC ASSOCIATOR MODULE 19
5.3.1.3. LEARNER 20
5.3.1.4. FINALIZER 20
5.3.1.5. SPELL CHECKER 20
5.3.2. Database Searcher Component 21
5.3.2.1. Database Connector 21
5.3.2.2. ResultSet Reader 21
5.3.2.3. Displayer 21
5.4. Class Modeling 21
5.4.1. Package/Class Descriptions & Class Diagrams 21
5.4.1.1. “anlam” package 21
5.4.1.2. “arac” package 23
5.4.1.3. “database” package 24
5.4.1.4. “lexicon” package 25
5.4.1.5. “malumat” package 26
5.4.1.6. “radex” package 26
5.4.1.7. “yapi” package 26
5.4.1.8. “gui” package 29
5.5. Behavioral Modeling 30
5.5.1. Sequence Diagrams 30
5.5.1.1. “Analist” Sequence Diagram 30
5.5.1.2. “Sorgucu” Sequence Diagram 31
5.5.1.3. “Preprocessor” Sequence Diagram 32
5.5.1.4. “AnlamsalIliskilendir” Sequence Diagram 33
5.5.2. Activity Diagrams 34
6. INTERFACE DESIGN 35
6.2. User Interface Design 35

6.3. Method/Class Interfaces 37
6.3.1. “arac” package 37
6.3.2. “database” package 40
6.3.3. “lexicon” package 44
6.3.4. “radex” package 45
6.3.5. “yapi” package 45
7. PROCEDURAL DESIGN 46
7.1. PseudoCodes 46
7.1.1. “radex” package 46
7.1.2. “arac” package 46
7.1.3. “database” package 47
8. FUTURE WORK 48
9. APPENDIX 49
9.1. Gantt Chart 49
10. REFERENCES 50

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

Since the proposal date till now, we have done a lot of research about the topic, namely text mining, machine learning and named entity recognition. Also we have done some progress on using helper tools such as Zemberek.

Besides researching, we have also started implementation of preprocessor module. In fact, we have released our first version, which is capable of extracting some basic information by looking meanings of certain common verbs. Moreover, we have managed to connect TDK, query it, and use it together with WordNet.

During this time, the architecture of the project became clearer. But there is still uncertainty on learning module, which is actually the heart of the program. Currently we are examining open-source information extraction programs that use machine learning methods. To sum up, we are trying to make a good design of the project besides basic implementations and research.

2.2. NEW 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].

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 ANNLearner, NERecognizer, 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 CONSTARINTS

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.

Rapor_no	Integer
Baslik	String
Tarih	Date
Doktorlar	Varchar(40)
Hasta	Varchar(20)
PRIMARY KEY (Rapor_no)	
INDEX(Rapor_no)	

- *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.
- *Doktor* 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	Integer
Rapor_no	Integer
Rapor_bolum	Varchar(15)
Problem	Varchar(20)
Bolge	Varchar(20)
Alt_bolge	Varchar(20)
Derece	Varchar(20)
Kesinlik	Varchar(20)
Aciklayicilar	Varchar(40)
Tespit_tarih	Date
PRIMARY KEY(Problem_no)	
FOREIGN KEY(Rapor_no) REFERENCES(Rapor)	
INDEX(Rapor_no)	
CONSTRAINT NOT NULL(Rapor_no), NOT NULL(Rapor_Bolum)	

- *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'.
- *Kesinlik* field holds the certainty of the assessment of the actuality of the problem.
- *Derece* field holds the severity of the problem.
- *Aciklayacilar* 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'.

Bulgu_no	Integer
Rapor_no	Integer
Rapor_bolum	Varchar(20)
Bulgu	Varchar(20)
Nitelik	Varchar(20)
Bolge	Varchar(20)
Kesinlik	Varchar(20)
Tespit_tarih	Date
PRIMARY KEY(Bulgu_no)	
FOREIGN KEY(Rapor_no) REFERENCES(Rapor)	
INDEX(Rapor_no)	
CONSTRAINT NOT NULL(Rapor_no), NOT NULL(Rapor_Bolum)	

- *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.

Islem_no	Integer
Rapor_no	Integer
Rapor_bolum	Varchar(20)
Islem_adi	Varchar(20)
Baslangic_tarih	Date
Sure	Integer
Bolge	Varchar(20)
Aygit	Varchar(20)
Aciklama	Varchar(30)
OneriMi	Boolean
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 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.

Ilac_tedavi_no	Integer
Rapor_no	Integer
Rapor_bolum	Varchar(20)
Ilac	Varchar(20)
Bolge	Varchar(20)
Baslangic_tarih	Date
Sure	Integer
OneriMi	Boolean
PRIMARY KEY(Ilac_tedavi_no)	
FOREIGN KEY(Rapor_no) REFERENCES(Rapor)	
INDEX(Rapor_no)	
CONSTRAINT NOT NULL(Rapor_no), NOT NULL(Rapor_Bolum)	

- *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.
- *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.
- *OneriMi* field holds if this operation is a suggestion or not.

4.1.6. KIYAS

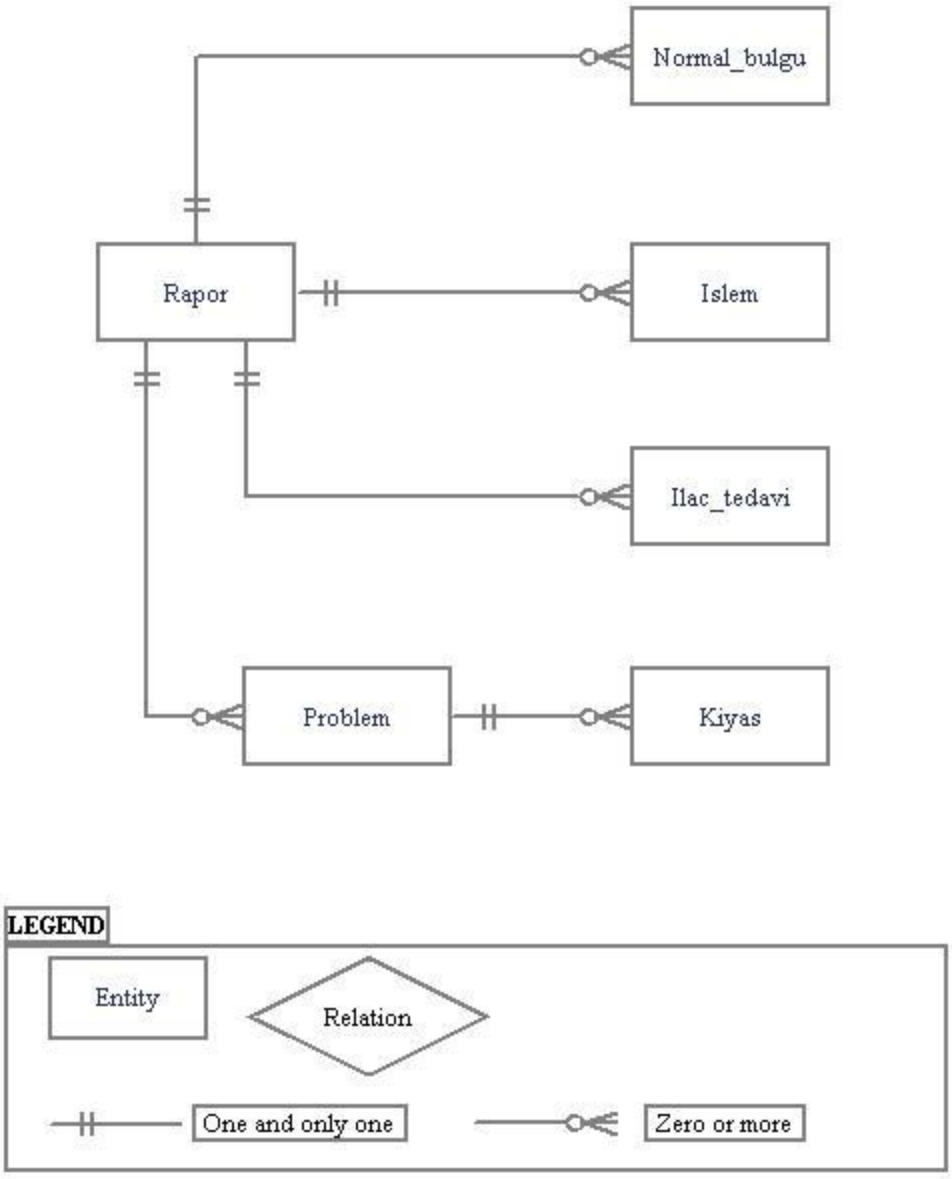
This table stores the comparison of a problem to an early phase. Not much of the problems have comparison information associated with them, so holding this information in another table hopefully will save some space.

Kiyas_no	Integer
Problem_no	Integer
Kiyas_tarih	Date
Degisiklik	Varchar(30)
PRIMARY KEY(Kiyas_no)	
FOREIGN KEY(Problem_no) REFERENCES(Problem)	
INDEX(Problem_no)	
CONSTRAINT NOT NULL(Problem_no)	

- *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.



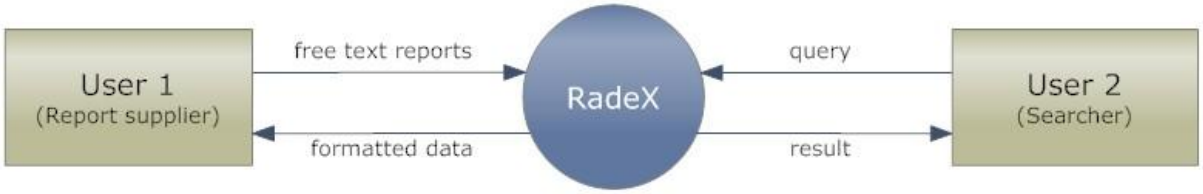
All the types except 'Rapor' should have all their instances contained by another type. This inference shows that, actually the four remaining tables may be modeled as weak entity sets. However, the fields in the other tables do not have key-like characteristics. To illustrate, we can't introduce a key 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

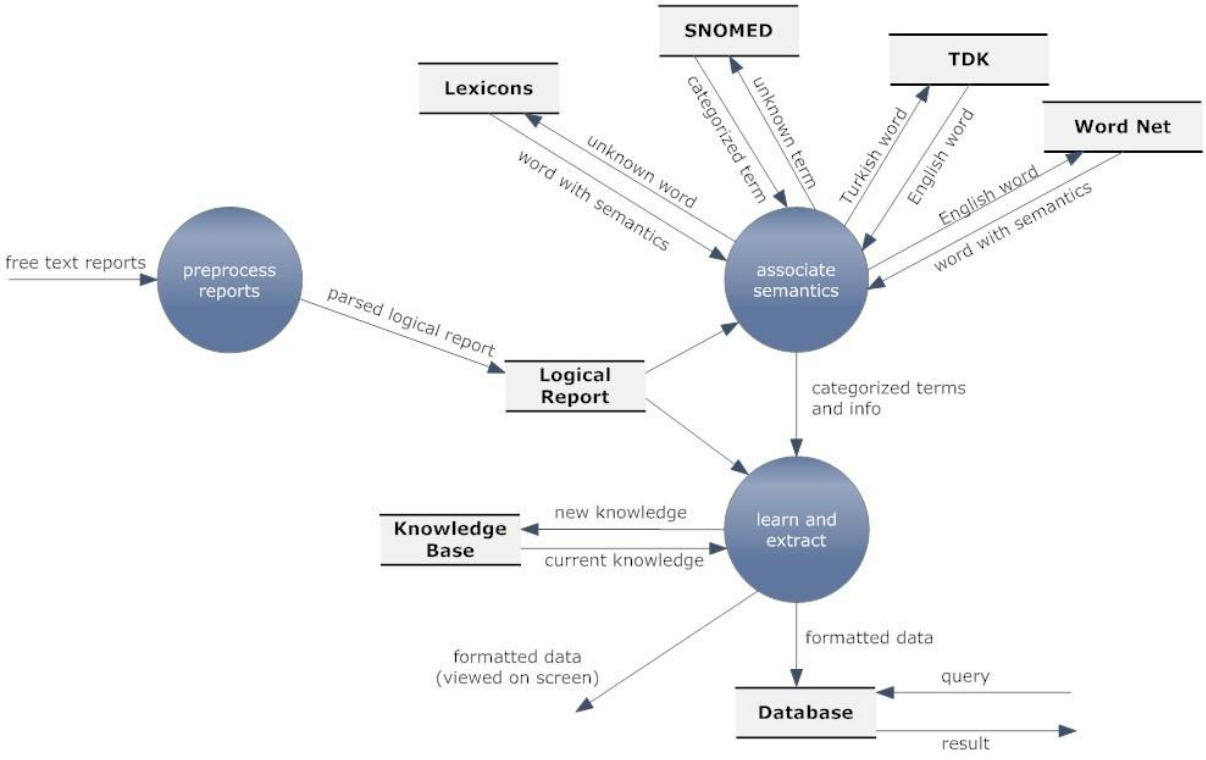
5.1. STRUCTURAL MODELING

5.1.1. DATA FLOW DIAGRAMS

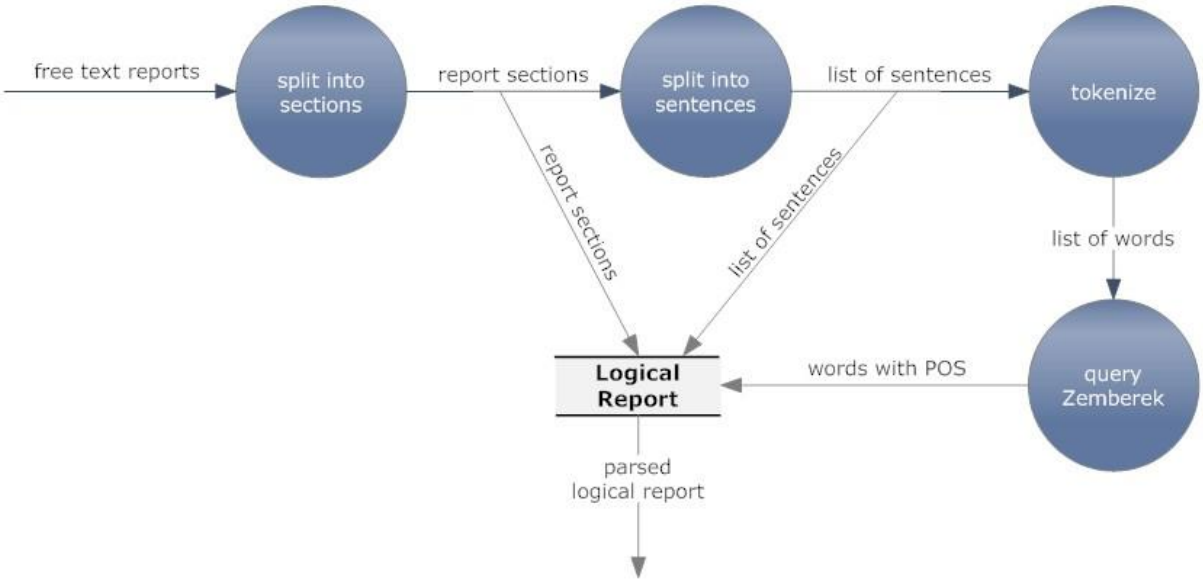
5.1.1.1. Level-0



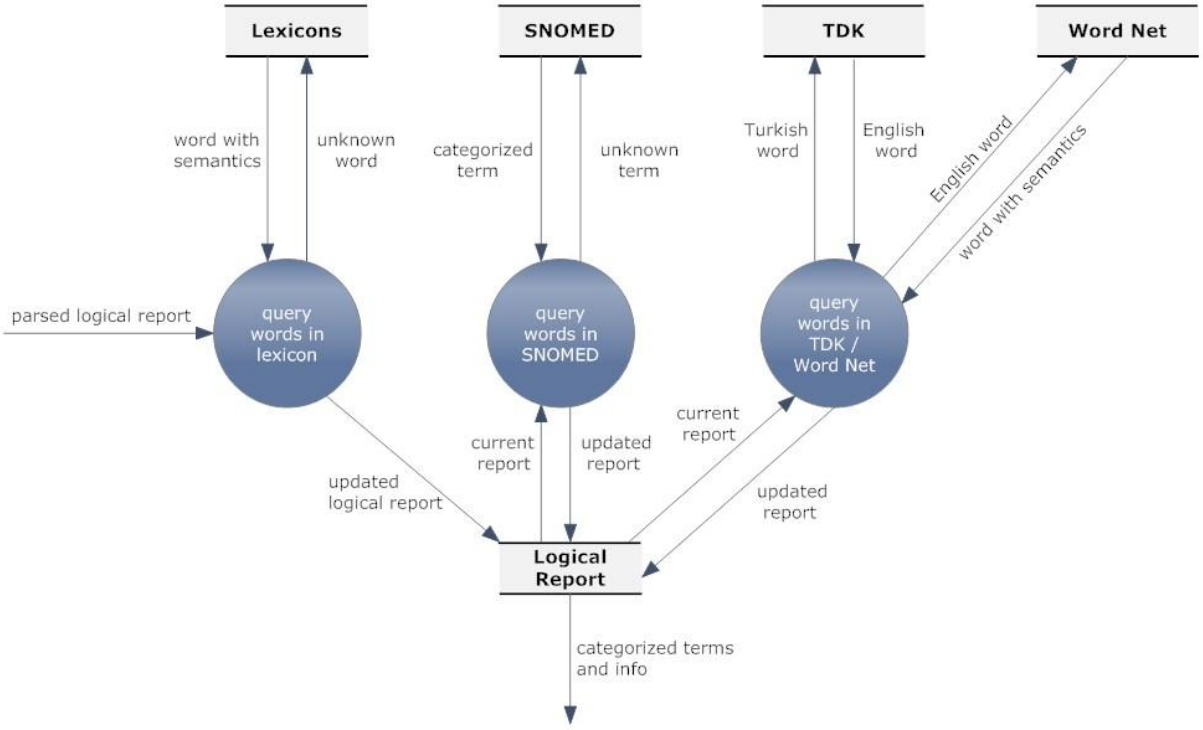
5.1.1.2. Level-1 "RadeX"



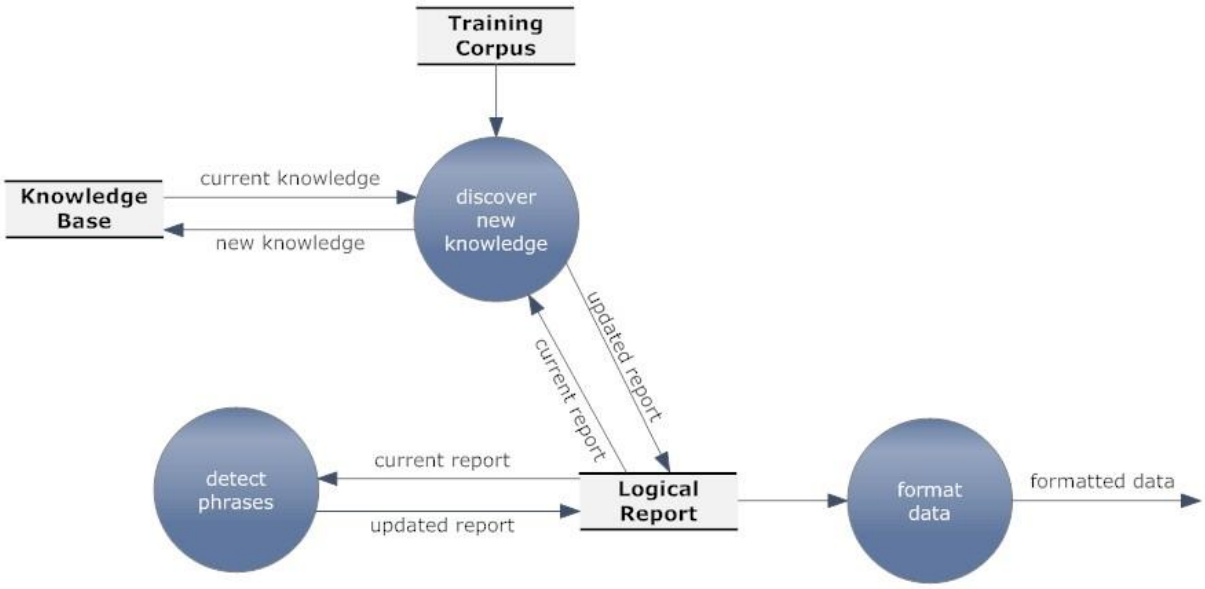
5.1.1.3. Level-2 “preprocess reports”



5.1.1.4. Level-2 “associate semantics”



5.1.1.5. Level-2 “learn and extract”



Explanations for data flow diagrams are omitted, since we already described most of them in the analysis report. But there are some important additions and modifications that need some attention.

First of all, we added Logical Report object in the new diagrams. This object is just the parsed form of the physical report in the memory. Almost every module accesses this object.

Second major change is the addition of the Learner lvl-2 flow diagram, although it is not much detailed right now. But, we are going to detail this part as we progress.

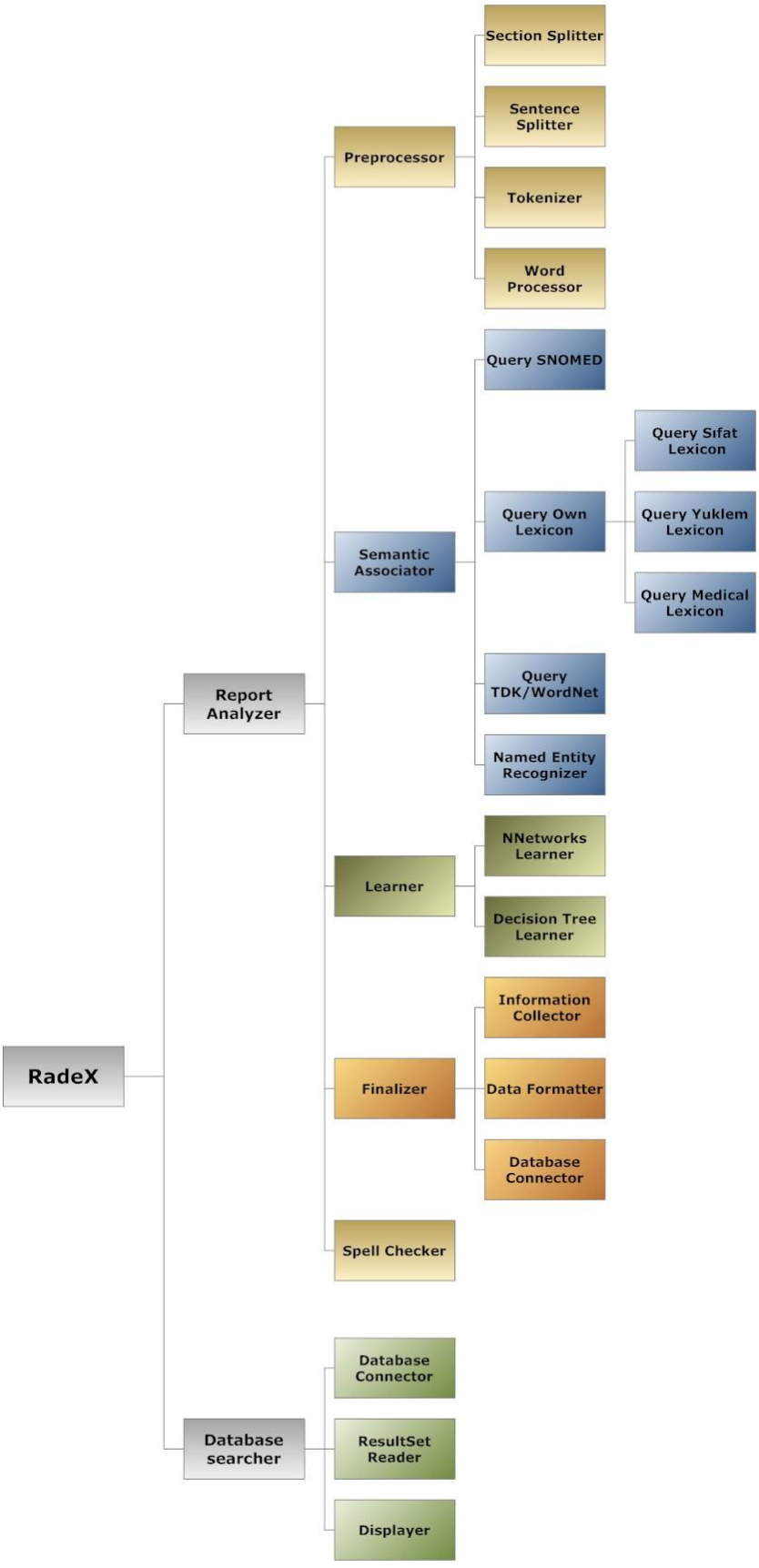
Finally, all of the data objects flowing throughout these diagrams are explained below, in 5.1.2 Data Dictionary.

5.1.2. DATA DICTIONARY

Data Dictionary for Data Flow Diagrams (LVL 0-1-2)

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

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 with the POS, kok and ekler fields filled. 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.

Neural Networks Learner
This module will use neural networks algorithms to extract new information.

Decision Tree Learner
This module will use decision trees 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.

5.3.2. DATABASE SEARCHER COMPONENT

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, yapi.*

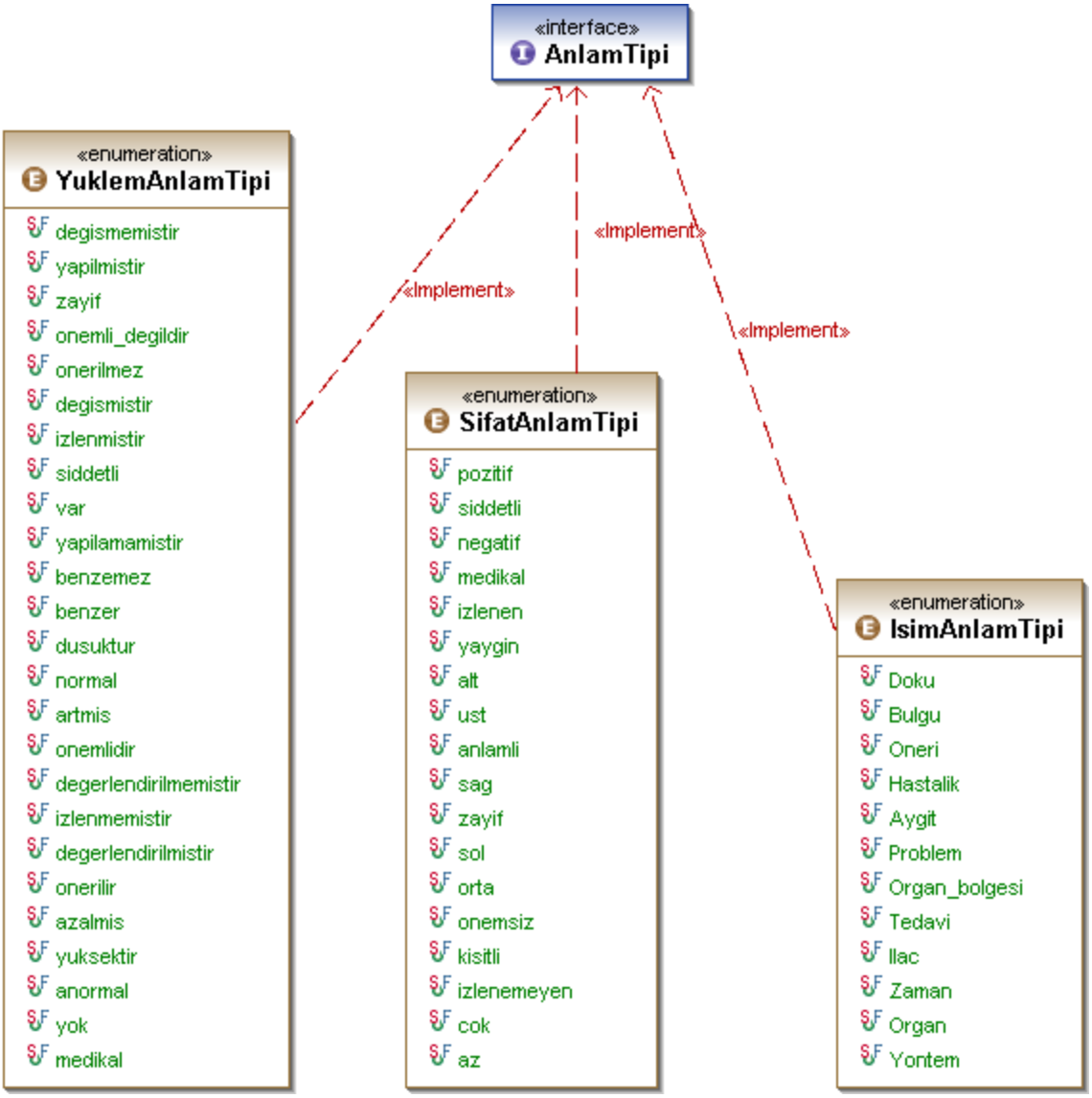
For a legend of class diagrams

- 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 no methods: AnlamTipi, and four enumarations 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 ***sense***s.

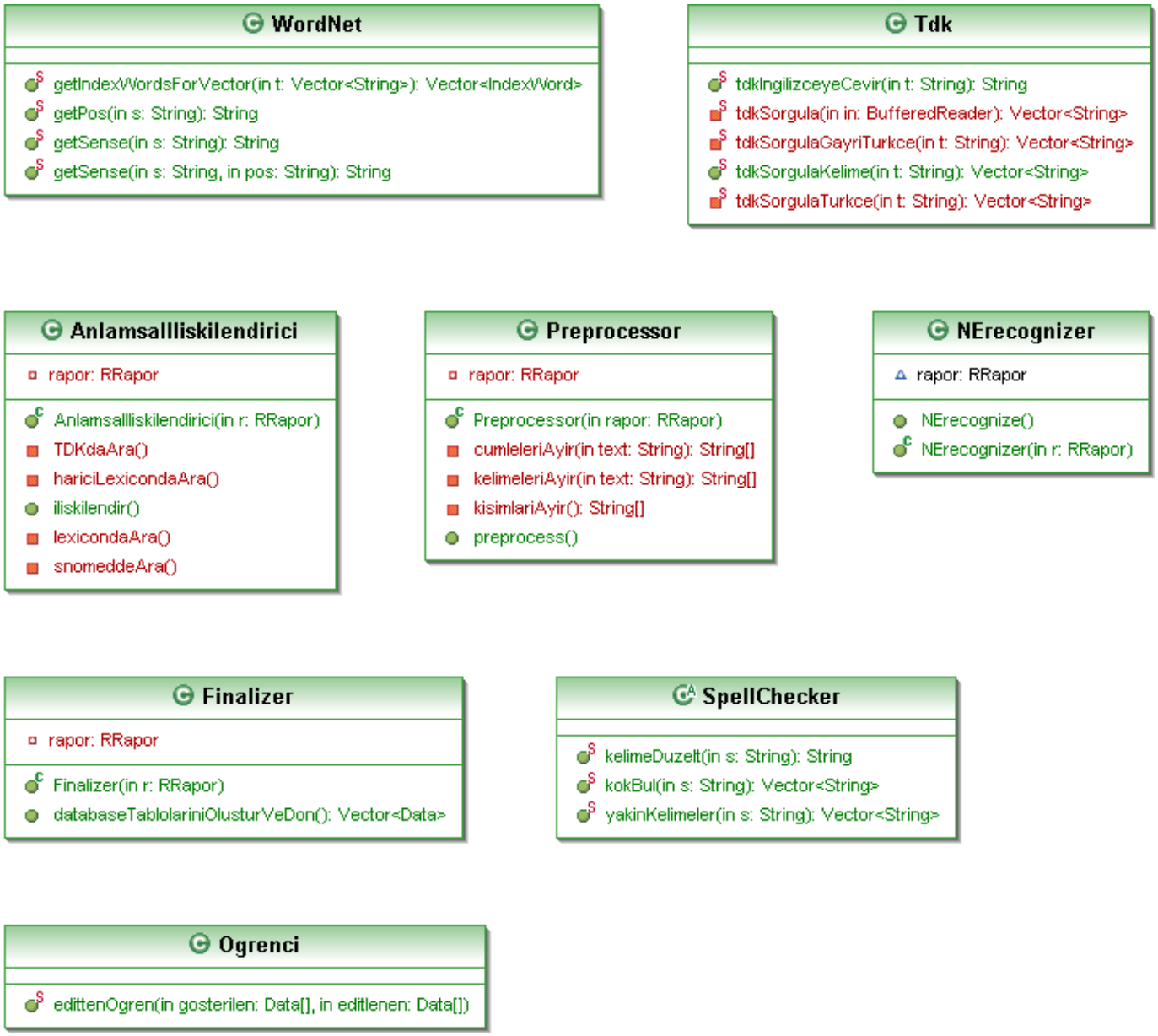
Yet our feature selection procedure is not finished. The one we have made for adjectives (SifatAnlamTipi) is premature, it is just a sketch. We also will dive into the same procedure for adverbs and connectives. YuklemAnlamTipi is to store the meanings of predicates. Predicates in the reports are usually 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.


«interface»
AnlamTipi
«enumeration»
YuklemAnlamTipi
degismemistir
yapilmistir
zayif
onemli_degildir
onerilmez
degismistir
izlenmistir
siddetli
var
yapilamamistir
benzemez
benzer
dusuktur
normal
artmis
onemlidir
degerlendirilmemistir
izlenmemistir
degerlendirilmistir
onerilir
azalmis
yuksektir
anormal
yok
medikal
«Implement»
«Implement»
«Implement»
«enumeration»
SifatAnlamTipi
pozitif
siddetli
negatif
medikal
izlenen
yaygin
alt
ust
anlamli
sag
zayif
sol
orta
onemsiz
kisitli
izlenemeyen
cok
az
«enumeration»
IsimAnlamTipi
Doku
Bulgu
Oneri
Hastalik
Aygit
Problem
Organ_bolgesi
Tedavi
Ilac
Zaman
Organ
Yontem

5.4.1.2. "arac" package

This package contains the all tool functioned classes that we will implement in our program, namely Anlamsallliskilendirici (Semantic Associator), Ogrenci (Learner), Preprocessor, SpellChecker, Tdk, Finalizer, NERecognizer, WordNet. The methods of Ogrenci apart from one are not there yet.

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.

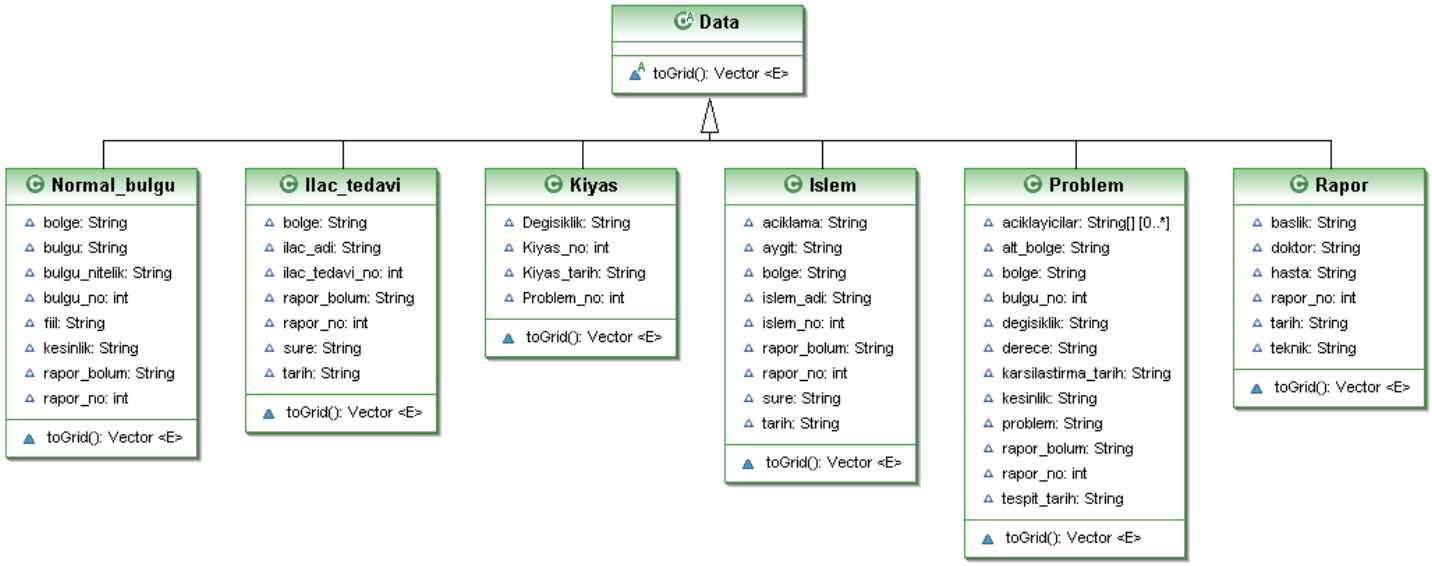


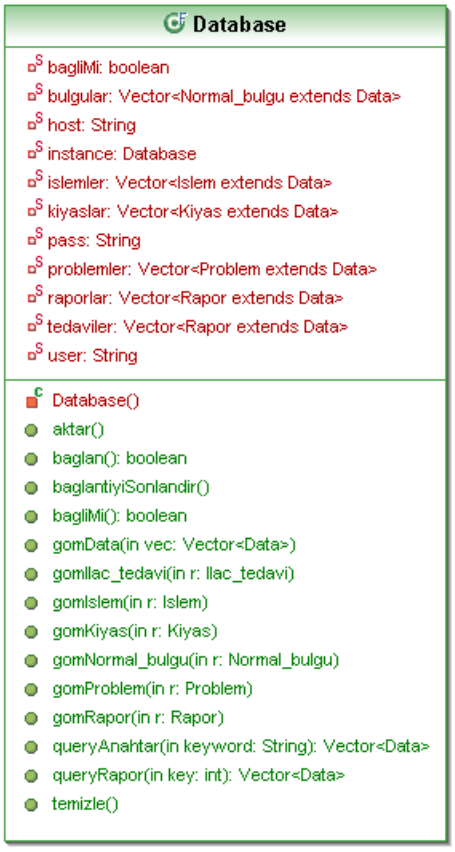
5.4.1.3. "database" package

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.

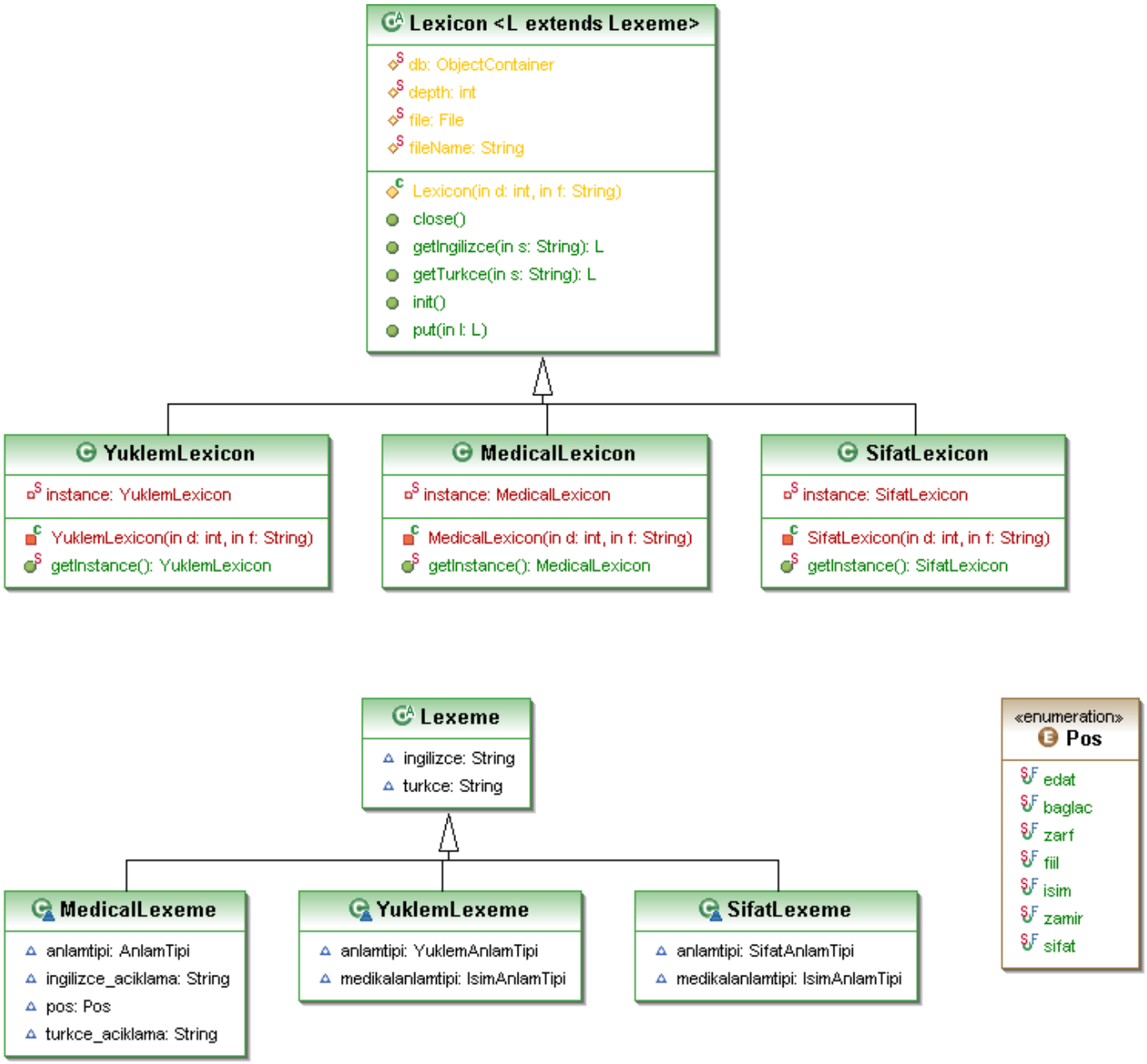
Database class is to abstract the database related functionalities of radex. Class diagrams for the database package are below.





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 dbo.ObjectContainter property named db. That property holds the 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.
- MedicalLexeme, YuklemLexeme, SifatLexeme are the classes that the lexicons store the instances of.

- *Lexeme* means a vocabulary entry in English, and doesn't have a Turkish counterpart. 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.

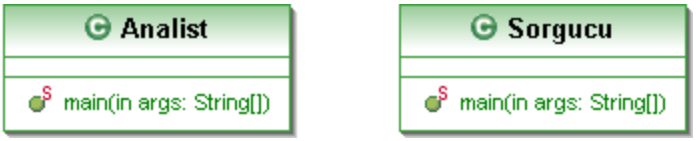
5.4.1.5. "malumat" package

This package is supposed to hold the knowledge bases that we will use. It will also be using db4o just like lexicons do. Unfortunately, the content of this package is not available at the moment, surely ☺ will be available at the final design report.

5.4.1.6. "radex" package

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

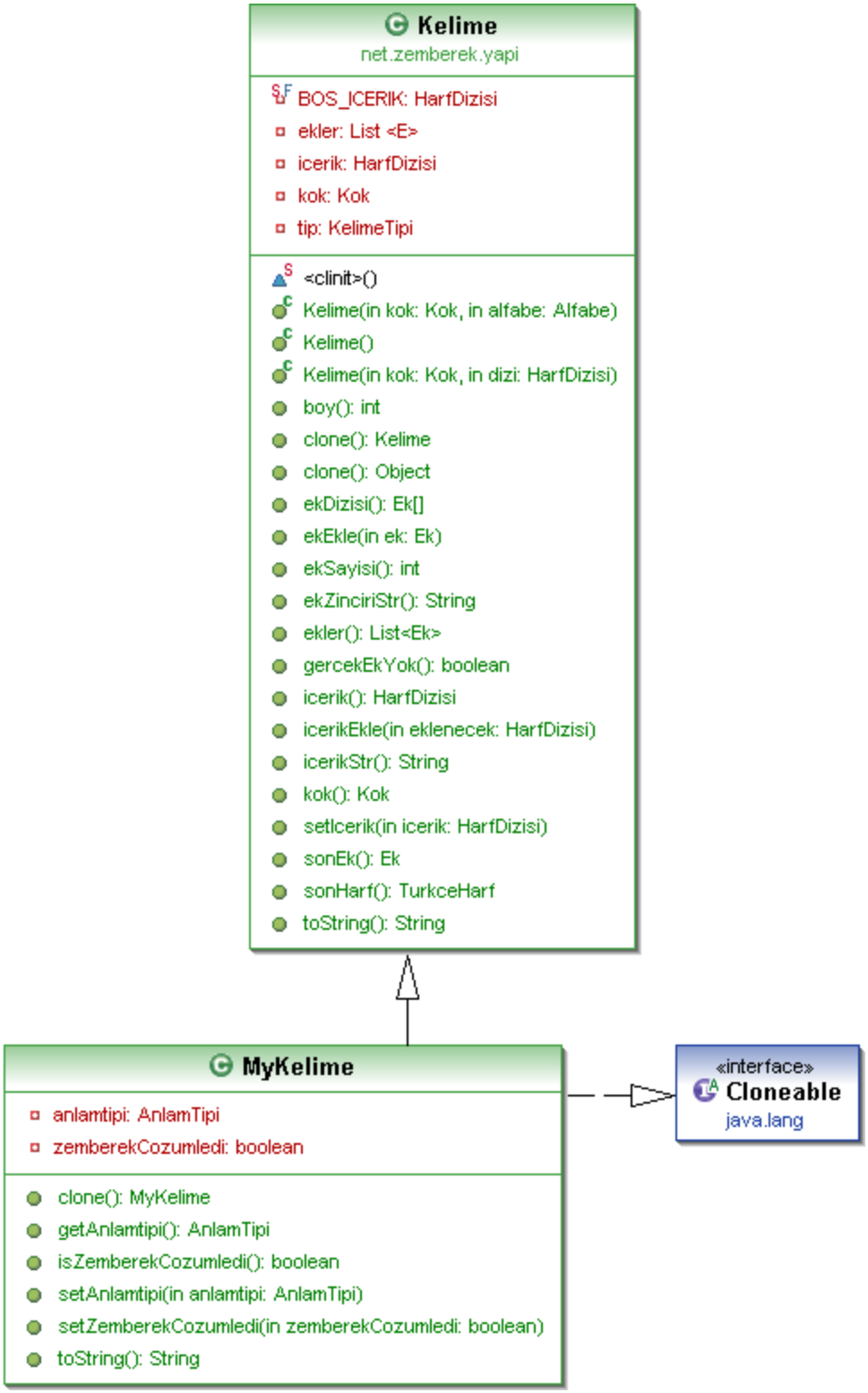
At the moment all these two classes have is the main method.

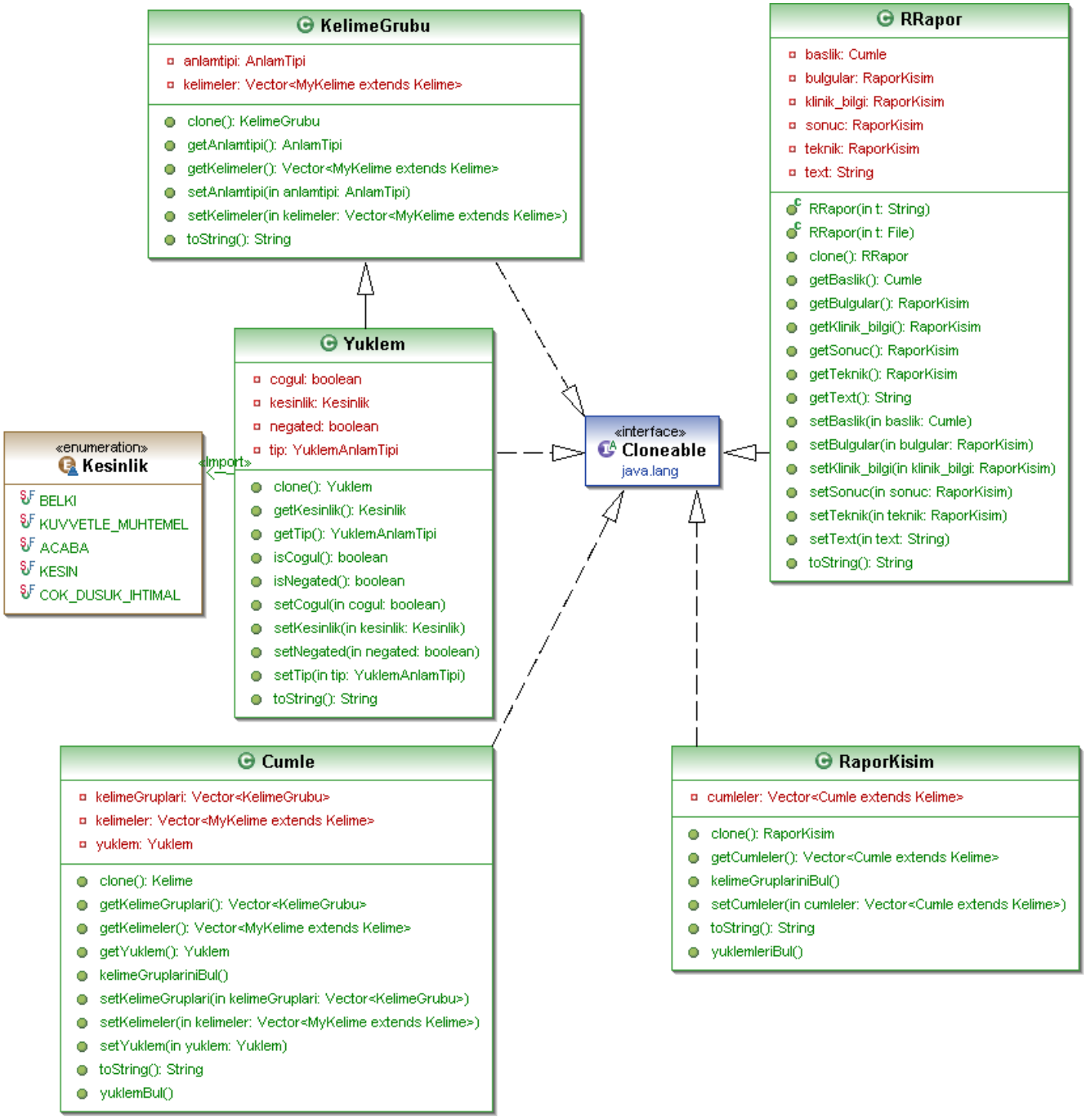


5.4.1.7. "yapi" package

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.

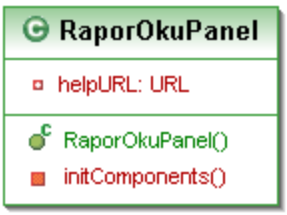
- Kelime is not a class that we created, it is part of Zemberek. Our own class is named MyKelime which extends Kelime
- KelimeGrubu stands for Noun Phrase.
- Cumle stands for sentence.
- Yuklem stands for predicate.
- Kesinlik stands for exactness.
- RRapor is the class that holds all the information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


Kelime
net.zemberek.yapi
BOS_ICERIK: HarfDizisi
ekler: List <E>
icerik: HarfDizisi
kok: Kok
tip: KelimeTipi
<clinit>()
Kelime(in kok: Kok, in alfabe: Alfabe)
Kelime()
Kelime(in kok: Kok, in dizi: HarfDizisi)
boy(): int
clone(): Kelime
clone(): Object
ekDizisi(): Ek[]
ekEkle(in ek: Ek)
ekSayisi(): int
ekZinciriStr(): String
ekler(): List<Ek>
gercekEkYok(): boolean
icerik(): HarfDizisi
icerikEkle(in eklenecek: HarfDizisi)
icerikStr(): String
kok(): Kok
setIcerik(in icerik: HarfDizisi)
sonEk(): Ek
sonHarf(): TurkceHarf
toString(): String
MyKelime
anlamtipi: AnlamTipi
zemberekCozumledi: boolean
clone(): MyKelime
getAnlamtipi(): AnlamTipi
isZemberekCozumledi(): boolean
setAnlamtipi(in anlamtipi: AnlamTipi)
setZemberekCozumledi(in zemberekCozumledi: boolean)
toString(): String
«interface»
Cloneable
java.lang


KelimeGrubu
anlamtipi: AnlamTipi
kelimeler: Vector<MyKelime extends Kelime>
clone(): KelimeGrubu
getAnlamtipi(): AnlamTipi
getKelimeler(): Vector<MyKelime extends Kelime>
setAnlamtipi(in anlamtipi: AnlamTipi)
setKelimeler(in kelimeler: Vector<MyKelime extends Kelime>)
toString(): String
RRapor
baslik: Cumle
bulgular: RaporKisim
klinik_bilgi: RaporKisim
sonuc: RaporKisim
teknik: RaporKisim
text: String
RRapor(in t: String)
RRapor(in t: File)
clone(): RRapor
getBaslik(): Cumle
getBulgular(): RaporKisim
getKlinik_bilgi(): RaporKisim
getSonuc(): RaporKisim
getTeknik(): RaporKisim
getText(): String
setBaslik(in baslik: Cumle)
setBulgular(in bulgular: RaporKisim)
setKlinik_bilgi(in klinik_bilgi: RaporKisim)
setSonuc(in sonuc: RaporKisim)
setTeknik(in teknik: RaporKisim)
setText(in text: String)
toString(): String
Yuklem
cogul: boolean
kesinlik: Kesinlik
negated: boolean
tip: YuklemAnlamTipi
clone(): Yuklem
getKesinlik(): Kesinlik
getTip(): YuklemAnlamTipi
isCogul(): boolean
isNegated(): boolean
setCogul(in cogul: boolean)
setKesinlik(in kesinlik: Kesinlik)
setNegated(in negated: boolean)
setTip(in tip: YuklemAnlamTipi)
toString(): String
«enumeration»
Kesinlik
BELKI
KUVVETLE_MUHTEMEL
ACABA
KESIN
COK_DUSUK_IHTIMAL
«import»
«interface»
Cloneable
java.lang
Cumle
kelimeGruplari: Vector<KelimeGrubu>
kelimeler: Vector<MyKelime extends Kelime>
yuklem: Yuklem
clone(): Kelime
getKelimeGruplari(): Vector<KelimeGrubu>
getKelimeler(): Vector<MyKelime extends Kelime>
getYuklem(): Yuklem
kelimeGruplariniBul()
setKelimeGruplari(in kelimeGruplari: Vector<KelimeGrubu>)
setKelimeler(in kelimeler: Vector<MyKelime extends Kelime>)
setYuklem(in yuklem: Yuklem)
toString(): String
yuklemBul()
RaporKisim
cumleler: Vector<Cumle extends Kelime>
clone(): RaporKisim
getCumleler(): Vector<Cumle extends Kelime>
kelimeGruplariniBul()
setCumleler(in cumleler: Vector<Cumle extends Kelime>)
toString(): String
yuklemleriBul()

5.4.1.8. "gui" package

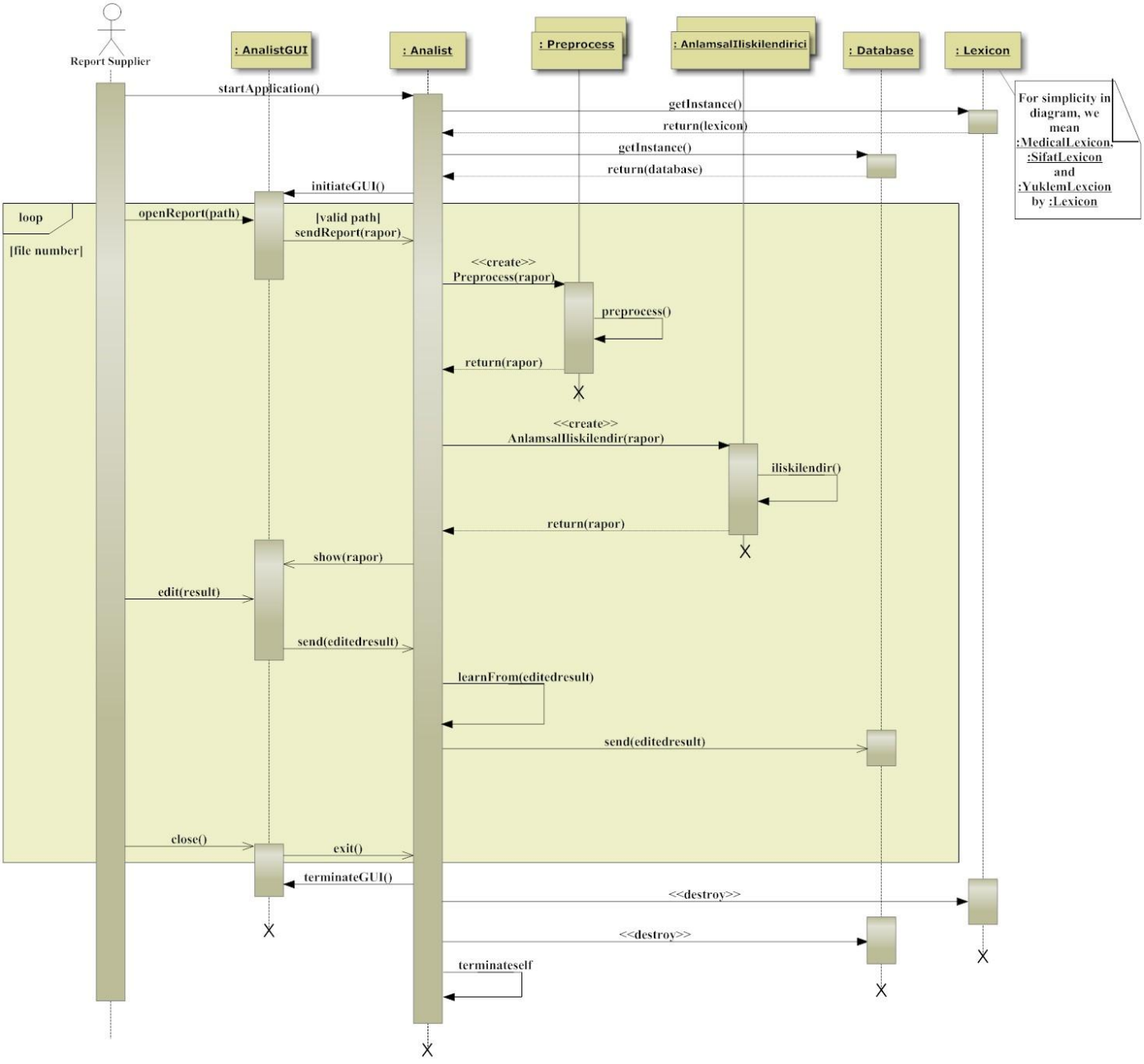
This package stores the classes related to our graphical user interfaces.



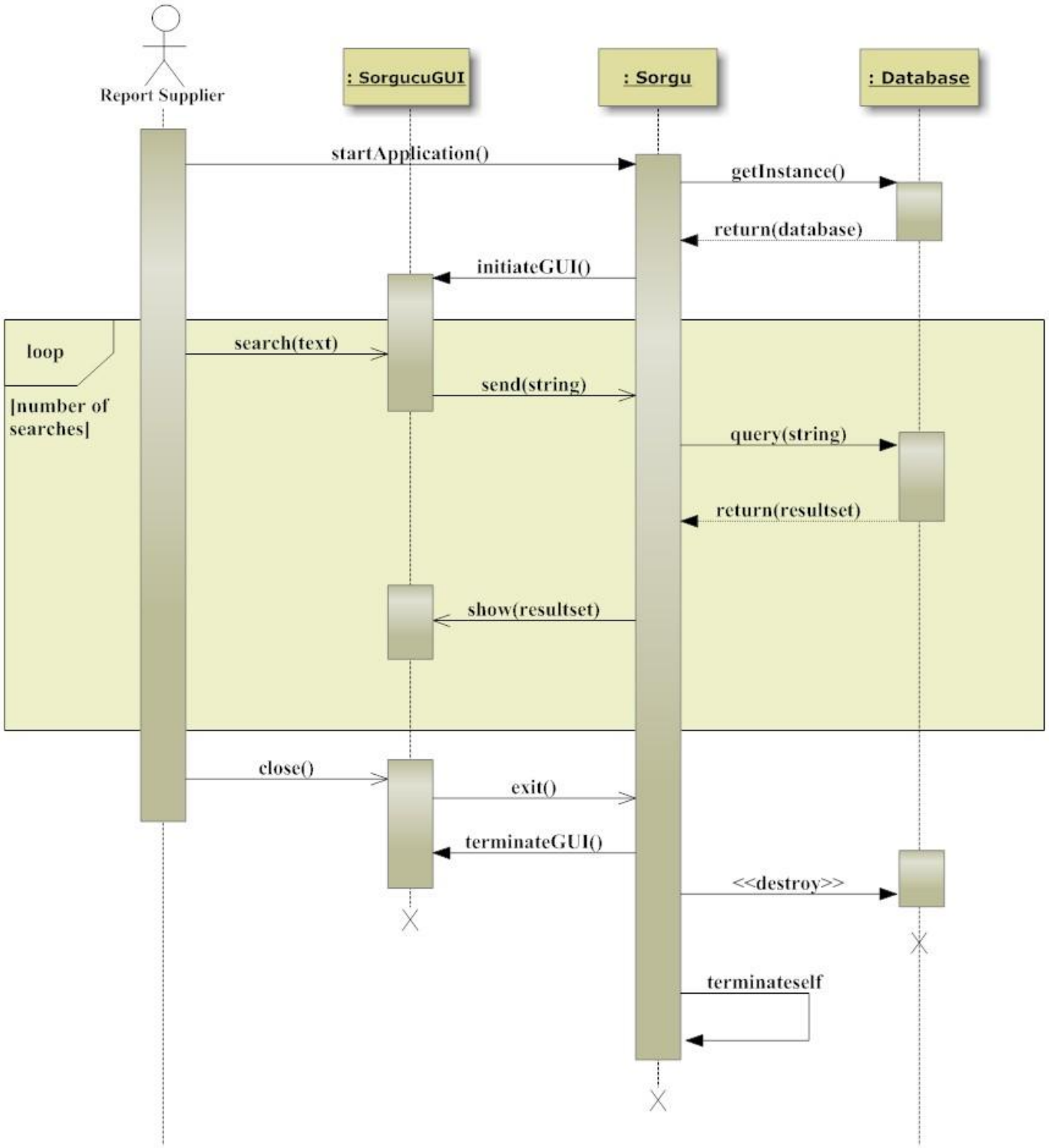
5.5. BEHAVIORAL MODELING

5.5.1. SEQUENCE DIAGRAMS

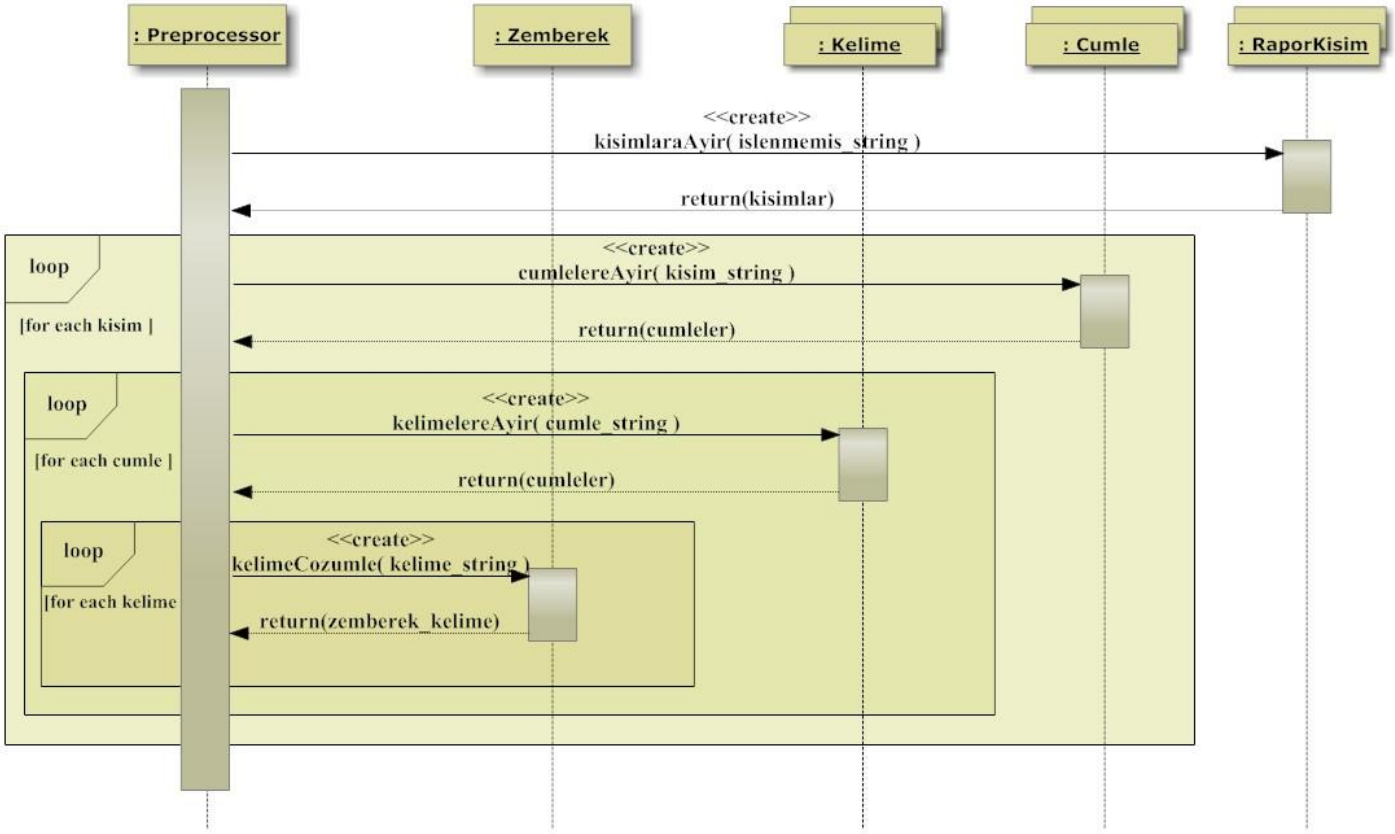
5.5.1.1. "Analist" Sequence Diagram



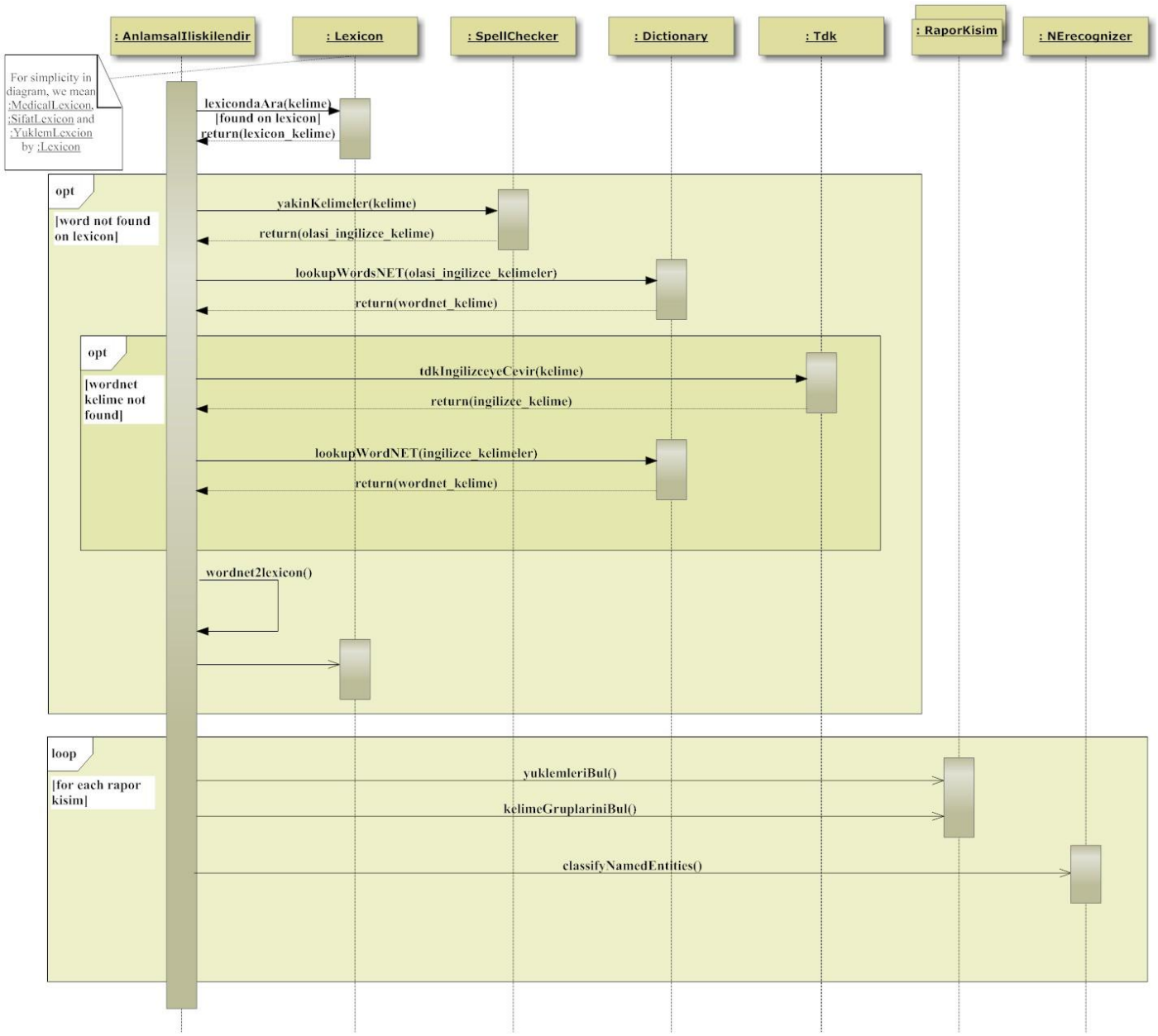
5.5.1.2. "Sorgucu" Sequence Diagram



5.5.1.3. "Preprocessor" Sequence Diagram

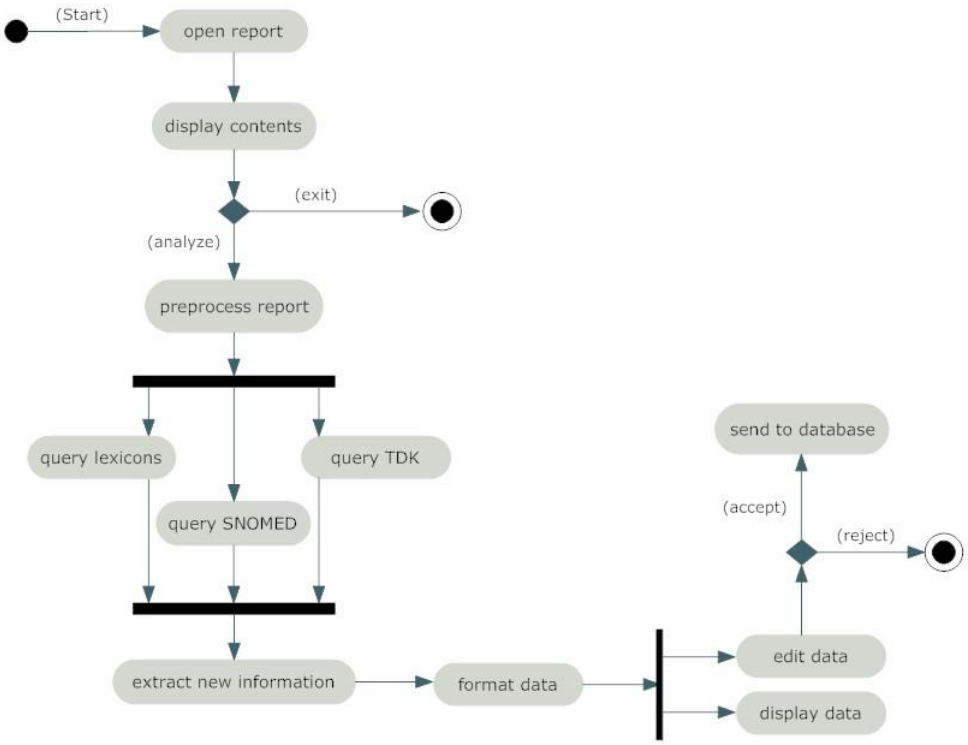


5.5.1.4. "AnlamsalIliskilendir" Sequence Diagram

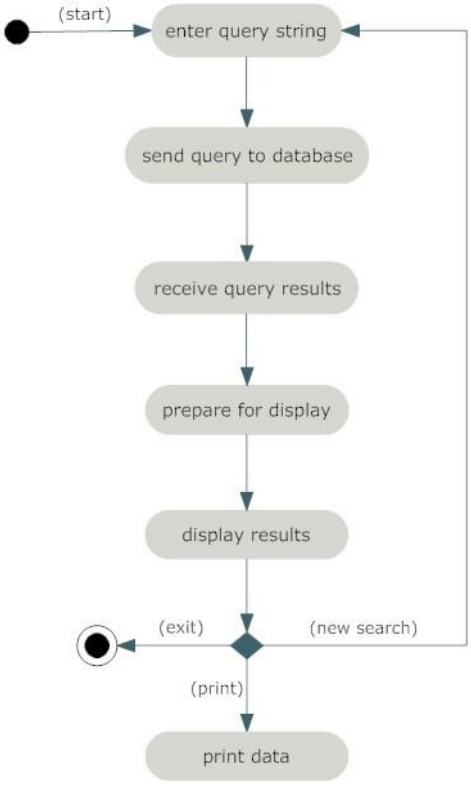


5.5.2. ACTIVITY DIAGRAMS

Activity Diagram for "Report Analyzer"



Activity Diagram for "Searcher"



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.
- *Open*ing 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.

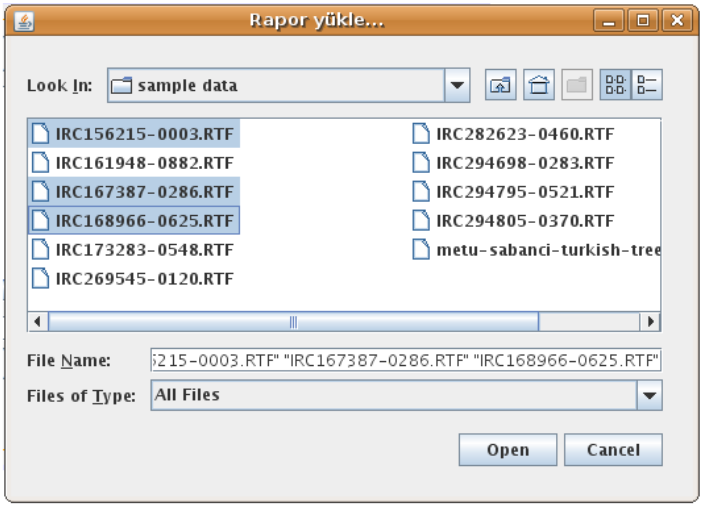


Figure A

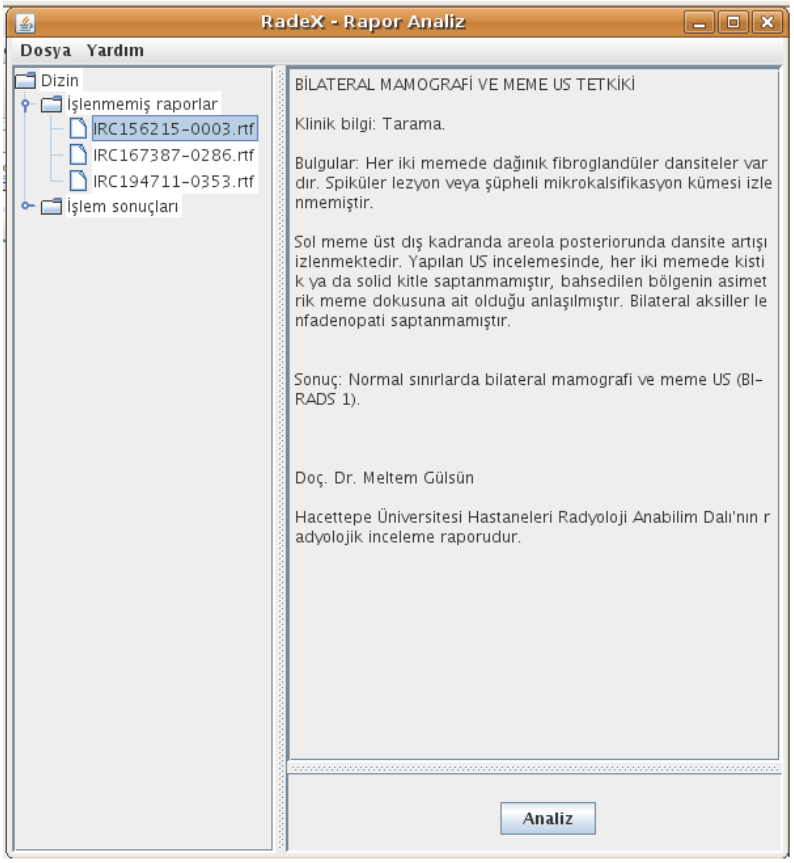


Figure B

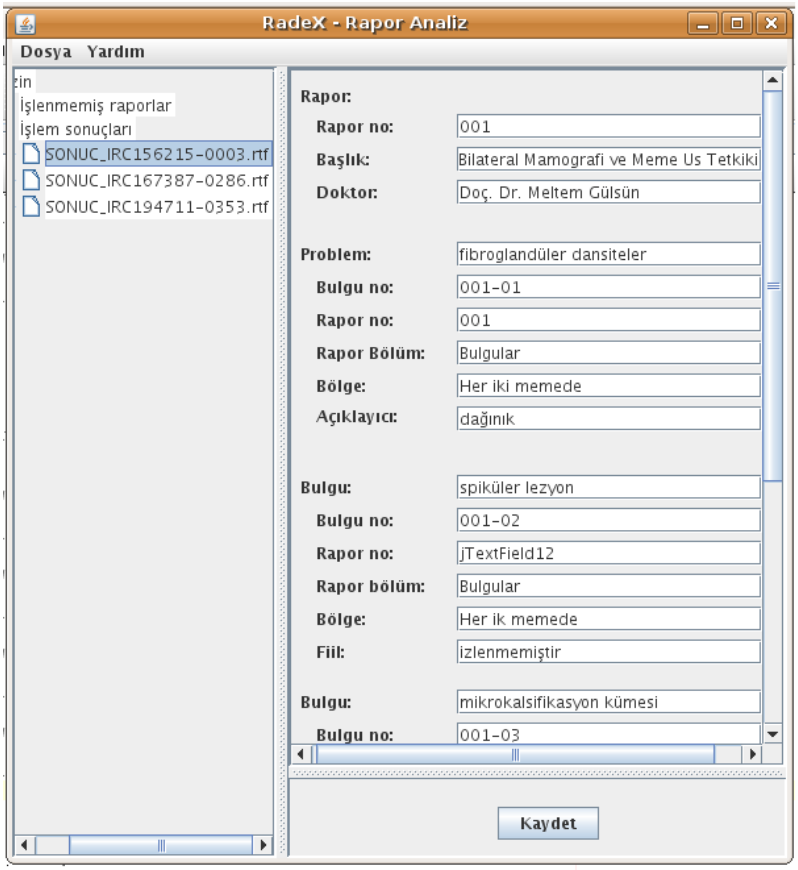


Figure C

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.

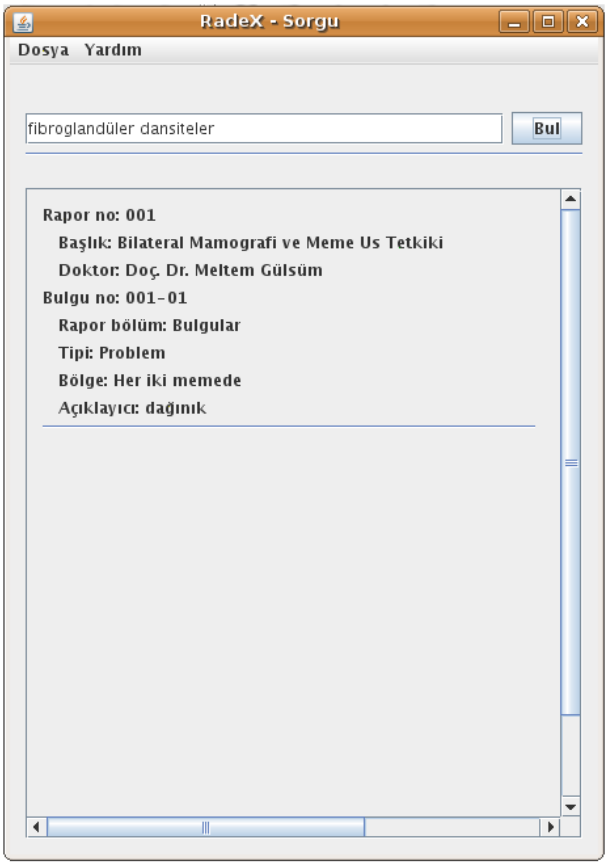


Figure D

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 :: yakinKelimeler

- **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 Preprocesor :: Preprocessor

- **input parameters :** *r* – yapi.RRapor

- Preprocessor constructor takes a RRapor object *r* as a single argument and initialize its *rapor* field with *r*

method Preprocessor :: preprocess

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

- This method does preprocessing of the report *rapor*.

method AnlamsalIliskilendirici :: AnlamsalIliskilendirici

- **input parameters :** *r* – yapi.RRapor

- AnlamsalIliskilendirici constructor takes a RRapor object *r* as a single argument and initialize its *rapor* field with *r*

method AnlamsalIliskilendirici :: iliskilendir

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

- This method does semantic association of the report *rapor.*

method Finalizer :: Finalizer

- **input parameters :** *r* – yapi.RRapor

- Finalyzer constructor takes a RRapor object *r* as a single argument and initialize its *rapor* field with *r*

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 NErecognizer :: NErecognizer

- **input parameters :** *r* **-** RRapor

- NErecongnizer constructor takes a RRApor object *r* as its only constructor.

method NErecognizer :: NErecognize

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

- This method tries to find out the *sense*s of the word phrases

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 WordNet :: getSense

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

- This method returns the sense of the English words in vector *s* using Word Net.

6.3.2. "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 .

static method Database :: getInstance

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

- This method returns the only instance of the Database class

method Database :: setHost

- **input parameters** : *h* – String
- **return type** : none

- This method sets the host address of the physical database to be *h.*

method Database :: setUser

- **input parameters** : *u* – String
- **return type** : none

- This method sets the username for the connection to the physical database to *u.*

method Database :: setPass

- **input parameters** : *p* – String
- **return type** : none

- This method sets the password of the connection to the physical database to be *p.*

method Database :: bagliMi

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

- This method returns if the physical database connection is active.

method Database :: baglan

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

- This method tries to open the connection to the host of the database.
- Returns
  - ***true*** on success.
  - ***false*** on failure.

method Database :: baglantiyiSonladir

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

- This method closes the connection to the host of the database.

method Database :: gomRapor

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

- This method inserts the Rapor *i* to the vector holding the Rapor objects in this Database class.
- Beware that this method doesn't insert *i* to the physical database directly, just stores it in the Database class.

method Database :: gomIslem

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

- This method inserts the Islem *i* to the vector holding the Islem objects in this Database class.
- Beware …

method Database :: gomNormal_bulgu

- **input parameters** : *i* – Normal_bulgu
- **return type** : None

- This method inserts the Normal_bulgu *i* to the vector holding the Normal_bulgu objects in this Database class.
- Beware …

method Database :: gomProblem

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

- This method inserts the Problem *i* to the vector holding the Problem objects in this Database class.
- Beware …

method Database :: gomKiyas

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

- This method inserts the Kiyas the vector holding the Kiyas objects in this Database class.
- Beware …

method Database :: gomIlac_tedavi

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

- This method inserts the Islem table *i* to the vector holding the 'Islem's in this Database class.
- Beware …

method Database :: gomIlac_tedavi

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

- This method inserts the Islem table *i* to the vector holding the 'Islem's in this Database class.
- Beware …

method Database :: gomData

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

- This method inserts the all the element *Data*s 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** : 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<Data>

- 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 :: queryAnahtar

- **input parameters** : *key* - int
- **return type** : Vector<Data>

- This method queries the physical database for all the Data having some field containing the keyword *key.* Next it returns the vector containing all the query resulted Data.

6.3.3. "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 exists 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 exists 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.)

6.3.4. "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.

6.3.5. "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

```
Analist :: main ()
      l <- Lexicon Instance
      d <- Database Instance

      initialize l
      baglan 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()
      kisimlariAyir 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
      yuklemleriBul
      kelimeGruplariniBul
      classifyNamedEntities
end
```

7.1.3. "database" PACKAGE

```
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

First of all, we will continue to study on learning algorithms and named entity recognition problems. Besides we will start to implement our modules, so that every component of the project will have some basic functionality until the first prototype. Moreover we will start building our own lexicon, which will be improved continuously. Another issue that we have to deal with is spellchecking algorithms.

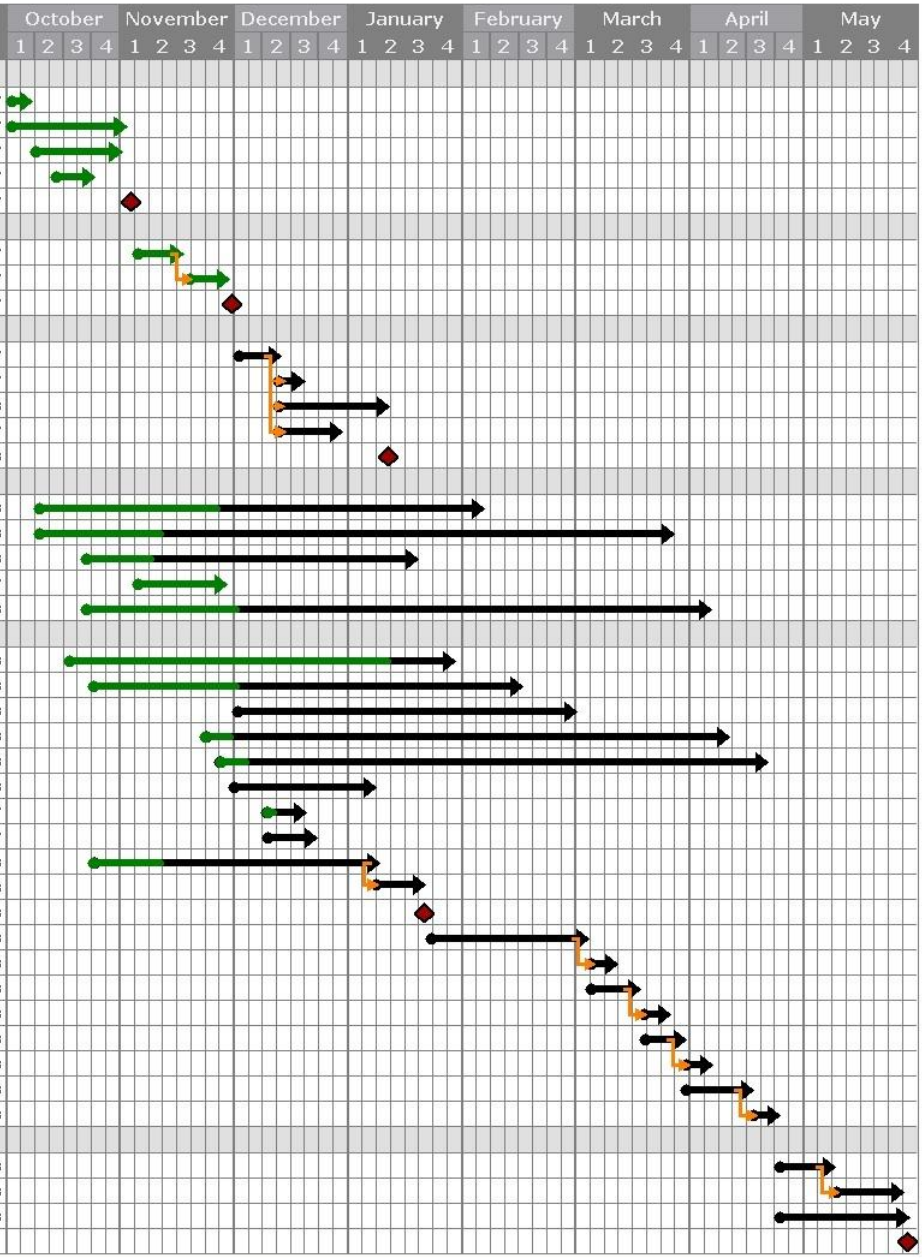
After the prototype, implementation will accelerate and we will incrementally release new versions over and over, until we are satisfied with the rate and quality of the extracted information.

9. APPENDIX

9.1. GANTT CHART

GANTT CHART - 9 MONTH TIME LINE

Tasks	Duties	Date Start	Date Finish
-Analysis-			
Proposal	All	28.09.07	05.10.07
Gathering requirements	All	30.09.07	02.11.07
Literature survey	All	06.10.07	31.10.07
Platform decisions	All	10.10.07	23.10.07
Milestone (Req.)			02.11.07
-Design-			
General architecture	All	05.11.07	20.11.07
Design of Components	All	10.11.07	30.11.07
Milestone (Init. Design)			30.11.07
-Detailed Design-			
Detailed architecture	All	02.12.07	12.12.07
User Interface design	Onur, Soner	12.12.07	15.12.07
Function/Class design	All	12.12.07	10.01.08
Database design	Tahir, Mert	12.12.07	30.12.07
Milestone (Final Design)			11.01.08
-Research -			
NLP	Soner, Mert	08.10.07	05.02.08
Machine Learning	Tahir, Onur	08.10.07	25.03.08
Ontology	Soner, Onur	18.10.07	17.01.08
Ext. dictionary conn.	Tahir	05.11.07	30.11.07
DBMSs	Tahir	18.10.07	06.04.08
-Implementation-			
Text Preprocessor	Soner	15.10.07	27.01.08
Morphologic analyzer	Mert	20.10.07	15.02.08
Spell checker	Tahir, Soner	02.12.07	01.03.08
Building lexion	All	21.11.07	10.04.08
Semantic associator	Tahir, Onur	25.11.07	19.04.08
First learning module	Tahir	01.12.07	07.01.08
User Interface	Onur	08.12.07	18.12.07
Database connection	Mert, Tahir	08.12.07	20.12.07
Prepare first prototype	All	20.10.07	07.01.08
Prototype testing	All	07.01.08	18.01.08
Milestone (Prot. demo)			18.01.08
Second build	All	20.01.08	03.03.08
2nd Build Testing	All	03.03.08	10.03.08
Third build	All	03.03.08	17.03.08
3rd Build Testing	All	17.03.08	24.03.08
Fourth build	All	17.03.08	30.03.08
4th Build Testing	All	30.03.08	06.04.08
Integrated builds	All	30.03.08	17.04.08
Builds Testing	All	17.04.08	23.04.08
-Final Release-			
Prepare final version	All	24.04.08	07.05.08
Testing&Debugging	All	07.05.08	26.05.08
Documentation	All	24.04.08	27.05.08
Milestone (Final Demo)			?



10. REFERENCES

1. Software Engineering A Practitioner's Approach, 5th Edition , **Roger S. Pressman**
2. Schaum's Outlines Software Engineering, **David Gustafson**
3. Component Oriented Software Engineering, **Ali H. Doğru**
4. Wikipedia , http://en.wikipedia.org/
5. db4o entry on Wikipedia, http://en.wikipedia.org/wiki/Db4o
6. db4o, http://www.db4o.com/
7. WordNet entry on Wikipedia, http://en.wikipedia.org/wiki/WordNet
8. WordNet, http://wordnet.princeton.edu/
9. TDK, http://www.tdk.gov.tr/
10. JWNL, http://sourceforge.net/projects/jwordnet
11. The Text Mining Handbook, **Ronen Feldman / James Sanger**
12. Database Management Systems, 2nd Edition, **Raghu Ramakrishnan / Johannes Gerke**
13. The Unified Modeling Language User Guide, **G. Booch / J. Rumbaugh / I. Jacobson**