[TURKUAZ]

MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

‘Text Mining On Turkish Medical Radiology Reports’

FINAL DESIGN REPORT

RadioRead

By

SELVİ BOYLUM AL
Yazılım

Fall, 2007

Esra Abacıoğlu – 1394568

Kerem Hadımlı – 1448752

Çiğdem Okuyucu – 1448976

Makbule Gülçin Özsoy – 1395383

İpek Tatlı – 1395557

TABLE OF CONTENTS

1. Introduction

1.1. Project Title

Our project title is ***RadioRead***.

1.2. Project Definition and Goal

Nowadays, medical imaging is gaining more and more importance in health care services. Quality of the medical images is not enough on its own for acquiring information on patients. Images need to be accurately interpreted and reported by doctors. Today, the reports of medical images are dictated as text by secretaries who listen to the tape records recorded by doctors while examining films and medical images of patients.

In systems currently in use, most of the medical information is stored as free-text. Getting and analyzing information from a text source is much more difficult than from a well structured information source. There is a need for extracting information from these text-based sources and storing the information in computationally accessible form.

It is obvious that there are plenty of documents which are kept in archives of hospitals. There are also many sources about medical situations like diseases, drugs and medical statistics on the Internet. The problem is that, nearly all of these are in textual formats. So there is a huge amount of data available which we can not benefit from with current methods in use. It is easy to access data from the Internet or from reports stored in hospital archives; but it is difficult to acquire and analyze the information enclosed inside these data.

RadioRead is a project in which we will do text mining on Turkish medical radiology reports. We aim to develop a useful information acquirement method from huge amount of electronic patient reports to enable secure, ethical and user friendly access to patient information. We will provide an environment for our users to access these information as easy as using a natural language; an environment in which the user does not have to know anything about technical aspects of how the information is represented in the database systems involved. As a result; detailed information about patients can be accessed easily; more information about a patient can be given to his/her doctor before consultations; the information can be used by doctors to diagnose diseases of other patients; and statistics can be derived.

According to the market research we have done so far, we have seen that there is neither enough research nor sufficient number of production level projects for text mining in Turkish. This insufficiency is caused by the difficulty in analyzing the characteristics of the language, and also by lack of market compared to English. In this project we plan to handle usual difficulties of extracting information from free-text clinical reports, besides providing a usable interface for different users (like doctors, assistants or statisticians) who may not have sufficient technical knowledge to use a complex program efficiently.

We have seen in our market research that most of current NLP tools use Machine Learning techniques to achieve success. We believe that our inputs, the radiology reports, contain dense information and are unsuitable for information extraction via Machine Learning. Also motivated by well-defined structures of Turkish language, our aim in this project is to implement a rule based technique for parsing medical reports.

1.3. Design Goals

1.3.1. Robustness

RadioRead will be able to manage invalid user inputs or inconsistent conditions. It provides error checking to ensure the right input format and returns errors and warnings to the user.

1.3.2. Usability

The users of RadioRead will be medical staff, doctors and statisticians. Since all staff will not be experienced in computers we have a special need for user friendly graphical user interface. While using RadioRead the user will face with a familiar environment, which eases the general use of the application.

1.3.3. Correctness

RadioRead will be used in a human-life related context, as in all medical software. All the results of RadioRead may affect diagnosis and medical decisions, thus, they need to be correct to their foremost. More important than the need of maximizing successful analysis, minimizing and preventing erroneous analysis is a must.

2. Design Constraints

2.1. Experience & Skills of Members

As developers, our programming and design skills and experiences is one of the restrictions. It is very difficult for us to manage unexpected problems about this field but we may consult experienced people to get help about solving problems. We may not be able to achieve 100% success because we were not familiar to this topic before; but we believe we have progressed a lot in this field since the beginning of this semester.

2.2. Time Constraints

We have to finish our project by June and also we should have provided a prototype at the end of this semester. Therefore, especially for a software project, this is the most important constraint.

We have already finished our prototype. Still, being able to use our time efficiently is very important for us to follow our schedule. Since we must finish the project at the end of next semester, we will focus on the project and spend more time on it.

2.3. Resource Constraints

While we are doing our project, we need different software resources such as external dictionaries. We will be able to access and use these resources. We will need a database server. Dictionaries that we will create manually during the project may also restrict our project development.

3. Project Requirements

3.1. System Requirements

General Aspects

- Java as a programming language
- PostgreSQL Database Management System
- Hibernate library for persisting Java objects directly in DBMS
- Zemberek library[1]

Development Side

- Eclipse as development environment
- Installed Java Development Kit
- Subversion server for version control
- GNU/Linux or Windows XP environment
- Internet access for online dictionary support (Zargan[2], TDK[3])

End-user Side

- PostgreSQL Database Management System
- Java Run Time Environment 6
- Windows XP or recent GNU/Linux Distribution
- Internet access for online dictionary support (Zargan, TDK)

3.2. Functional Requirements

3.2.1 Text-Mining and Representing Information Formally

RadioRead application will be provided with free-text radiology reports. We have to extract the information in these texts and represent these in a database, in a structured way. We will use Natural Language Processing (NLP) with rule based techniques for this task. NLP requires Morphological, Syntactic and Semantic analysis. We will utilize Zemberek library for morphological analysis, and will use Zargan and TDK online dictionaries in the cases when Zemberek does not have the roots of a given word. We will then apply syntactic analysis, where we will spot verbs, subject(s) and indirect objects (“Dolaylı Tümleç”). After this step, we enter the Semantic Analysis step. The verb or verb phrase found in the sentence will be first looked up from our internal predefined “meaning list”, and in cases it does not exist in our internal dictionaries, it will be looked up from external dictionaries, such that Zargan, TDK Dictionary, to find synonyms that matches with the internal dictionary. This result can be also affected with qualifiers such as “-ma” negativity suffix that inverses meaning of a verb. This information will be used to mark the information listed in the sentence to be “normal”, “abnormal”, “exists”, “not exists”. The subject(s) of the sentence found in this syntactic analysis step are groups of noun phrases referring to what-quality information about findings mentioned. In order to break them into meaningful pieces, we have written our own noun phrase parser. Indirect-objects refer to location-measurement information. The same noun phrase parser is utilized for these too. The structured information will then be recorded in the database, linked with similar records.

3.2.2. Statistical Analysis and Information Retrieval

We need to provide reasonable methods to a statistician for querying the accumulated information from the analyzed radiology reports. The accumulated information is valuable as a large-scale radiology data mined from free texts, which can be benefited from. A statistician does not only require to query the data using qualifiers mined from the free-texts, but also additional qualifiers (meta information) such as age range, date, frequency.

Besides the requirements to analyze accumulated information in a broad sense, a doctor needs to query a specific patient’s history about a specific diagnosis or disease. This way, a doctor can control the progress of a patient without having to search all the reports of the patient for a specific item manually.

Thus, we need to provide two similar but slightly different ways for retrieval of mined information.

3.2.3. Holding Meta Information about Patients and Reports

In order to meet requirements of statistical analysis and information retrieval, we need to store meta information about patients and reports. Meta information of a patient holds fields such as age or gender. We also need to store date, or doctor (writing the report) information with the report records. Besides being useful in statistical analysis, the application has to provide a convenient and intuitive way for users, mostly for doctors, to access data. That’s why we need to hold additional information such as name of a patient.

3.2.4. User Interface

- Authentication / Authorization

    - Logging into system via usernames and passwords

- User account management
  - Adding accounts
  - Modifying accounts
- Patient management
  - Adding patients
  - Modifying patient information
  - Listing patients, filtering
- Report management
  - Adding reports associated with patients: This will invoke data mining
  - Listing reports, querying for a patient's specific reports: There needs to be options for querying mined information, besides simple filtering based on meta data
  - Viewing reports
- Statistical querying
  - Query interface: We need a query interface where a user can create his/her queries in an intuitive way, such as constructing free-text like sentences using dropdowns. Our users are not technically skilled, so users need to see the constructed queries in a natural way, for ease of use.

3.3. Non-Functional Requirements

We need to provide the user an intuitively usable interface, which will require almost no training to learn, and consume minimum time to fill data and query information. Our intended users will be neither skilled nor interested in computers. In order for them to use the application efficiently, the user interface needs to be simple and useful. Especially the statistical query and information retrieval interfaces need to be designed with ease in mind, as these can be complex even for experienced computer users if not designed carefully.

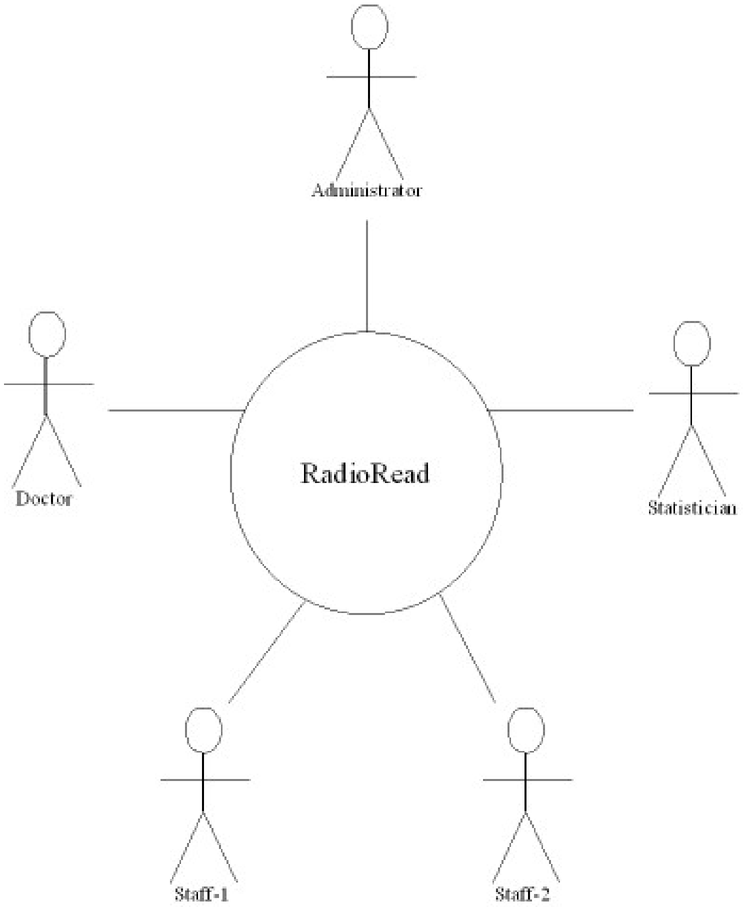
Besides the interface, we need to provide the security of the patients' information. Patients trust doctors and hospitals to store their data, and only the people who are authorized to see their information should be able to view them.

The application needs to be responsive, especially in mining information from reports and querying of mined information. Both reports and queries may be very complicated, but they should not discourage the user because of latency.

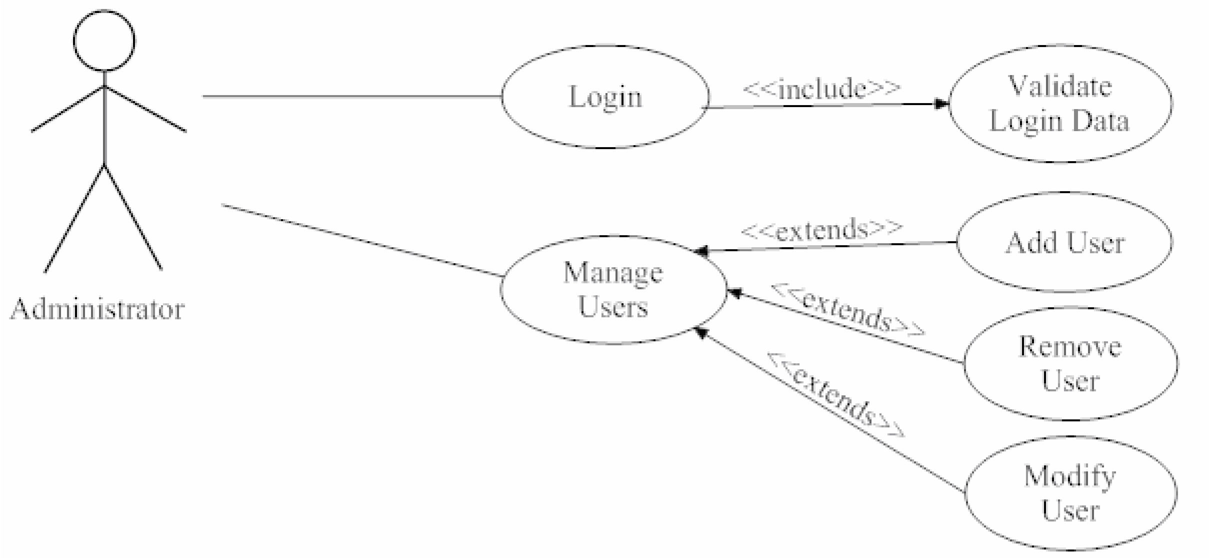
3.4. User Requirements

3.4.1. Use Case Diagrams

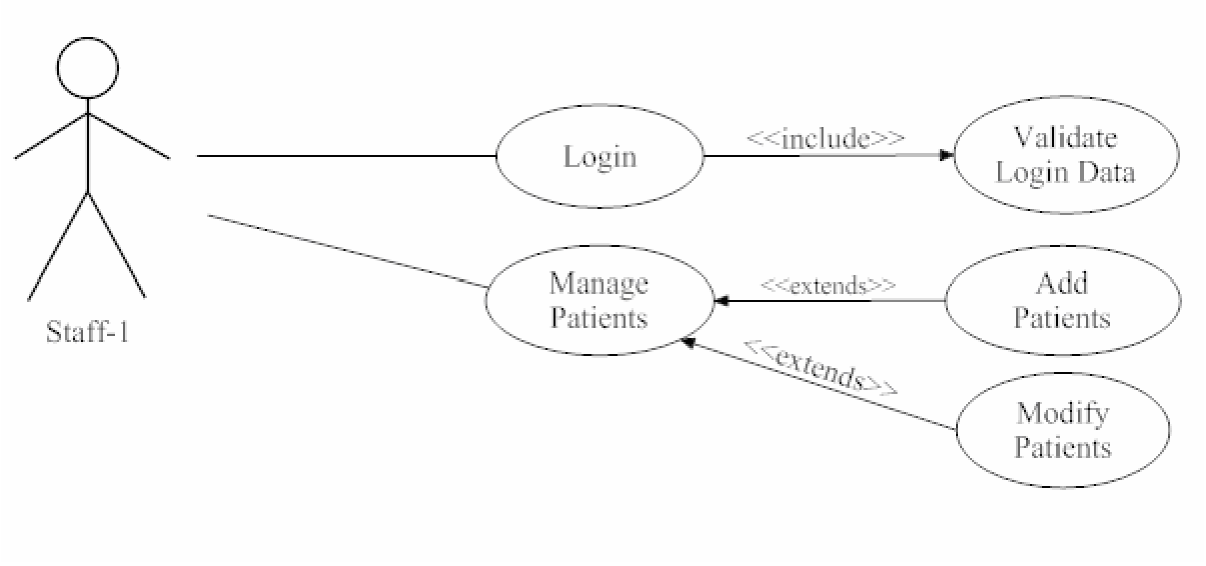
3.4.1.1. Overview



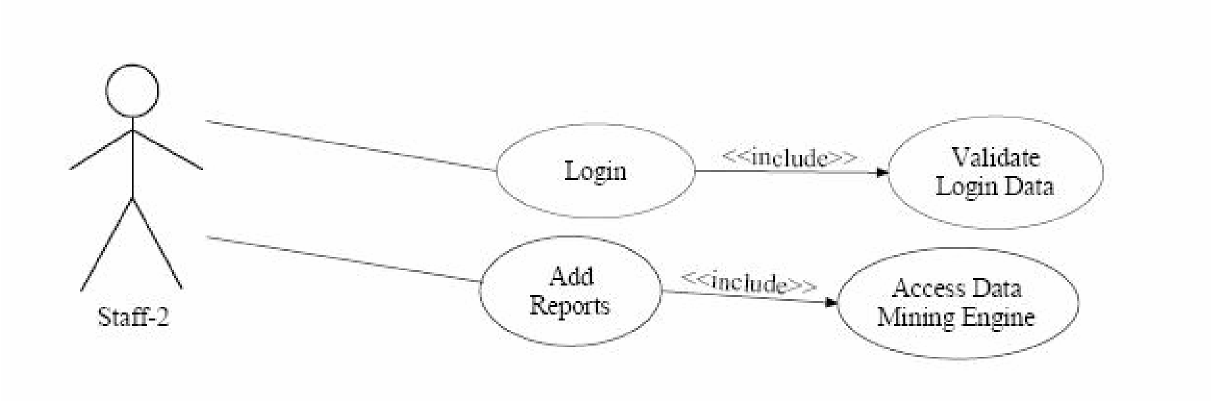
3.4.1.2. Use Case Diagram for Administrator



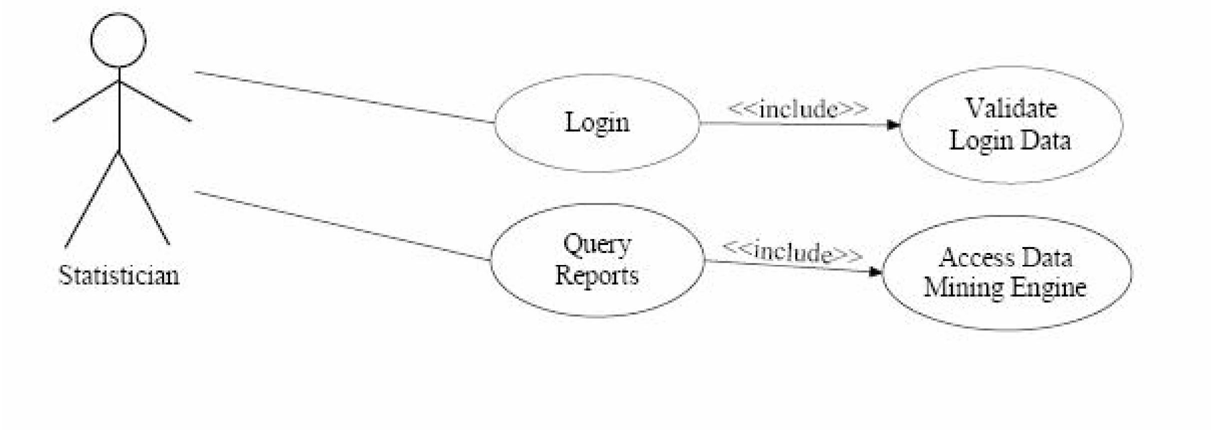
3.4.1.3. Use Case Diagram for Staff-1



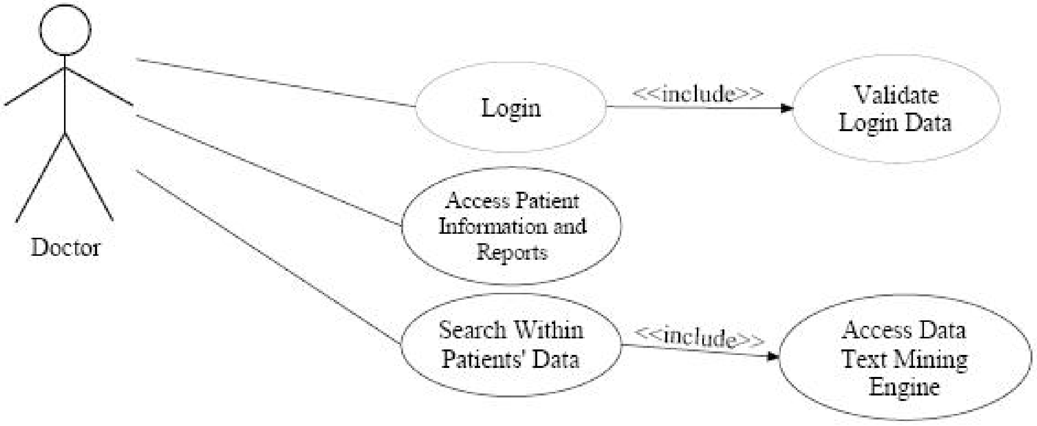
3.4.1.4. Use Case Diagram for Staff-2



3.4.1.5. Use Case Diagram for Statistician



3.4.1.6. Use Case Diagram for Doctor


Doctor
Login
<<include>>
Validate Login Data
Access Patient Information and Reports
Search Within Patients' Data
<<include>>
Access Data Text Mining Engine

3.4.2. Use Case Scenarios

3.4.2.1. Administrator

- ***Login:*** An administrator has to login to the system in order to realize administrative roles. There will be a user interface for administrative roles. After validation of login information, the administrator will be able to manage users.
- ***Manage Users:*** Administrator may add, remove users and modify the user information. There will be specified user roles and rights and administrator will control users and will be able to restrict the user rights.

3.4.2.2. Staff-1

- ***Login:*** A staff1 has to login to the system in order to realize his/her roles. There will be a user interface for him/her. After validation of login information, the staff1 will be able to manage patients.
- ***Manage Patients:*** Staff1 may add patients and modify the patient information. None of the patients who had been in the clinic will be deleted even if they are dead.

3.4.2.3. Staff-2

- ***Login:*** A staff2 has to login to the system in order to realize his/her roles. There will be a user interface for him/her. After validation of login information, the staff2 will be able to manage reports.
- ***Add Reports:*** Staff2 may add reports to the records of related patients. These reports will then be used for acquiring necessary information.

3.4.2.4. Statistician

- ***Login:*** A statistician has to login to the system in order to realize his/her roles. There will be a user interface for him/her. After validation of login information, the statistician will be able to manage query reports.
- ***Query Reports:*** Statistician may send queries about reports to data mining engine through GUI, and get statistical mined information.

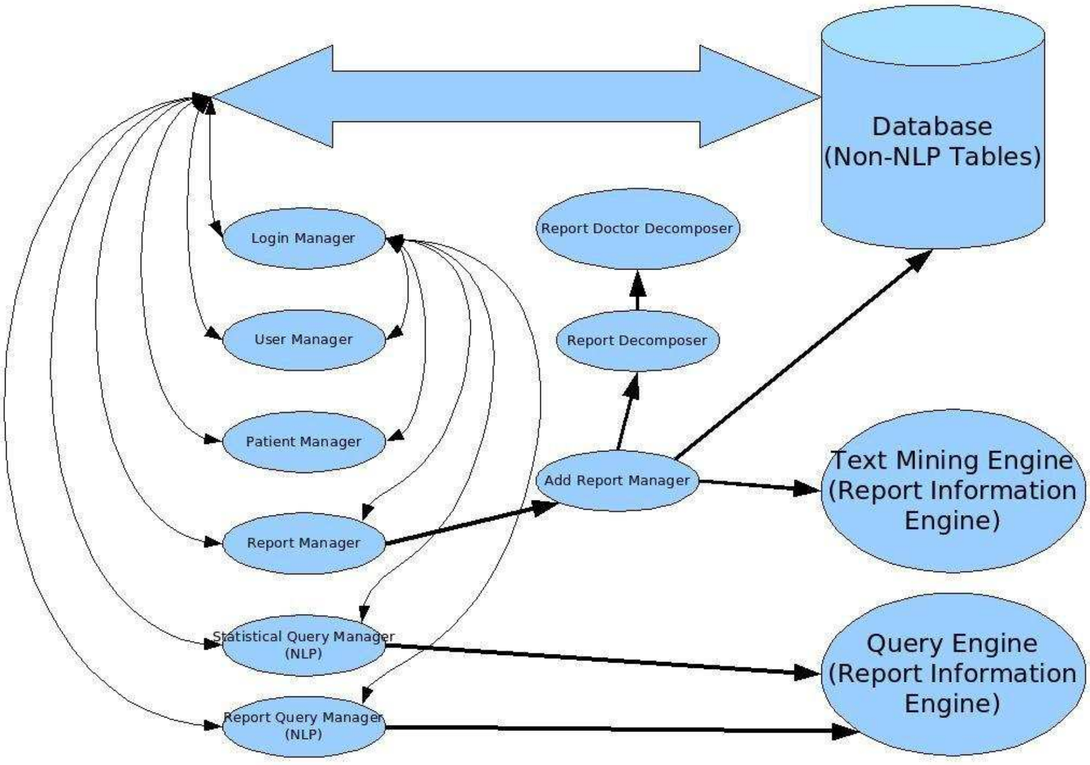
3.4.2.5. Doctor

- ***Login:*** A doctor has to login to the system in order to realize his/her roles. There will be a user interface for him/her. After validation of login information, the doctor will be able to manage query reports.
- ***Access Information of Reports:*** Doctor is the only user who can reach the pure text of patients' reports.
- ***Search within Patients Data:*** Doctor may send queries about patients to data mining engine through GUI, and get of mined information of patients.

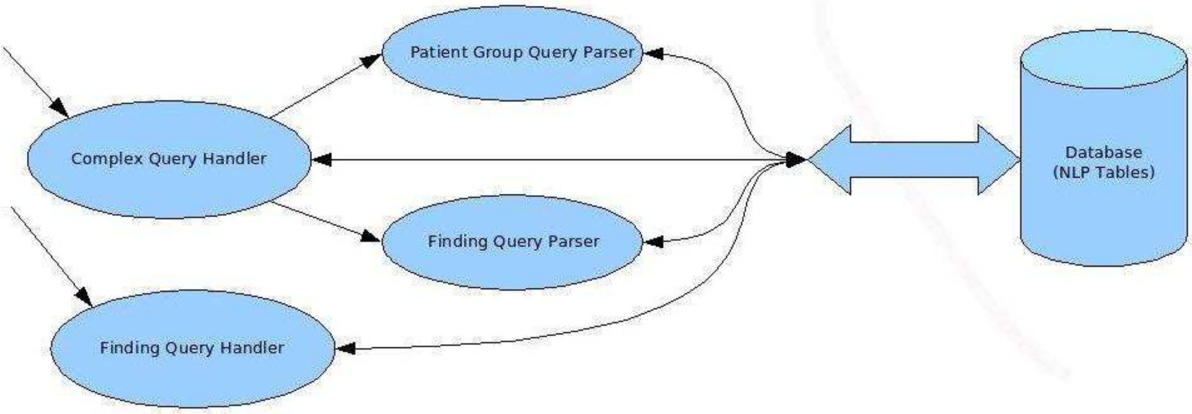
4. System Architecture and Component Level Design

4.1. System architecture

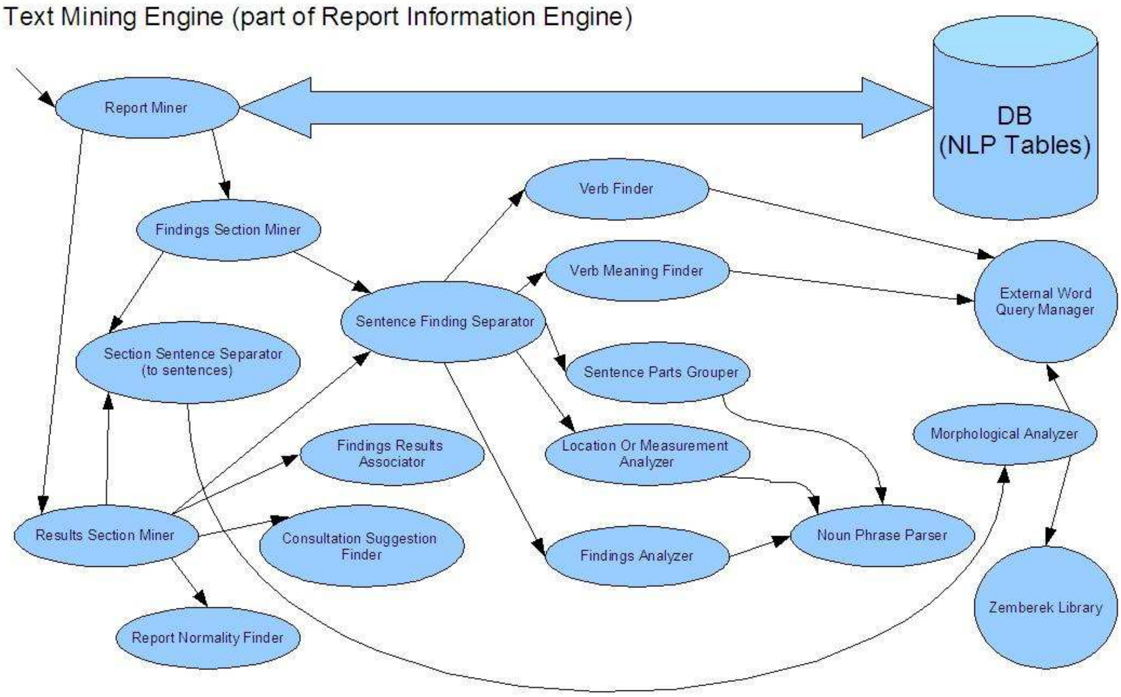
4.1.1. Data Engine



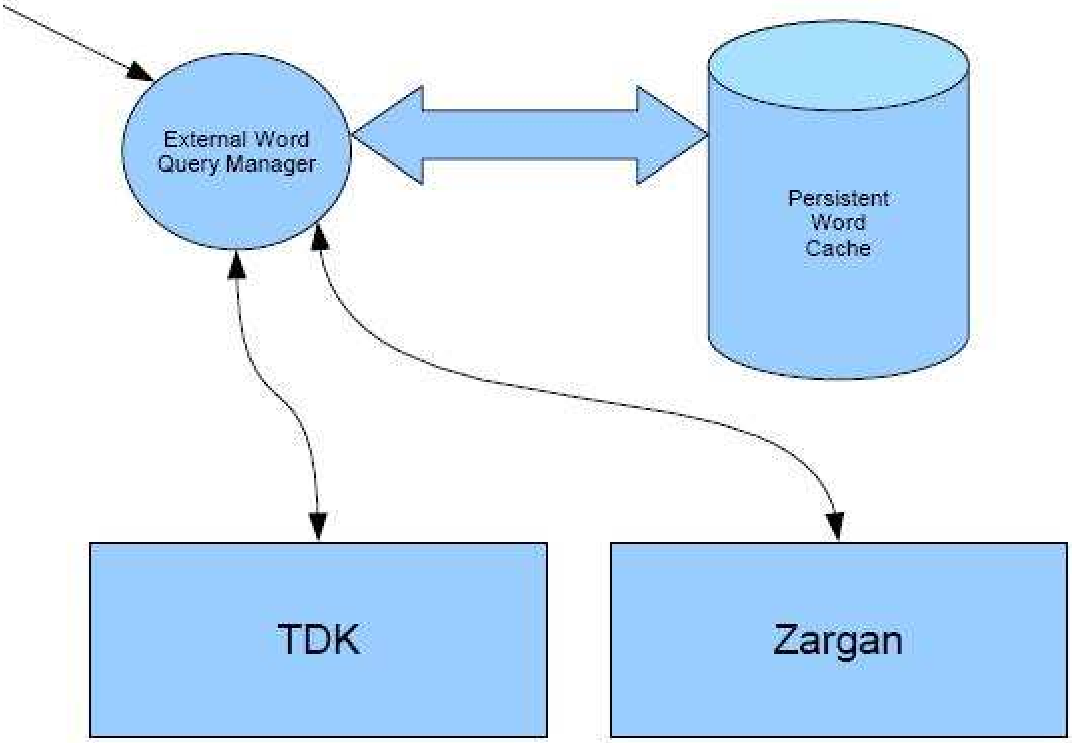
4.1.2. Query Analyzer (Query Engine)



4.1.3. Text Mining Engine

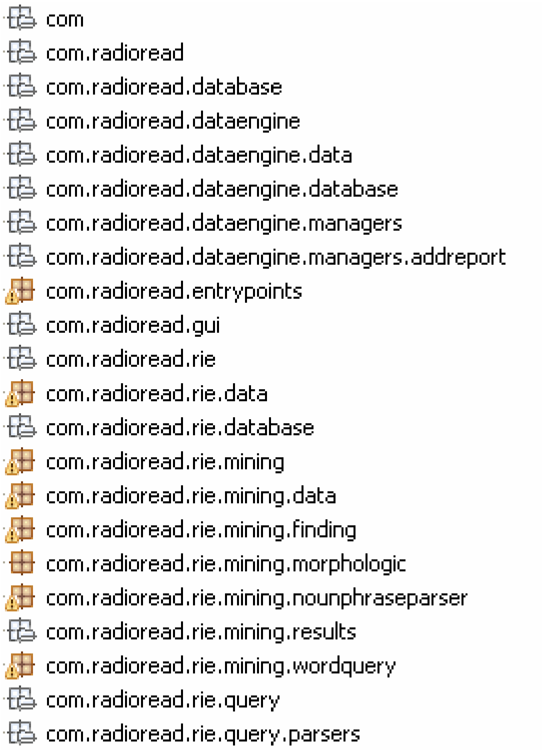


4.1.4. External Word Query Manager



4.2. Java Package Hierarchy

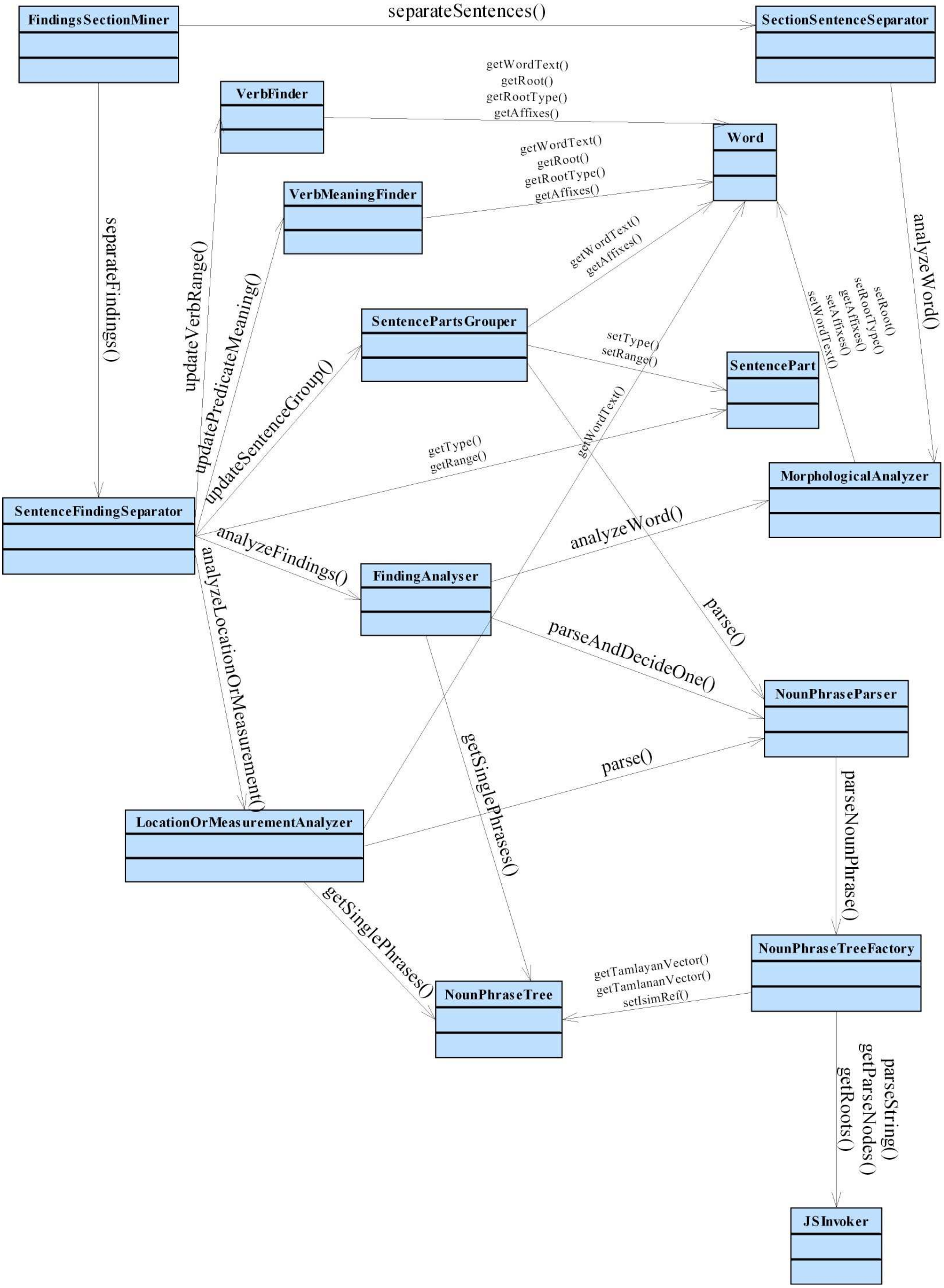
We have grouped our classes in packages and designed our package hierarchy in a clean way to get the most out of Java. This hierarchy can be seen below.



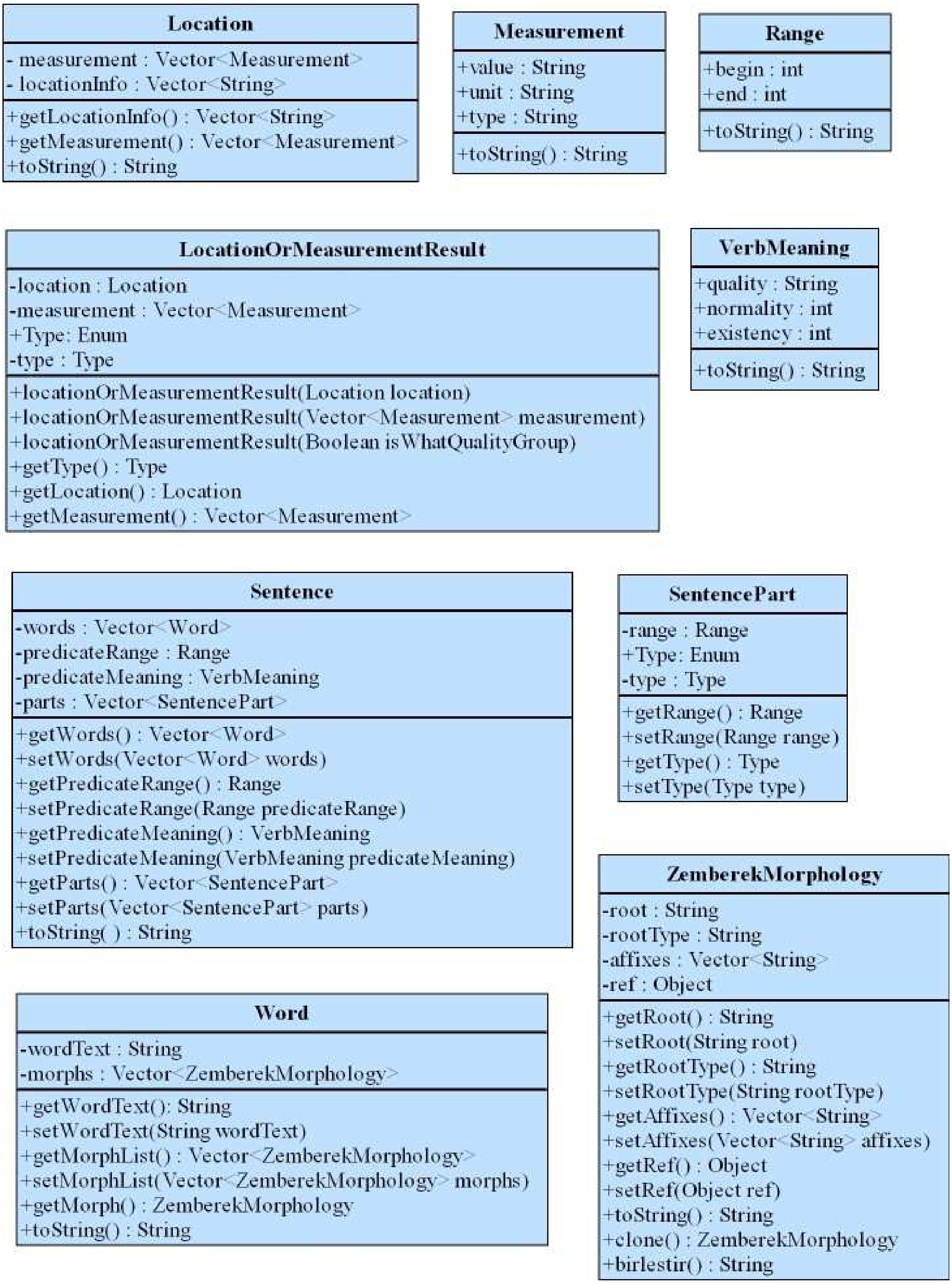
```
com
com.radioread
com.radioread.database
com.radioread.dataengine
com.radioread.dataengine.data
com.radioread.dataengine.database
com.radioread.dataengine.managers
com.radioread.dataengine.managers.addreport
com.radioread.entrypoints
com.radioread.gui
com.radioread.rie
com.radioread.rie.data
com.radioread.rie.database
com.radioread.rie.mining
com.radioread.rie.mining.data
com.radioread.rie.mining.finding
com.radioread.rie.mining.morphologic
com.radioread.rie.mining.nounphraseparser
com.radioread.rie.mining.results
com.radioread.rie.mining.wordquery
com.radioread.rie.query
com.radioread.rie.query.parsers
```

4.3. Component Level Design

4.3.1. Class Diagram



4.3.2. Classes



Location

- measurement : Vector<Measurement>
- locationInfo : Vector<String>

+getLocationInfo() : Vector<String>
+getMeasurement() : Vector<Measurement>
+toString() : String

Measurement

+value : String
+unit : String
+type : String

+toString() : String

Range

+begin : int
+end : int

+toString() : String

LocationOrMeasurementResult

-location : Location
-measurement : Vector<Measurement>
+Type: Enum
-type : Type

+locationOrMeasurementResult(Location location)
+locationOrMeasurementResult(Vector<Measurement> measurement)
+locationOrMeasurementResult(Boolean isWhatQualityGroup)
+getType() : Type
+getLocation() : Location
+getMeasurement() : Vector<Measurement>

VerbMeaning

+quality : String
+normality : int
+existency : int

+toString() : String

Sentence

-words : Vector<Word>
-predicateRange : Range
-predicateMeaning : VerbMeaning
-parts : Vector<SentencePart>

+getWords() : Vector<Word>
+setWords(Vector<Word> words)
+getPredicateRange() : Range
+setPredicateRange(Range predicateRange)
+getPredicateMeaning() : VerbMeaning
+setPredicateMeaning(VerbMeaning predicateMeaning)
+getParts() : Vector<SentencePart>
+setParts(Vector<SentencePart> parts)
+toString( ) : String

SentencePart

-range : Range
+Type: Enum
-type : Type

+getRange() : Range
+setRange(Range range)
+getType() : Type
+setType(Type type)

ZemberekMorphology

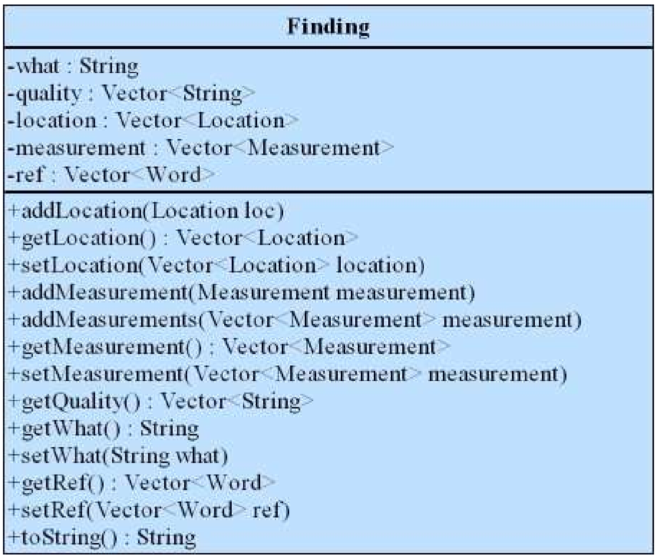
-root : String
-rootType : String
-affixes : Vector<String>
-ref : Object

+getRoot() : String
+setRoot(String root)
+getRootType() : String
+setRootType(String rootType)
+getAffixes() : Vector<String>
+setAffixes(Vector<String> affixes)
+getRef() : Object
+setRef(Object ref)
+toString() : String
+clone() : ZemberekMorphology
+birlestir() : String

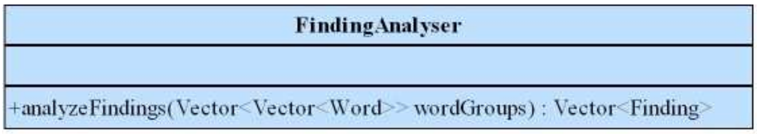
Word

-wordText : String
-morphs : Vector<ZemberekMorphology>

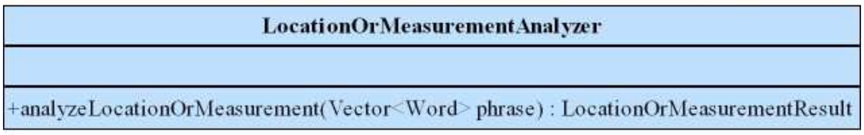
+getWordText(): String
+setWordText(String wordText)
+getMorphList() : Vector<ZemberekMorphology>
+setMorphList(Vector<ZemberekMorphology> morphs)
+getMorph() : ZemberekMorphology
+toString() : String



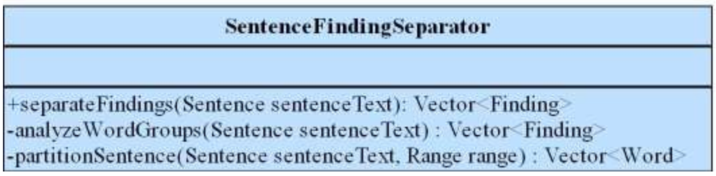
```
Finding
-what : String
-quality : Vector<String>
-location : Vector<Location>
-measurement : Vector<Measurement>
-ref : Vector<Word>
+addLocation(Location loc)
+getLocation() : Vector<Location>
+setLocation(Vector<Location> location)
+addMeasurement(Measurement measurement)
+addMeasurements(Vector<Measurement> measurement)
+getMeasurement() : Vector<Measurement>
+setMeasurement(Vector<Measurement> measurement)
+getQuality() : Vector<String>
+getWhat() : String
+setWhat(String what)
+getRef() : Vector<Word>
+setRef(Vector<Word> ref)
+toString() : String
```



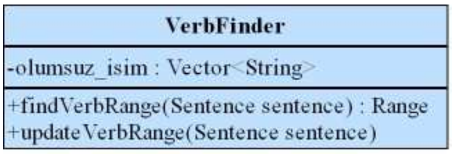
```
FindingAnalyser
+analyzeFindings(Vector<Vector<Word>> wordGroups) : Vector<Finding>
```



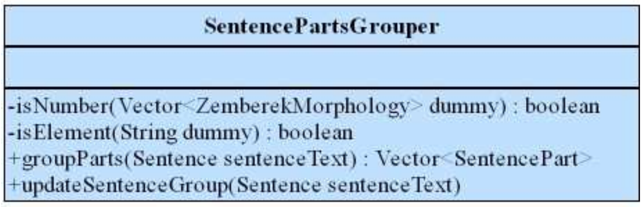
```
LocationOrMeasurementAnalyzer
+analyzeLocationOrMeasurement(Vector<Word> phrase) : LocationOrMeasurementResult
```



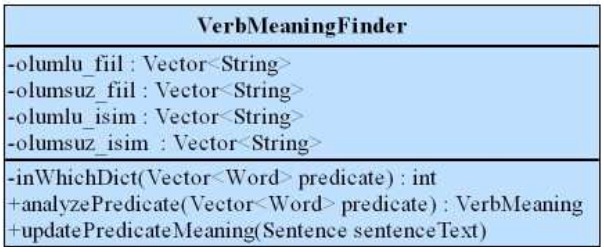
```
SentenceFindingSeparator
+separateFindings(Sentence sentenceText): Vector<Finding>
-analyzeWordGroups(Sentence sentenceText) : Vector<Finding>
-partitionSentence(Sentence sentenceText, Range range) : Vector<Word>
```



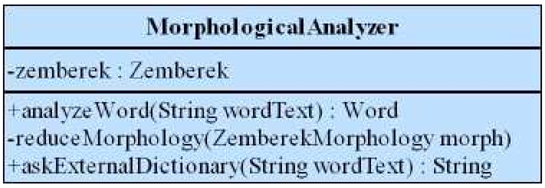
```
VerbFinder
-olumsuz_isim : Vector<String>
+findVerbRange(Sentence sentence) : Range
+updateVerbRange(Sentence sentence)
```

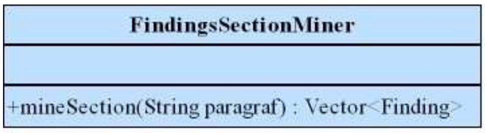


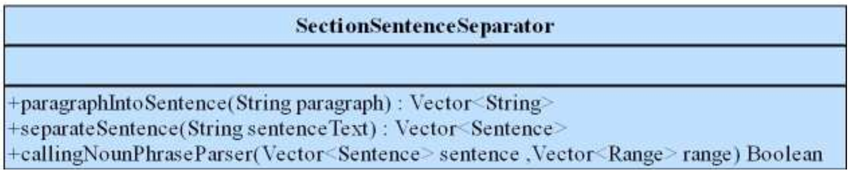
```
SentencePartsGrouper
-isNumber(Vector<ZemberekMorphology> dummy) : boolean
-isElement(String dummy) : boolean
+groupParts(Sentence sentenceText) : Vector<SentencePart>
+updateSentenceGroup(Sentence sentenceText)
```

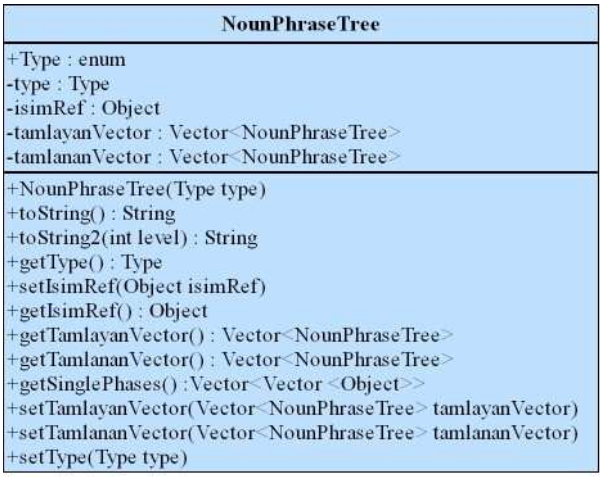


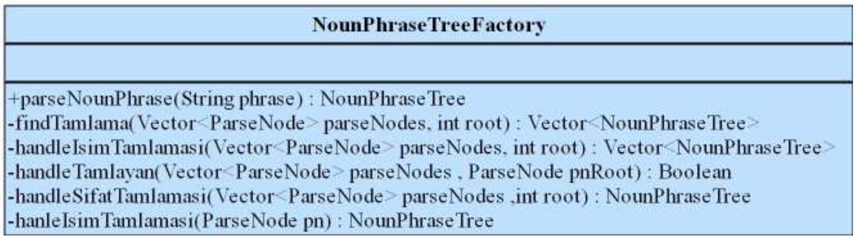
```
VerbMeaningFinder
-olumlu_fiil : Vector<String>
-olumsuz_fiil : Vector<String>
-olumlu_isim : Vector<String>
-olumsuz_isim  : Vector<String>
-inWhichDict(Vector<Word> predicate) : int
+analyzePredicate(Vector<Word> predicate) : VerbMeaning
+updatePredicateMeaning(Sentence sentenceText)
```

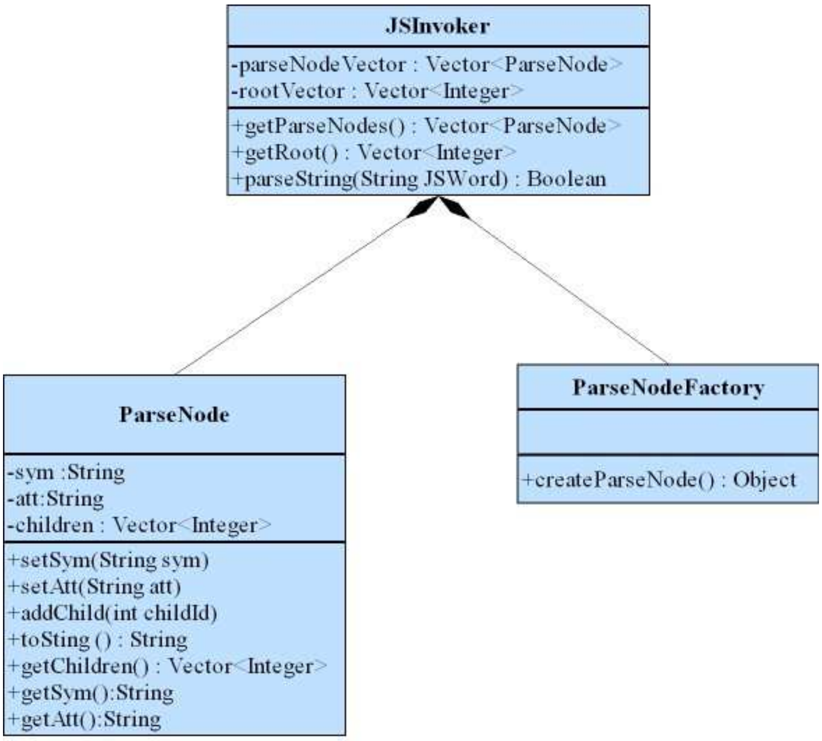

MorphologicalAnalyzer
-zemberek : Zemberek
+analyzeWord(String wordText) : Word
-reduceMorphology(ZemberekMorphology morph)
+askExternalDictionary(String wordText) : String

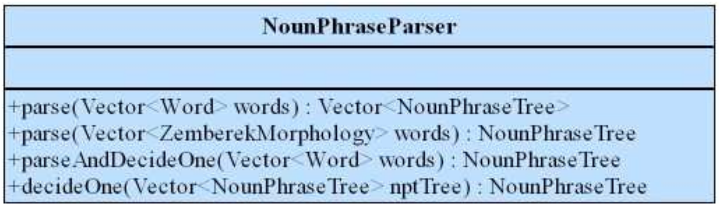

FindingsSectionMiner
+mineSection(String paragraf) : Vector<Finding>

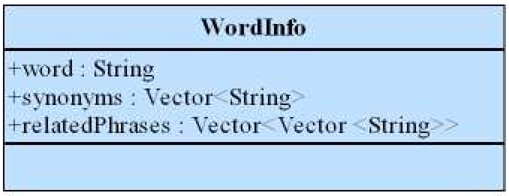

SectionSentenceSeparator
+paragraphIntoSentence(String paragraph) : Vector<String>
+separateSentence(String sentenceText) : Vector<Sentence>
+callingNounPhraseParser(Vector<Sentence> sentence ,Vector<Range> range) Boolean

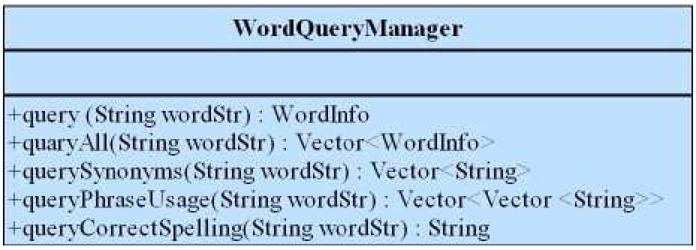

NounPhraseTree
+Type : enum
-type : Type
-isimRef : Object
-tamlayanVector : Vector<NounPhraseTree>
-tamlananVector : Vector<NounPhraseTree>
+NounPhraseTree(Type type)
+toString() : String
+toString2(int level) : String
+getType() : Type
+setIsimRef(Object isimRef)
+getIsimRef() : Object
+getTamlayanVector() : Vector<NounPhraseTree>
+getTamlananVector() : Vector<NounPhraseTree>
+getSinglePhases() :Vector<Vector <Object>>
+setTamlayanVector(Vector<NounPhraseTree> tamlayanVector)
+setTamlananVector(Vector<NounPhraseTree> tamlananVector)
+setType(Type type)


NounPhraseTreeFactory
+parseNounPhrase(String phrase) : NounPhraseTree
-findTamlama(Vector<ParseNode> parseNodes, int root) : Vector<NounPhraseTree>
-handleIsimTamlamasi(Vector<ParseNode> parseNodes, int root) : Vector<NounPhraseTree>
-handleTamlayan(Vector<ParseNode> parseNodes , ParseNode pnRoot) : Boolean
-handleSifatTamlamasi(Vector<ParseNode> parseNodes ,int root) : NounPhraseTree
-hanleIsimTamlamasi(ParseNode pn) : NounPhraseTree


JSInvoker
-parseNodeVector : Vector<ParseNode>
-rootVector : Vector<Integer>
+getParseNodes() : Vector<ParseNode>
+getRoot() : Vector<Integer>
+parseString(String JSWord) : Boolean
ParseNode
-sym :String
-att:String
-children : Vector<Integer>
+setSym(String sym)
+setAtt(String att)
+addChild(int childId)
+toSting () : String
+getChildren() : Vector<Integer>
+getSym():String
+getAtt():String
ParseNodeFactory
+createParseNode() : Object


NounPhraseParser
+parse(Vector<Word> words) : Vector<NounPhraseTree>
+parse(Vector<ZemberekMorphology> words) : NounPhraseTree
+parseAndDecideOne(Vector<Word> words) : NounPhraseTree
+decideOne(Vector<NounPhraseTree> nptTree) : NounPhraseTree


WordInfo
+word : String
+synonyms : Vector<String>
+relatedPhrases : Vector<Vector <String>>


WordQueryManager
+query (String wordStr) : WordInfo
+quaryAll(String wordStr) : Vector<WordInfo>
+querySynonyms(String wordStr) : Vector<String>
+queryPhraseUsage(String wordStr) : Vector<Vector <String>>
+queryCorrectSpelling(String wordStr) : String

4.3.3. Class Diagram Dictionary

LoginManager	
`login(username, password)`	Query from database if user has valid username and password. Set the logged in user (internally)
`logout()`	Logout current user (internally)
`getUserName()`	Returns username of currently logged in user
`canManagePatients()`	Checks privileges
`canAddReports()`	Checks privileges
`canQueryReports()`	Checks privileges
`canAccessPatients()`	Checks privileges
`canManageUsers()`	Checks privileges

UserManager	
`addUser( userInfo )`	Creates a new user with given information
`updateUser( userInfo )`	Updates user information
`listUsers( )`	Returns list of users in system

PatientManager	
`addPatient( patientInfo )`	Creates a new patient with given information
`updatePatient( patientInfo )`	Updates patient information
`listPatients( )`	Returns list of patients in system
`listPatients( constraints )`	Lists patients fulfilling constraints. `constraints` contains information such as gender, age range, fragments of name/surname

ReportManager	
`addReport( reportText )`	Adds a new report to system. The report is sent to Add Report Manager. `reportText` is in the format given as examples to SBAYazılım (type is `String`)
`listReports( )`	Lists all reports in system
`listReports( constraints )`	Lists reports fulfilling constraints. `constraints` contains information such as patient id, date range, words in title.

AddReportManager	
`addReport( reportText )`	Adds a new report to system. The report text is first sent to ReportDecomposer to extract 6 components of the report, and then added to database (to Non-NLP tables). Then, it calls TextMiningEngine with the report id, to make the report analyzed. `reportText` is in the format given as examples to SBAYazılım (type is `String`)

ReportDecomposer	
`decompose( reportText )`	Given the free text, it extracts 6 components and returns them. The components are “Başlık” ,“Klinik Bilgi”, ”Teknik”, ”Bulgular”, ”Sonuç” and “Yazan Doktorlar” `reportText` is in the format given as examples to SBAYazılım (type is `String`)

ReportDoctorDecomposer	
`decompose( doctorsComponent )`	Given the “Doktorlar” component of free text, and extracts the list of doctor names.

StatisticalQueryManager	
`queryForCount( hastaGrubuNode )`	Given the root node of `hasta_grubu` (see statistical query grammar), returns number of patients in that group
`queryForPercentage( hastaGrubuNode1, hastaGrubuNode2 )`	Given the root nodes of 2 `hasta_grubu` (see statistical query grammar), returns percentage of group 2 over group 1
`queryForMeasurementGraph( hastaGrubuNode, bulguNode, measurementType, numberOfBars )`	Given 2 root nodes, one measurement type, and a number specifying number of groups in the resulting graph, returns a list containing `numberOfBars` numbers (see statistical query grammar)

ReportQueryManager	
`setPatientId( patientId )`	Sets the patient ID to be used for subsequent calls
`getFindings( )`	Returns all findings extracted from all reports of the patient
`getFindings( constraints )`	Returns all findings extracted from all reports of the patient, limited by `constraints`

ComplexQueryHandler	
queryForCount(hastaGrubuNode)	Given the root node of hasta_grubu (see statistical query grammar), returns number of patients in that group
queryForPercentage(hastaGrubuNode1, hastaGrubuNode2)	Given the root nodes of 2 hasta_grubu (see statistical query grammar), returns percentage of group 2 over group 1
queryForMeasurementGraph(hastaGrubuNode, bulguNode, measurementType, numberOfBars)	Given 2 root nodes, one measurement type, and a number specifying number of groups in the resulting graph, returns a list containing numberOfBars numbers (see statistical query grammar)

PatientGroupQueryParser	
queryPatientGroup(hastaGrubuNode)	Given the root node of hasta_grubu (see statistical query grammar), returns 1. list of patient IDs in that group 2. list of report IDs in that group

FindingQueryParser	
queryFinding(bulguNode)	Returns list of finding IDs described by bulguNode root node (see statistical query grammar)
queryFinding(bulguNode, reportIds)	Returns list of finding IDs described by bulguNode root node (see statistical query grammar) which exist in one of reports in reportIds

FindingQueryHandler	
queryFinding(constraints)	Returns list of findings according to given constraints

ReportMiner	
mineReport(reportId, findingsText, resultsText)	Mine information and insert it to NLP tables. In the end, it inserts a record to Islenmis_Raporlar with reportId (and inserts other mined information to NLP tables associated with Islenmis_Raporlar)

FindingsSectionMiner*	
Vector<Finding> mineSection(String paragraph)	Calls methods of SectionSentenceSeparator and SentenceFindingSeparator classes and returns a list of findings extracted from the paragraph.

SectionSentenceSeparator*	
`Vector<String> paragraphIntoSentence( String paragraph )`	Returns a list of string sentences extracted from the given paragraph.
`Vector<Sentence> separateSentence( String sentenceText )`	Returns a list of semantic sentences from a string sentence. Uses punctuations and verbal (fiilimsiler) to separate sentence. (Every “semantic” sentence has only one verb)

SentenceFindingSeparator*	
`Vector<Finding> separateFindings( Sentence sentenceText )`	Calls methods of `VerbFinder`, `VerbMeaningFinder`, `SentencePartsGrouper`, `LocationOrMeasurementAnalyser and FindingsAnalyser` classes and returns a list of findings extracted from the given sentence.

SentencePartsGrouper*	
`Vector<SentencePart> groupParts( Sentence sentenceText )`	Returns “location-measurement” and “quality-what” groups from the given sentence. Uses affix information.
`void updateSentenceGroup ( Sentence sentenceText)`	Calls `groupParts` and updates parts of sentenceText.

VerbFinder*	
`Range findVerbRange( Sentence sentence )`	Returns verb / verb phrase range forming the predicate of the given sentence
`void updateVerbRange ( Sentence sentence )`	Calls `findVerbRange` and updates verb range of given sentence.

VerbMeaningFinder*	
`VerbMeaning analyzePredicate( Vector<Word> predicate )`	Returns if the given verb / verb phrase gives information about normality, existence and quality.
`void updatePredicateMeaning( Sentence sentenceText )`	Calls `analyzePredicate` and updates verb meaning of given sentence.

LocationOrMeasurementAnalyzer*	
`LocationOrMeasurementResult analyzeLocationOrMeasurement( Vector<Word> phrase )`	Decides whether the given phrase shows a location, a measurement, or is irrelevant. All “Dolaylı tümleç”s are analyzed in this class. Examples: “sol memede”, “3 mm çapında”, “yapılan us incelemesinde”

FindingsAnalyzer*	
`Vector<Finding> analyzeFindings( Vector<Vector<Word>> wordGroups )`	Parses the given phrase into a list of findings (“ne”) and their qualities (“nasıl”). Most of the time, the given phrase is a complex noun phrase consisting of quality and finding name information. Returns a list of finding names and their related qualities. More than one finding may be found from a single phrase due to nature of complex noun phrases. Examples “kistik ya da solid lezyon” -> (qualities:”kistik”,finding:”lezyon”), (qualities:”solid”,finding:”lezyon”) “Midenin konturları, pasaj ve peristaltizmi ve mukozal rölyefi” -> (qualities: none, finding: “midenin konturu”), (qualities: none, finding: “midenin pasajı”), (qualities: none, finding: “midenin peristaltizmi”), (qualities: none, finding: “midenin mukozal rölyefi”)

NounPhraseTreeFactory*	
`NounPhraseTree parseNounPhrase( String phrase )`	For a given noun phrase string in form of lexemes (“^ word1 –IN word2 –I $” for “word1in word2si”), it invokes JavaScript code (using `JSInvoker` class) and calls internal methods to convert the returned syntax tree to a `NounPhraseTree`.

NounPhraseParser*	
`Vector<NounPhraseTree> parse( Vector<Word> words )`	A `Word` object can hold multiple alternative `ZemberekMorphology objects`. This method parses all `ZemberekMorphology` combinations of the given complex phrase (which is composed of adjective and nouns) into alternative parse trees. Resulting trees can be scanned from leaves to root to get single phrases. See noun phrase grammar for details. (Appendix B)
`NounPhraseTree parse( Vector<ZemberekMorphology> words )`	For a given a complex phrase of `ZemberekMorphology` objects (which is composed of adjective and nouns) this method parses it into a single parse tree. This tree can be scanned from leaves to root to get single phrases. See noun phrase grammar for details. (Appendix B)
`NounPhraseTree parseAndDecideOne( Vector<Word> words)`	First calls `parse(Vector<Word> words)` and then selects and returns the best alternative syntax tree.

MorphologicalAnalyzer*	
`Word analyzeWord( String wordText )`	Morphologically analyzes the given word string and returns a `Word` object. Uses Zemberek for morphological analyzing, and uses external dictionaries (to get root) in case Zemberek doesn’t have the root of the word in its root dictionary. Zemberek may return multiple morphological alternatives for a given word, that’s why the `Word` object contains a list of `ZemberekMorphology` objects.
`String askExternalDictionary( String wordText)`	Asks external dictionaries (Zargan,TDK) to find the root of a given wordText.

ResultsSectionMiner	
`mineResults( resultsText, findingsList )`	Returns a list of metioned associations with findings, mentioned in the result text. Also extracts other information such as “6 ay sonra gelsin”, “Normaldir” related to the report.

FindingsResultsAssociator	
`associateFindingsResults( sentenceText, findingsList )`	Finds the mentioned findings in `sentenceText`, to associate with findings in Findings section.

ConsultationSuggestionFinder	
`findConsultationSuggestion( sentenceText )`	Returns when the patient should visit doctor again if specified

ExternalWordQueryManager	
`queryAllDetails( word )`	Given `word`, ask information (on synonyms, root, correct spelling, usage in phrases (“dansite” -> “parazitel dansite”), whether the word is a medical term, etc.) from Zargan, TDK Dictionary, or other dictionaries. The results are persistently cached. Returns detailed information.
`querySynonyms( word )`	Returns synonym information
`queryPhraseUsage( word )`	Returns all different usages of word
`queryCorrectSpelling( word )`	Returns correct spelling of the word (if Zemberek does not have the root of a word in its root-dictionary, it cannot separate its suffixes. We ask external sources “çekilmiş” words, and get the “suggested” word as the root, injecting it into Zemberek, to make it separate suffixes)

ReportNormalityFinder	
`findReportNormality( sentenceText )`	Returns if the normality of the report is mentioned

* Marked classes are already implemented in the prototype.

4.3.4. Language Processing Components Explanations and Rules

Our components are designed to extract information from a sentence like in the example below:

Step 1:

Sağda 6 mm çapında kistik, solda ise areola arkasında 6x3 mm boyutlarında solid kitleler vardır.

Step 2:

Sağda 6 mm çapında kistik, solda **ise** areola arkasında 6x3 mm boyutlarında solid kitleler vardır.

ignored&removed

Step 3:

Sağda 6 mm çapında kistik, solda areola arkasında 6x3 mm boyutlarında solid kitleler **vardır.**

predicate

Predicate Meaning: Existency: Existant, Normality: Unspecified

Step 4:

Sağda **6 mm çapında** kistik, **solda** **areola arkasında** **6x3 mm boyutlarında** solid kitleler **vardır.**

LM LM LM LM LM predicate

**L/M: Location/Measurement Group*

Step 5:

Sağda	**6 mm çapında**	**kistik**,	**solda**	**areola arkasında**	**6x3 mm boyutlarında**	**solid kitleler**	**vardır.**
LM	**LM**	**WQ**	**LM**	**LM**	**LM**	**WQ**	**predicate**

Compound What/Quality Group: kistik, solid kitleler
Separated Findings:
Finding1: kistik kitleler
Finding2: solid kitleler

** L/M: Location/Measurement Group*
** W/Q: What/Quality Group*

Step 6:

Separated Findings:
Finding1: kistik kitleler
L: Sağ
M: 6 mm “çap”
Finding2: solid kitleler
L: Sol
L: Aerola arkası
M: 6x3 mm “boyut”

The designed classes and detailed algorithm and rules for applying this idea can be found below.

4.3.4.1. SectionSentenceSeparator

This component separates the given the paragraph of the findings section of a report into its sentences. It first separates a section into sentences according to full stops. Sentences which have “:” at the end, and the text inside a pair of “( )” will be ignored here.

Then it separates the sentences into multiple semantic sentences (every “semantic” sentence has only one verb or verb phrase). It uses punctuation and verbals for this step. For example “;” is mostly used to separate a sentence into two different, but related sentences.

Ex: “Mukozal doku değerlendirilmiş, peristaltik hareketler ileride incelenecektir.”

Result:

“Mukozal doku değerlendirilmiş” (semantic sentence 1)

“peristaltik hareketler ileride incelenecektir.” (semantic sentence 2)

4.3.4.2. VerbFinder

This component checks last words of a given sentence to find predicate candidate, as Turkish sentences have their predicates at their ends.

First; the last word will be taken as the predicate, and then the previous ones will be tried to be joined to this word from the left, to catch a predicate of multiple words.

At each step, the current predicate candidate will be checked from internal and external dictionaries (via External Word Query Manager) to find out if it is a verb. The longest predicate candidate will be the result of VerbFinder component.

The predicate candidate may be a verbal (fiilimsi) instead of a full verb, as SectionSentenceSeparator separated the text into logical semantic sentences.

Ex: “Sağ memede dansitenin artışı normal değildir.”

Result:

```
VERB_RANGE: 4-5
```

4.3.4.3. VerbMeaningFinder

This component finds the meaning of a predicate (in the groups of normality, existence and sometimes quality).

The predicate is first checked in our internal dictionaries (positive verb rooted, negative verb rooted, positive noun rooted, negative noun rooted dictionaries which have been manually created before) for meaning. If internal dictionaries do not contain the predicate, it is asked against external sources (via External Word Query Manager) to find synonyms that may exist in our dictionary.

If the result is reached through external sources, the predicate is added to our internal meaning dictionary for later use.

There are two cases. In the first case, the predicate has a verb root, which will give us existence information. In the second case, the predicate has a noun root (“normaldir”, “doğaldır”, “difüzdür”, ”büyüktür”), which may give information on

1. normality/abnormality
2. existence / nonexistence (and also contain quality to be associated with the findings in the source sentence)

Ex: “Sağ memede dansitenin artışı normal değildir.”

Result:

```
Existence: yes
Normality: no
Quality: none (here)
```

4.3.4.4. SentencePartsGrouper

This component tags parts of the sentence as groups of “location-measurement” or “what-quality”. These groups can be easily identified by the help of the rules of Turkish.

The predicate of a sentence will not be passed to this component.

We define a crush element as a word with suffixes “-e, -a” (yönelme hali), “-de, -da” (bulunma hali), “-den, -dan” (ayrılma hali), “-(y)le -(y)la” (birliktelik durumu – but only as a suffix, not the distinct word “ile”), a verbal (fiilimsi – such as “olup”, “olarak”), some predefined conjunction words (such as “için”).

First the sentence will be scanned from left to right, looking for crush elements. When a crush element is found, the part of the sentence to the left of the crush element (up to a previous crush element or the beginning of the sentence) will be processed depending on the type of the crush element. This part will be referred as “the part associated with the crush element”. Depending on the crush element:

- If the crush element is a suffix of “-e, -a”, “-(y)le, -(y)la”, or a conjunction word, then the associated part of the crush element will be tagged (ignored).
- If the crush element is a suffix of “-den, -dan”, there are two cases depending on the leftmost word to the right of the crush element. If it is a number, no action will be taken (we will continue processing the associated part of the crush element), otherwise, the associated part of the crush element will be ignored as in “-e, -a” rule.
- If the crush element is a verbal, the verbal will be ignored from the sentence, and the associated part of the verbal will be joined to the right of the verbal.
- If the crush element is a suffix of “-de, -da”, the associated part of the crush element will be scanned from right to left. Initially, the rightmost word will form the “location + measurement” group. In each step, the rightmost word to the left of the current “location+measurement” group will be tried to be joined to the group on its left. This new candidate group will then be passed to Noun Phrase Parser to check if it forms a valid noun phrase. Finally, the longest valid rightmost noun phrase to the left of the crush element will be tagged as “location+measurement” group, and the remaining part of the associated sentence part will be tagged as a “what+quality” group.

After the last crush element, the remaining part of the sentence (on the right) will be tagged as a “what+quality” group.

Examples:

Sağda sigmoid sinus açıktır
L+ M W + Q predicate

Her iki memede dağınık fibroglandüler dansiteler vardır.
L + M W + Q predicate

Non-dominant olduğu için yavaş akıma bağlı teknik nedenlerle
ignored ignored ignored

görüntülenememiş olabilir.
predicate

4.3.4.5. LocationOrMeasurementAnalyzer

This component decides whether the given phrase points a location, a measurement or is irrelevant. All “Dolaylı tümleç”s are analyzed in this component.

There are exactly five cases, illustrated below:

“Sol memede” -> Location, name: “sol meme”

“saat 6 hizasında” -> Location, name: “saat 6 hizası”

“aeroladan 2 cm uzaklıkta” -> Location, name: “aerola”, distance: 2, distance_unit: “cm”

“7x5 mm boyutunda” -> 2 measurements; measurement: 7, measurement unit: “mm”, type: “length1”; measurement: 5, measurement unit: “mm”, type: “length2”

“3 mm çapında” -> 1 measurement; measurement: 3, measurement unit: “mm”, type: “diameter”

“(yapılan) US incelemesinde” -> irrelevant (“yapılan” crush element was ignored from the sentence). There is a finite set of phrases: “ * sırasında”, “bunun dışında”, “incelemesinde”, “ile karşılaştırıldığında”, “ * esnasında”

“kemik iliği difüz (olarak) baskılanmakta” -> (“olup” crush element was ignored from the sentence). Here, to the left of “-de, -da” there is a verbal. We ignore the verbal, and return “kemik iliği difüz” as “this should be tagged as ‘what-quality’ ”

4.3.4.6. FindingsAnalyzer

This component parses the given phrase into a list of findings (“ne”) and their qualities (“nasıl”). Most of the time, the given phrase is only a complex noun phrase consisting of quality and finding name information. It returns a list of finding names and their related qualities. More than one finding may be found from a single phrase due to nature of complex noun phrases.

There can be multiple noun phrases in the given phrase. If the last phrase is a single word, then it will be marked as a “quality” associated with all other findings to be found in the phrase. (a special case: if the last phrase is a single word and is in a special set of words “normal”, “anormal”, “subnormal”, it won’t be marked as a quality but as a normality / abnormality specifier for the findings)

Ex: Kemik dokusu ve kemik iliği difüz (olarak …) -> “kemik dokusu”, “kemik iliği”, “difüz”

Ex: Kemik dokusu ve iliği difüz (olarak …) -> “kemik dokusu ve iliği”, “difüz”

Ex: Kemik dokusu ve kemik iliği -> “kemik dokusu”, “kemik iliği

Parsing in this component is done from left to right similar to LocationMeasurementAnalyzer. In each step, first the end of the input is checked if it contains a conjunction word (comma, “ve”, “ile”, “ya da”, “veya”). If there is, it is removed. Then, the rightmost side of the candidate noun phrase is fixed to the end of input. The leftmost side of the candidate noun phrase is moved to left in steps, until the beginning of the input is reached. The longest valid noun phrase will be marked as one item, and the same process will be applied to the rest of the input.

Ex:

```
Kemik dokusu ve kemik iliği difüz
                            ^   ^  valid noun phrase

Kemik dokusu ve kemik iliği difüz
                      ^         ^  invalid noun phrase
Kemik dokusu ve kemik iliği difüz
                ^               ^  invalid noun phrase
Kemik dokusu ve kemik iliği difüz
      ^                         ^  invalid noun phrase

Kemik dokusu ve kemik iliği difüz
^                               ^  invalid noun phrase
```

The longest valid noun phrase is “difüz”, so it is marked as one item, removed from the input, and the process is repeated. This technique relies on the fact that our noun phrase parser can only parse single noun phrases (“kemiğin dokusu ve iliği”, “kemiğin dokusu”, but not phrases containing distinct items as in “kemik dokusu ve kemik iliği” or “doku ve ilik”).

The noun phrase parser will return each item as a list of simple noun phrases.

Ex: Midenin konturları, pasaj ve peristaltizmi ile mukozal rölyefi

“midenin konturları”, “midenin pasajı”, “midenin peristaltizmi” “midenin mukozal rölyefi” are simple noun phrases parsed from the initial long noun phrase.

Ex: Kistik ve solid lezyon bulunmuştur.

2 findings: quality: kistik what: lezyon ; quality: solid what: lezyon

Ex: Kistik solid lezyon bulunmuştur.

1 finding: quality: kistik, solid what: lezyon

Ex: Kistik ve solid mide lezyonu bulunmuştur.

2 findings: quality: kistik what: mide lezyonu ; quality: solid what: mide lezyonu.

4.3.4.7. SentenceFindingSeparator

This component will mine findings from a single logical semantic sentence, and returns them as a list.

First, the sentence is sent to VerbFinder to find the position of the predicate. Predicate may be composed of one or more words. Once the predicate is found, it is separated from the sentence body. The predicate is passed to VerbMeaningFinder to get normality and existence. VerbMeaningFinder may also return a single quality to associate with all findings extracted from the sentence. If there is no normality / existence information retrieved from the VerbMeaningFinder, the sentence is no further processed.

Then, the rest of the sentence is sent to SentencePartsGrouper to tag parts of the sentence “location-measurement” or “what-quality” groups. These groups are then processed in left to right order. Each group is passed to LocationOrMeasurementAnalyzer or FindingAnalyzer depending on group type.

The locations are attached to findings and the finding list is returned.

4.3.4.8. FindingsSectionMiner

This component calls SectionSentenceSeparator and SectionFindingSeparator to extract the distinct findings mentioned in each of the semantic sentences of a given of the findings section of a report (Every “semantic” sentence has only one verb.). The result is a list of findings. Every finding has properties such as finding type (“ne”), quality list, location list, measurement list, and information on normality and existence.

Ex: "Sol memede kitle, sağ memede dansitenin artışı normal değildir."

Result:

```
Existence: yes
Normality: no
What     : kitle artışı
Qualities: []
Locations: sol meme
Measurement: []

Existence: yes
Normality: no
What     : dansitenin artışı
Qualities: []
Locations: sağ meme
Measurement: []
```

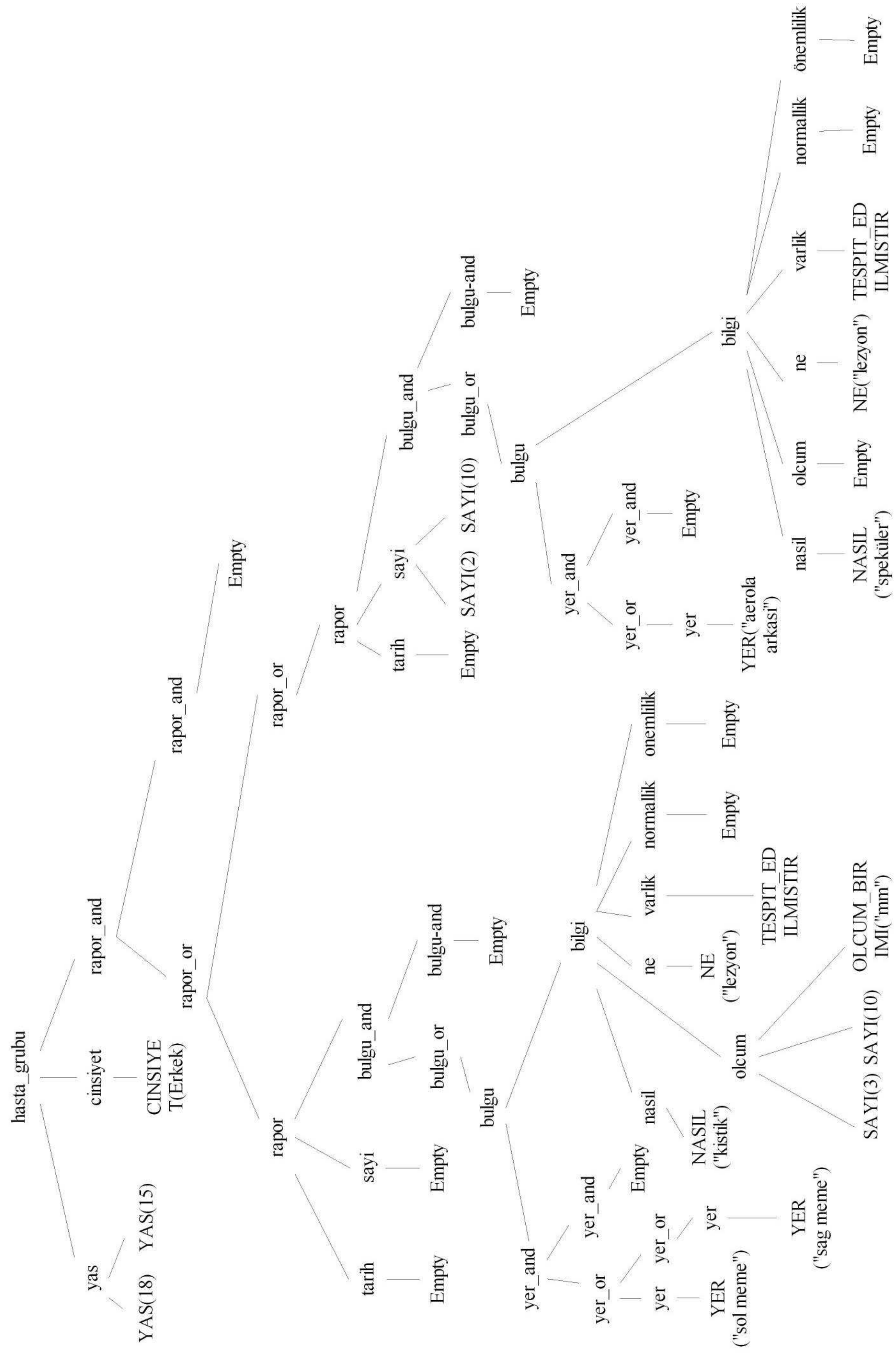
4.3.5. Statistical Querying Components Explanations and Rules

Mining by itself is not enough for an application like RadioRead. There is an important need to access acquired information later.

For querying of accumulated information, we have designed a language grammar. This grammar allows both SQL queries and Turkish sentences to be generated from the same syntax tree.

The main component of statistical queries is the `hasta_grubu` node in a query syntax tree. All statistical queries contain one or more `hasta_grubu` nodes to determine targets and some extra information such as the number of columns to be shown on the bar-graph for the `measurement_graph_query`. In this section, we will show a sample hasta_grubu node, its pseudo-SQL and Turkish counterparts. The Turkish counterpart will be shown to the user to validate his ideas.

Parse tree (rooted on `hasta_grubu` node):



Its Turkish counterpart (to be shown to doctor to validate his selections in GUI):

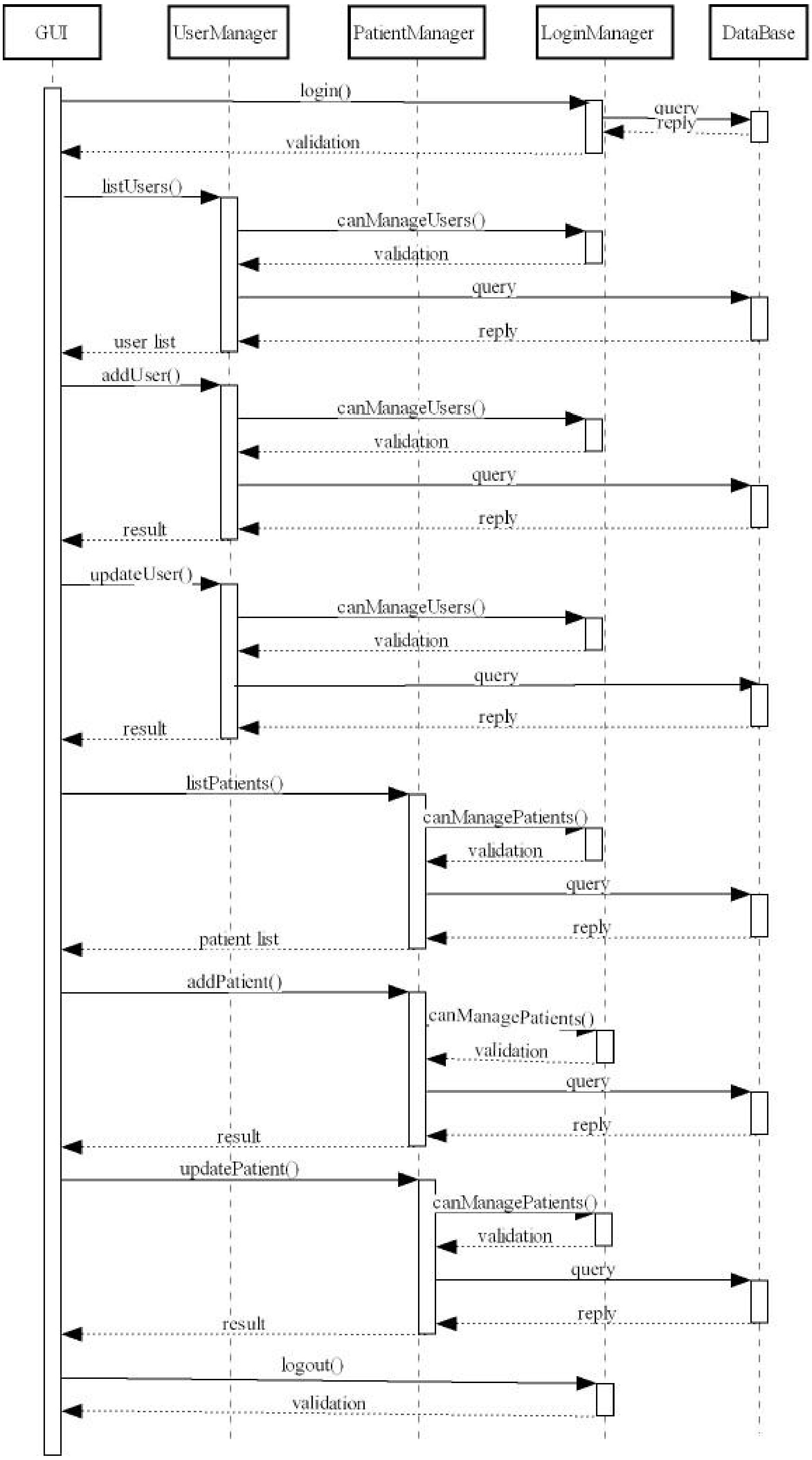
```
18 ile 25 yaş arası kadın hastalardan;
(sağ memesinde VEYA sol memesinde
3 ile 10 mm arasında kistik lezyon tespit edilmiş)
VEYA
(en az 2 en fazla 10 raporunda;
areola arkasında spekȕler lezyon tespit edilmiş)
kişiler
```

Its pseudo-SQL counterpart (to be executed on database server):

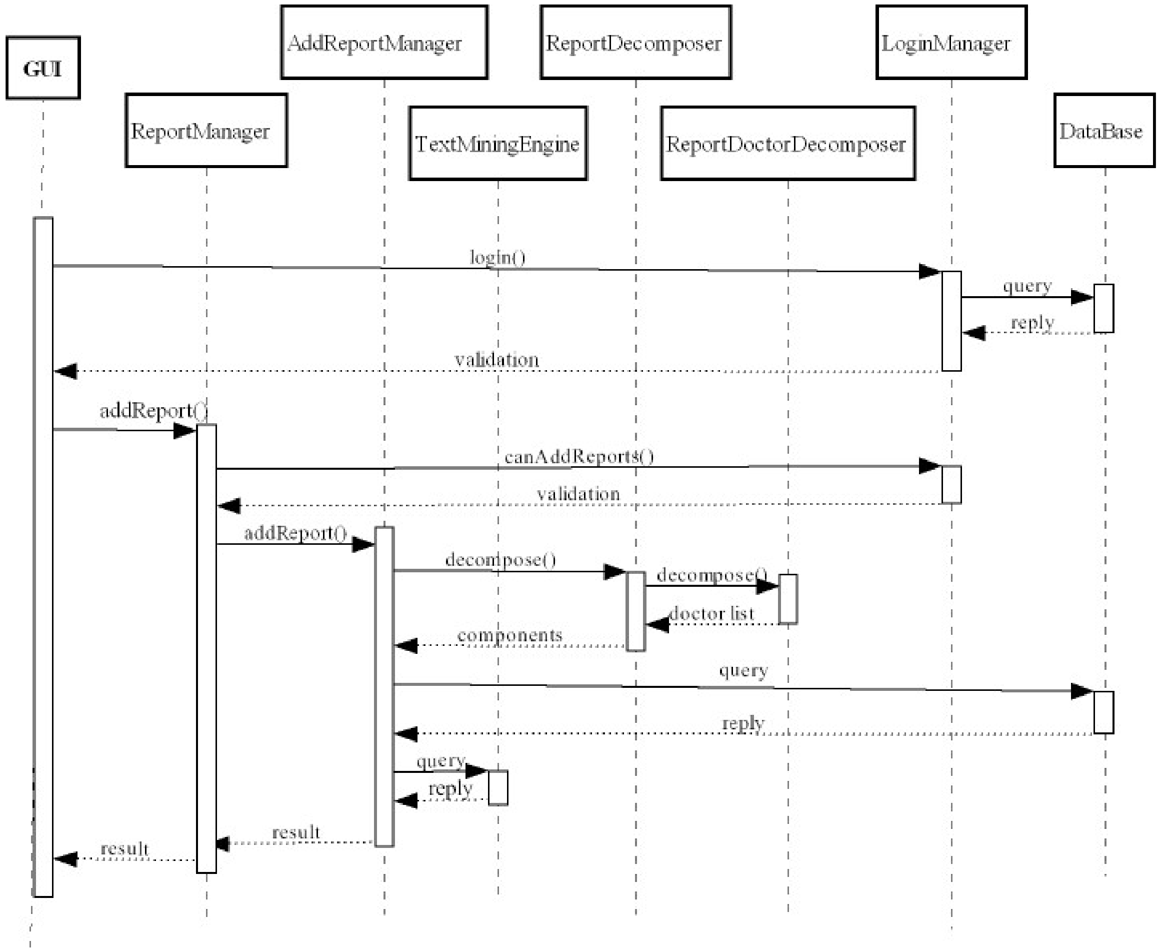
```
SELECT Hasta.hasta_id
WHERE
(Hasta.yas>18 AND Hasta.yas<25 AND Hasta.cinsiyet="kadın")
AND
(
    (
        Rapor1.hasta_id = Hasta.hasta_id
        AND
        (
            Bulgu1.rapor_id = Rapor1.rapor_id
            AND Bulgu1.ne="lezyon"
            AND Bulgu1.nasil="kistik"
            AND (Bulgu1.yer="sol meme" OR Bulgu1.yer="sağ meme")
            AND (Bulgu1.olcum>3 AND Bulgu1.olcum<10
                AND Bulgu1.olcum_turu="mm")
            AND Bulgu1.varlik=TRUE
        )
    )
    OR
    (
        Rapor2.hasta_id = Hasta.hasta_id
        AND
        (
            Bulgu2.rapor_id = Rapor2.rapor_id
            AND Bulgu2.ne="lezyon"
            AND Bulgu2.nasil="spekȕler"
            AND Bulgu2.yer="areola arkasi"
            AND Bulgu2.varlik=TRUE
            AND HAVING COUNT(Bulgu2.*)>2 AND COUNT(Bulgu2.*)<10
        )
    )
)
```

4.4. Sequential Diagrams

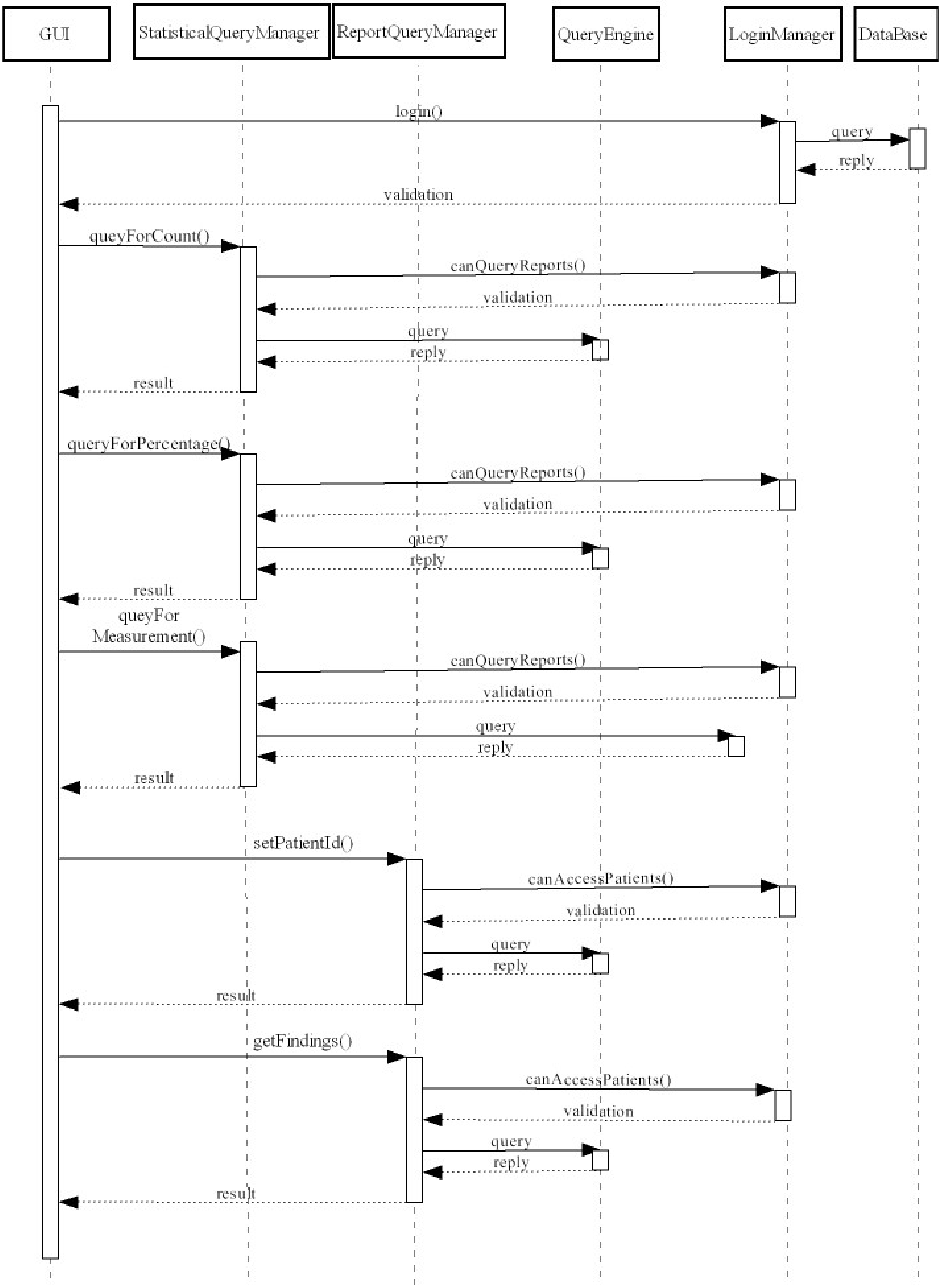
4.4.1. Sequential Diagram for UserManager, PatientManager, LoginManager



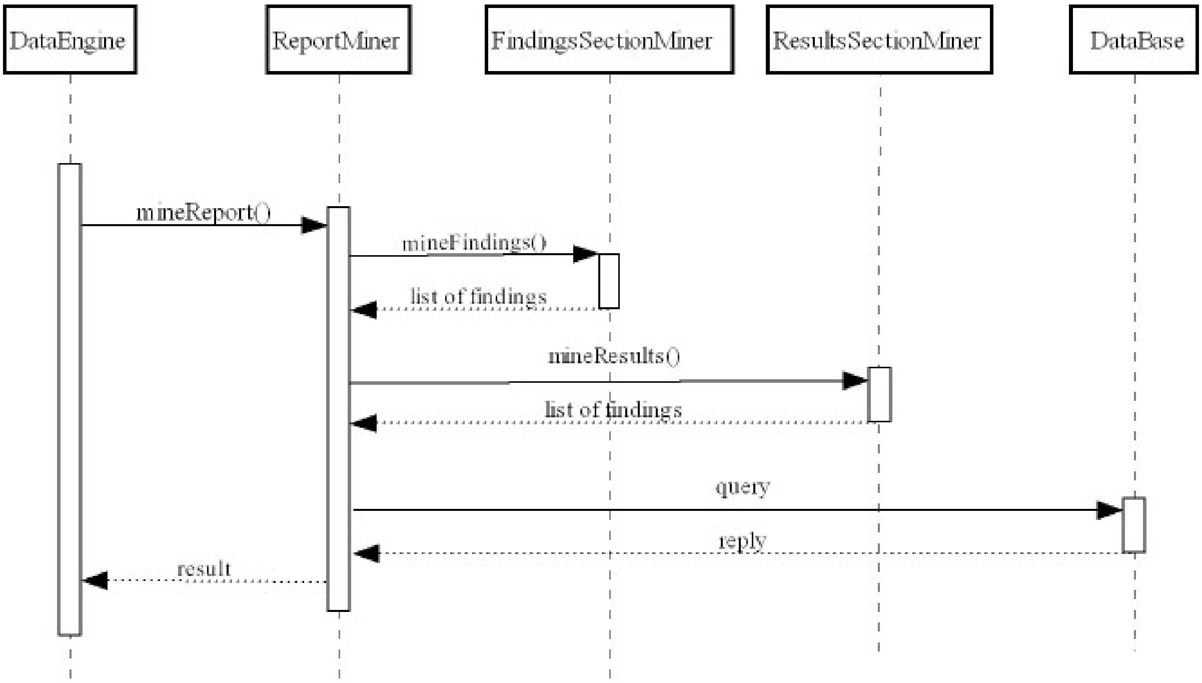
4.4.2. Sequential Diagrams for AddReportManager, ReportManager and LoginManager



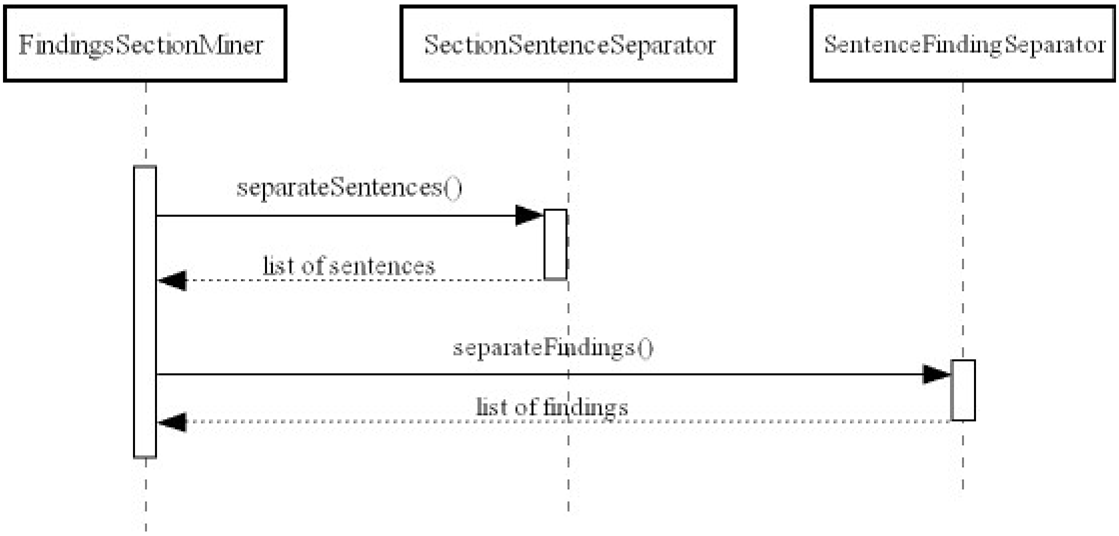
4.4.3. Sequential Diagrams for StatisticalQueryManager and ReportQueryManager



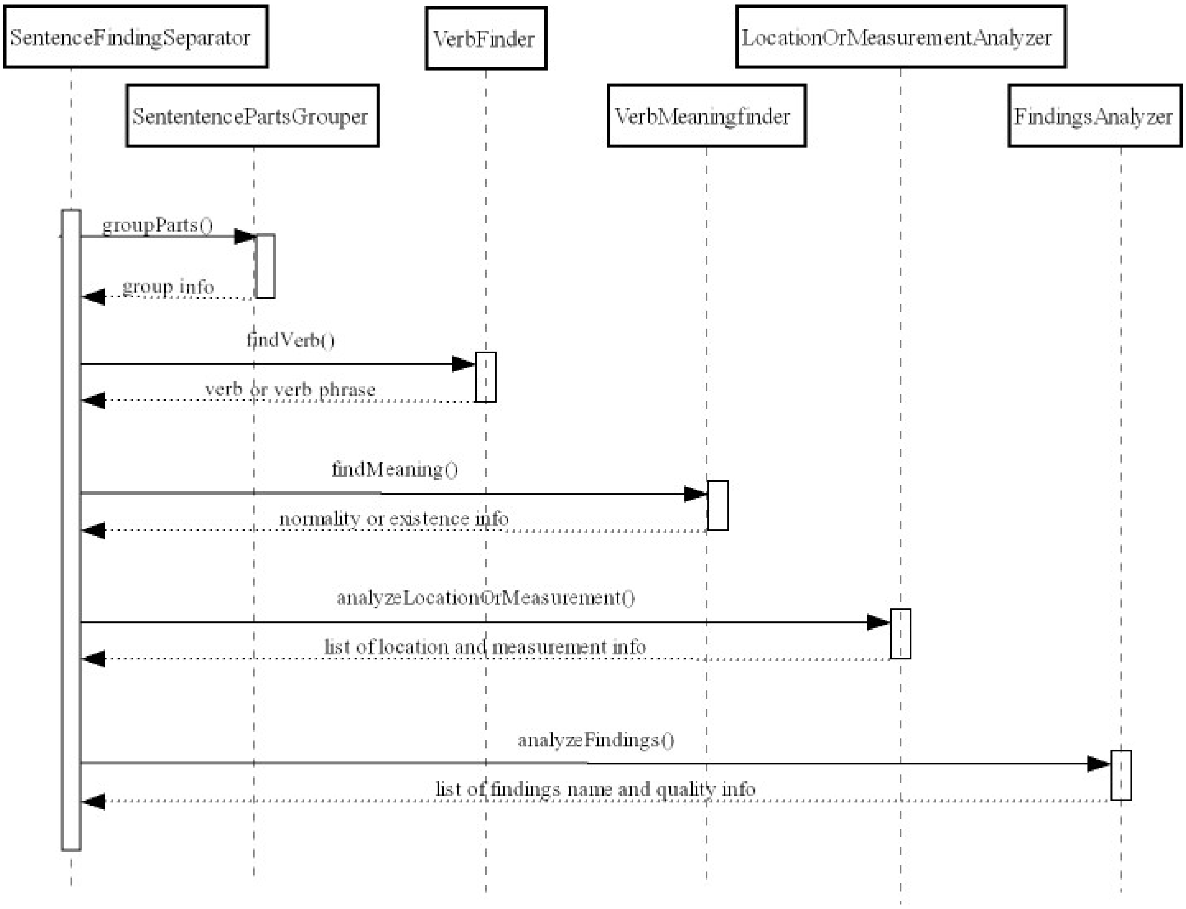
4.4.4. Sequential Diagrams for ReportMiner, FindingsSectionMiner and ResultsSectionManager



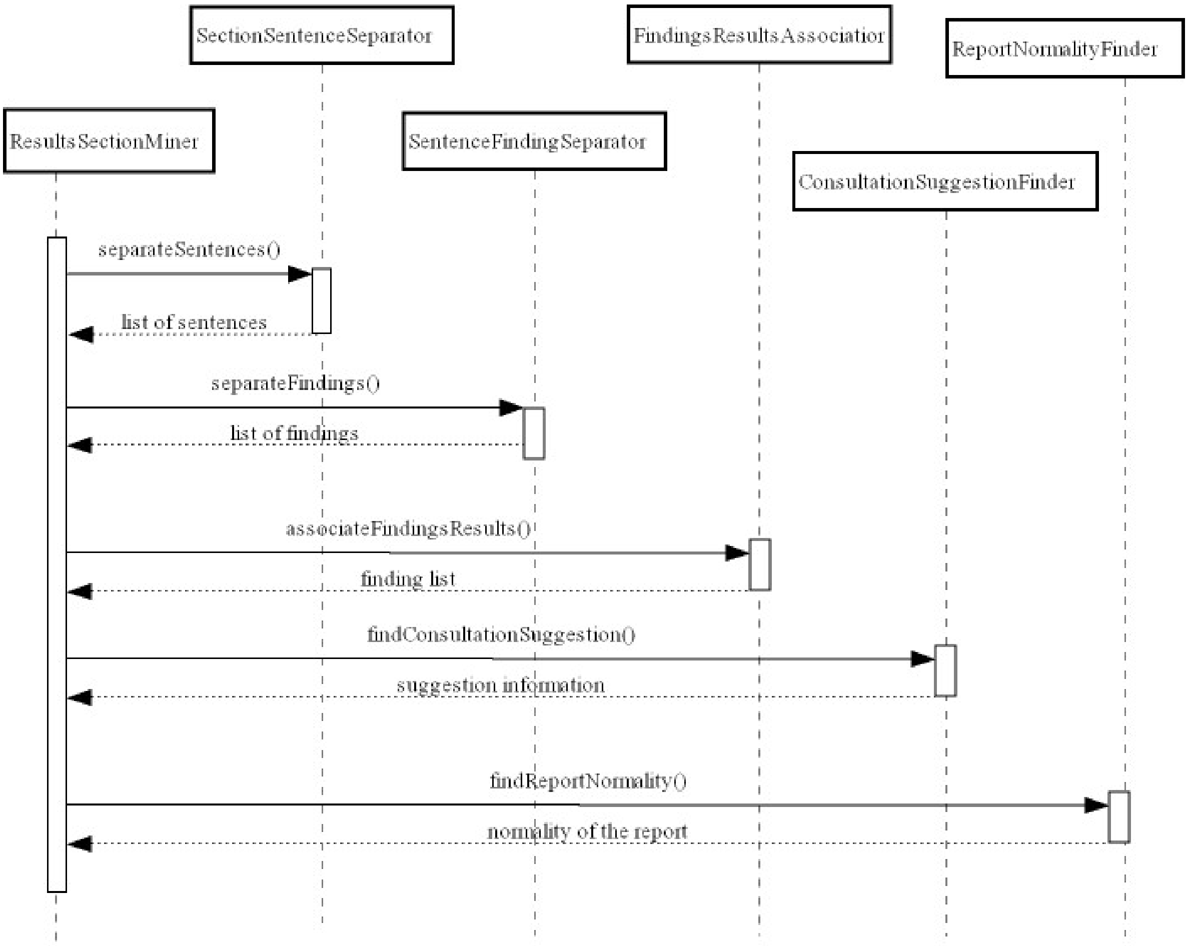
4.4.5. Sequential Diagrams for FindingsSectionMiner



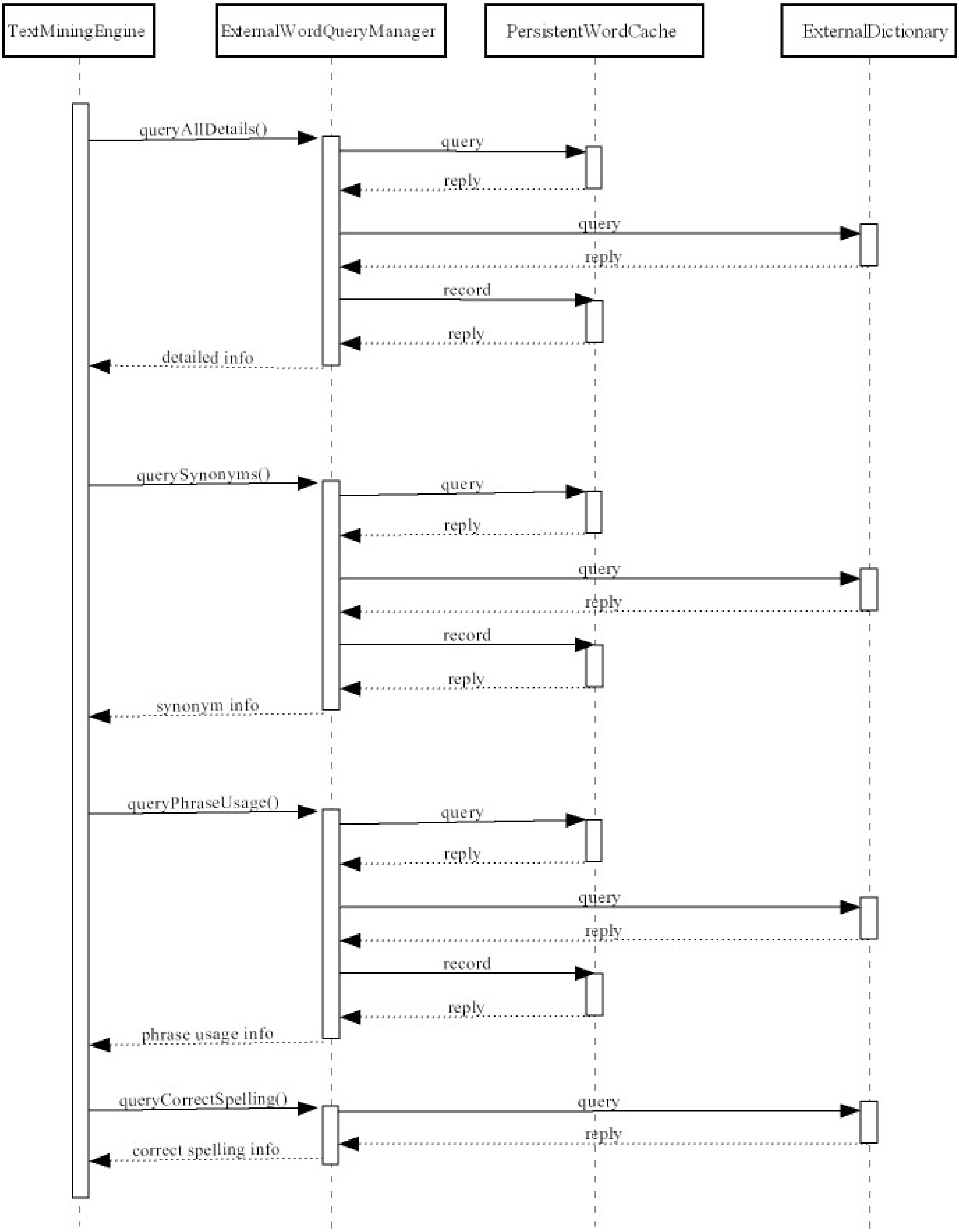
4.4.6. Sequential Diagrams for SentenceFindingSeparator



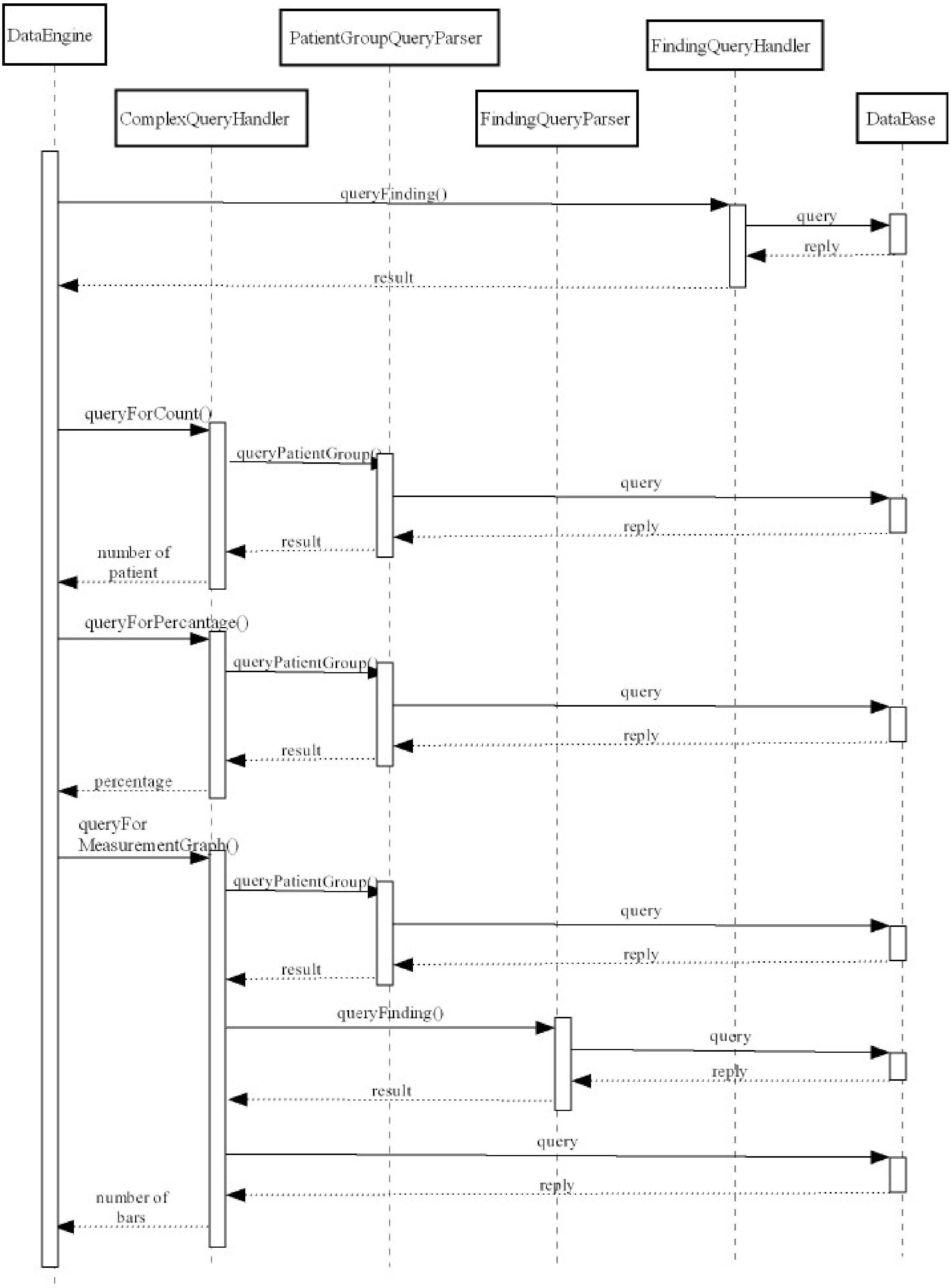
4.4.7. Sequential Diagrams for ResultsSectionMiner



4.4.8. Sequential Diagrams for ExternalQueryManager



4.4.9. Sequential Diagrams for ExternalQueryManager

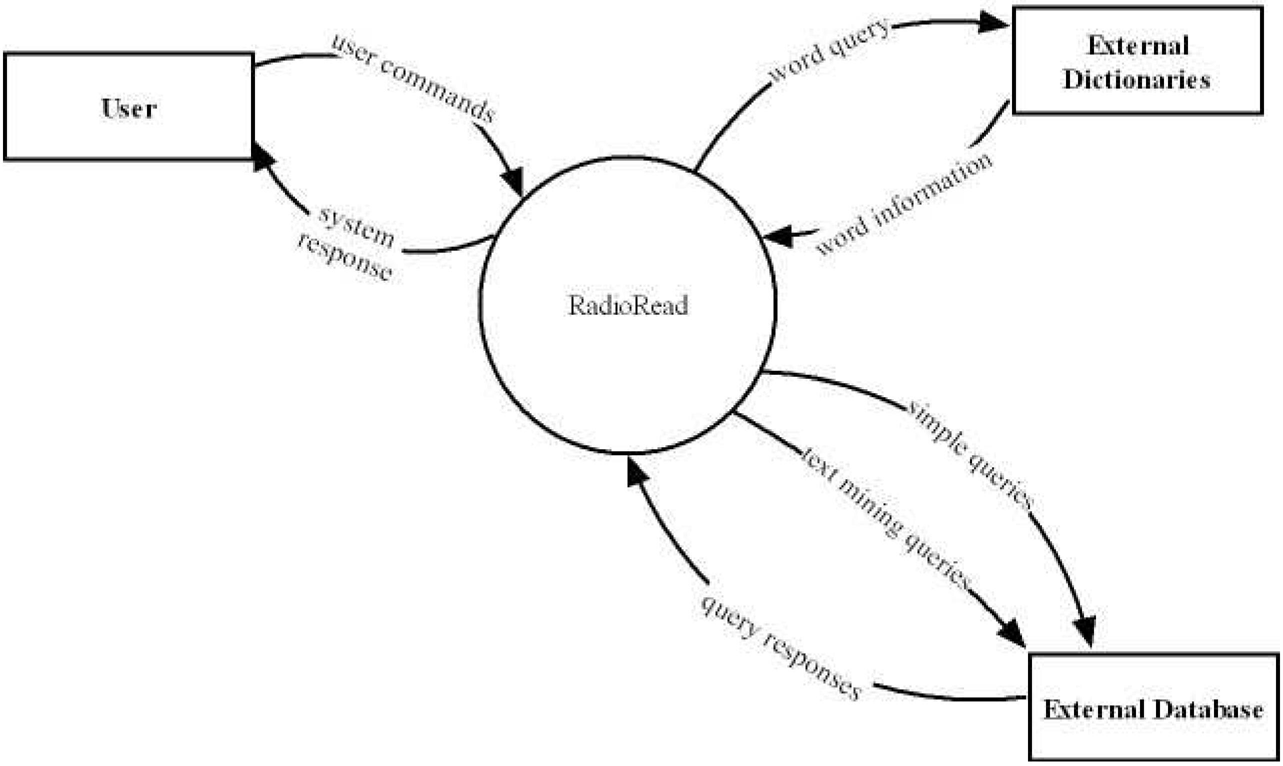


5. Modelling

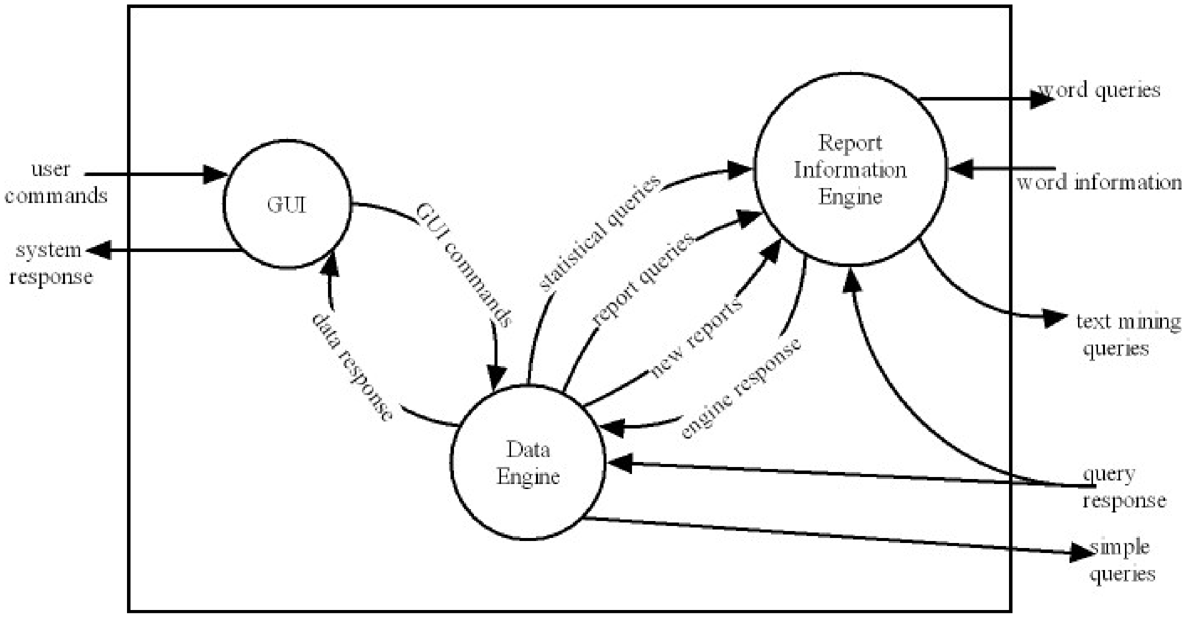
5.1. Functional Modelling

5.1.1. Data Flow Diagrams

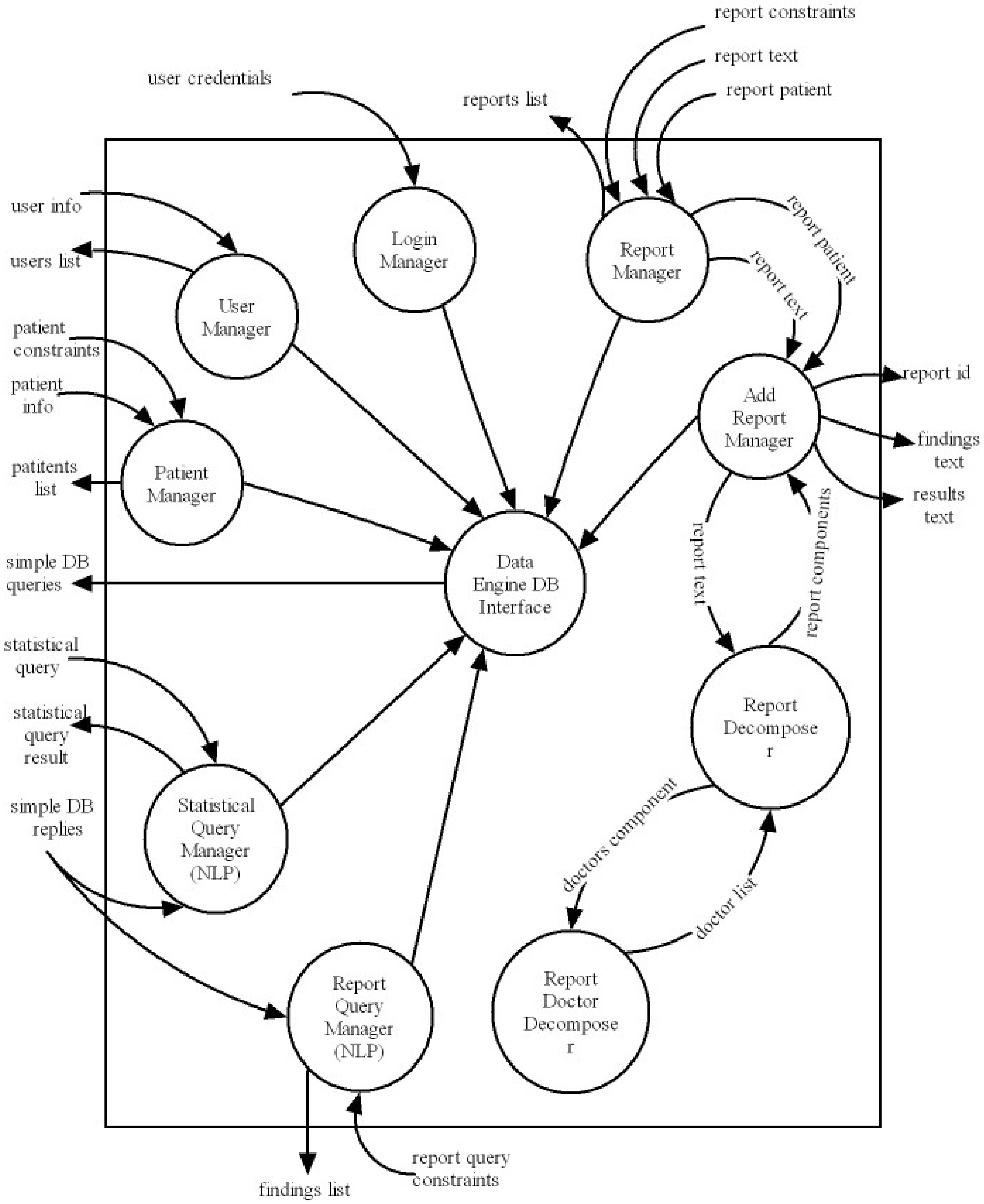
5.1.1.1. Level-0 Data Flow Diagram



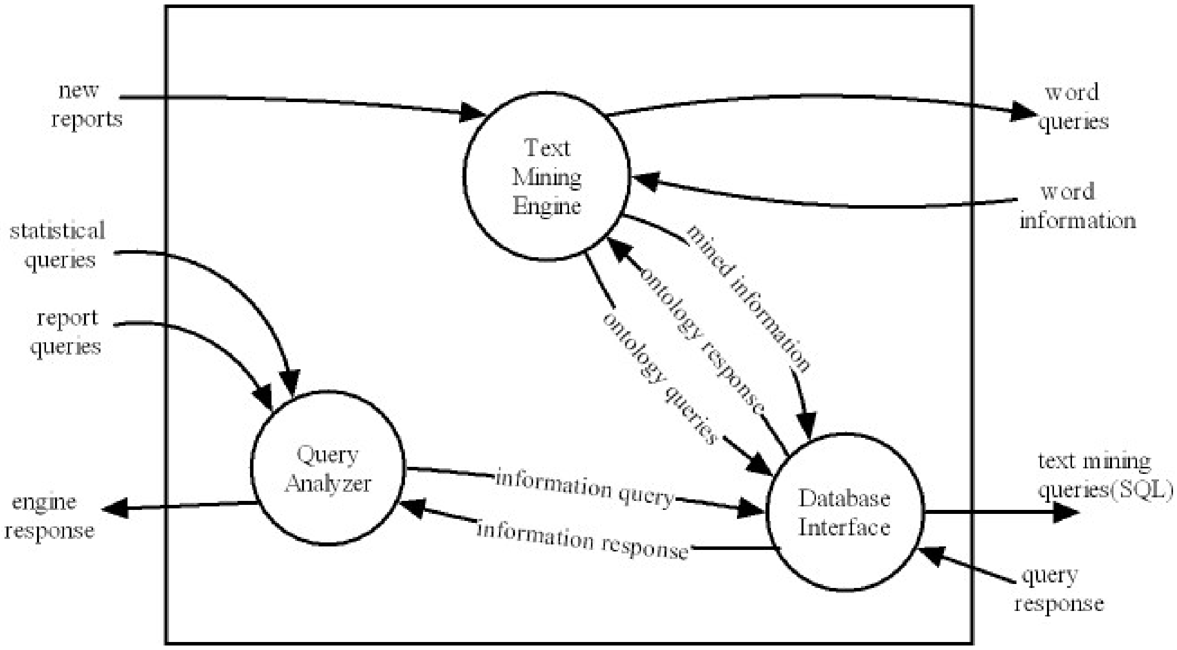
5.1.1.2. Level-1 Data Flow Diagram: RadioRead



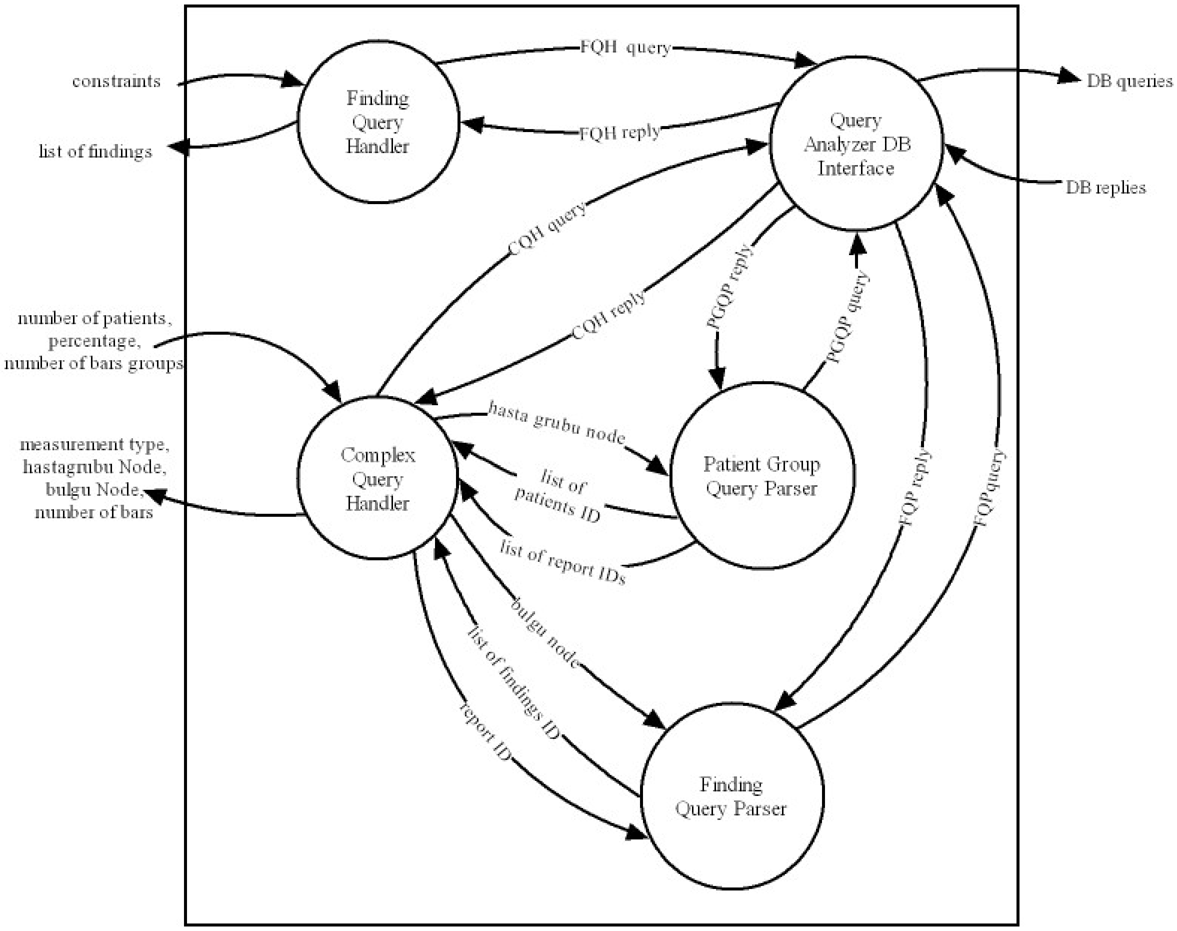
5.1.1.3 Level-2 Data Flow Diagram: Data Engine



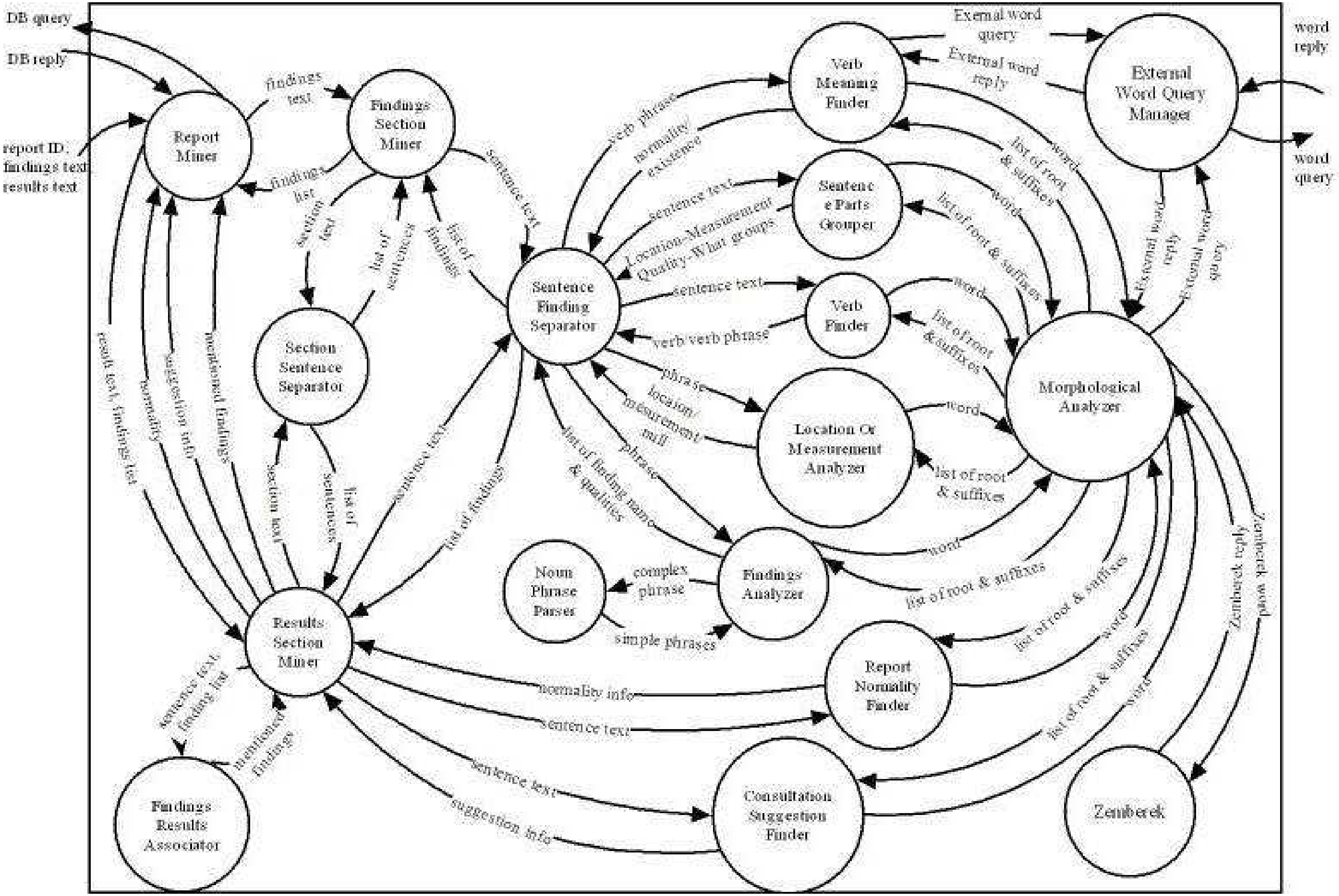
5.1.1.4 Level-2 Data Flow Diagram: Report Information Engine



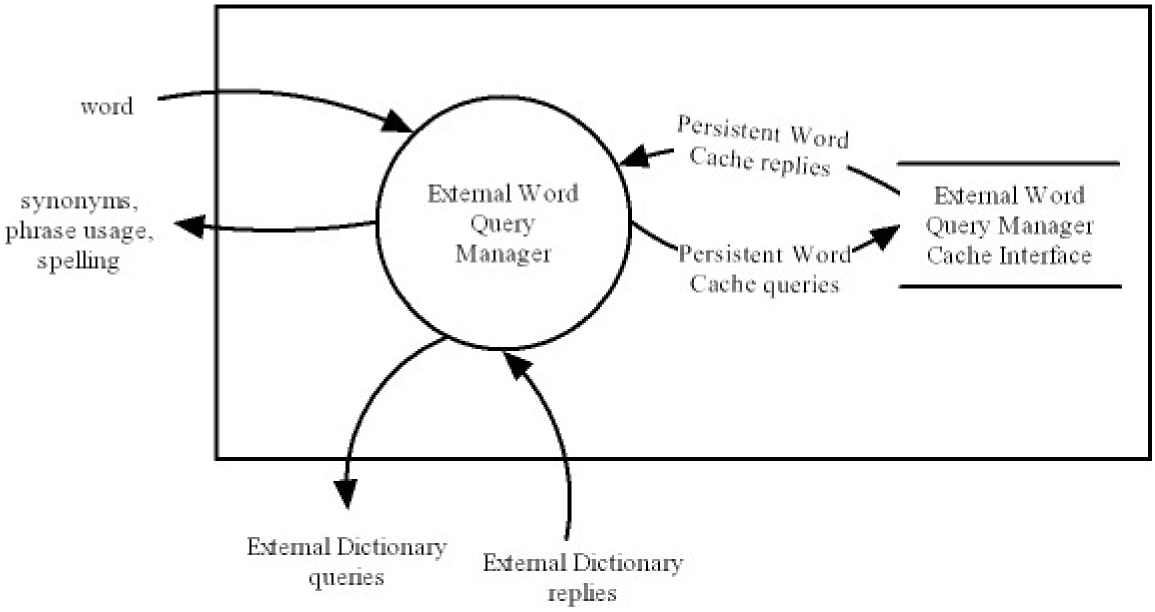
5.1.1.5 Level-3 Data Flow Diagram: Query Analyzer



5.1.1.6 Level-3 Data Flow Diagram: Text Mining Engine



5.1.1.7 Level-4 Data Flow Diagram: External Word Query Manager (Part of Morphological Analyzer)


word
synonyms, phrase usage, spelling
External Word Query Manager
Persistent Word Cache replies
Persistent Word Cache queries
External Word Query Manager Cache Interface
External Dictionary queries
External Dictionary replies

5.1.2. Data Dictionary

Name:	Word query
Where used?	Output of External Word Query Manager (Level 3: Text Mining Engine) Input to External Dictionaries (Level 0)
Description	Query that is sent to external dictionaries to get meaning information.

Name:	Simple queries
Where used?	Output of Data Engine DB Interface(Level 2: Data Engine) Input to External Database (Level 0)
Description	SQL queries that are sent to external database to get/set information which are not mined from reports, but about meta data.

Name:	Text mining queries
Where used?	Output of Database Interface (Level 2: Report Information Engine) Input to External Database (Level 0)
Description	SQL queries that are sent to external database to get/set information which are mined from reports.

Name:	Data Engine
Where used?	Level 1
Description	Internal engine that separates data depending on whether they will be sent to Report Information Engine to be text-mined or External Database to be stored.

Name:	Report Information Engine
Where used?	Level 1
Description	Internal engine that extracts information from reports and handles complex queries such that statistical and report queries.

Name:	New Reports
Where used?	Output of Data Engine (Level 1) Input to Report Miner (Level 3: Text Mining Engine) (renamed as ‘findings text’, ‘results text’ in Level 3)
Description	Original text reports that are to be mined.

Name:	Text Mining Engine
Where used?	Level 2
Description	Internal engine that extracts information from reports by using text mining techniques.

Name:	Query Analyzer
Where used?	Level 2
Description	Internal analyzer that sends a stream of simpler queries which are obtained from complex queries (statistical/report queries), and merges results.

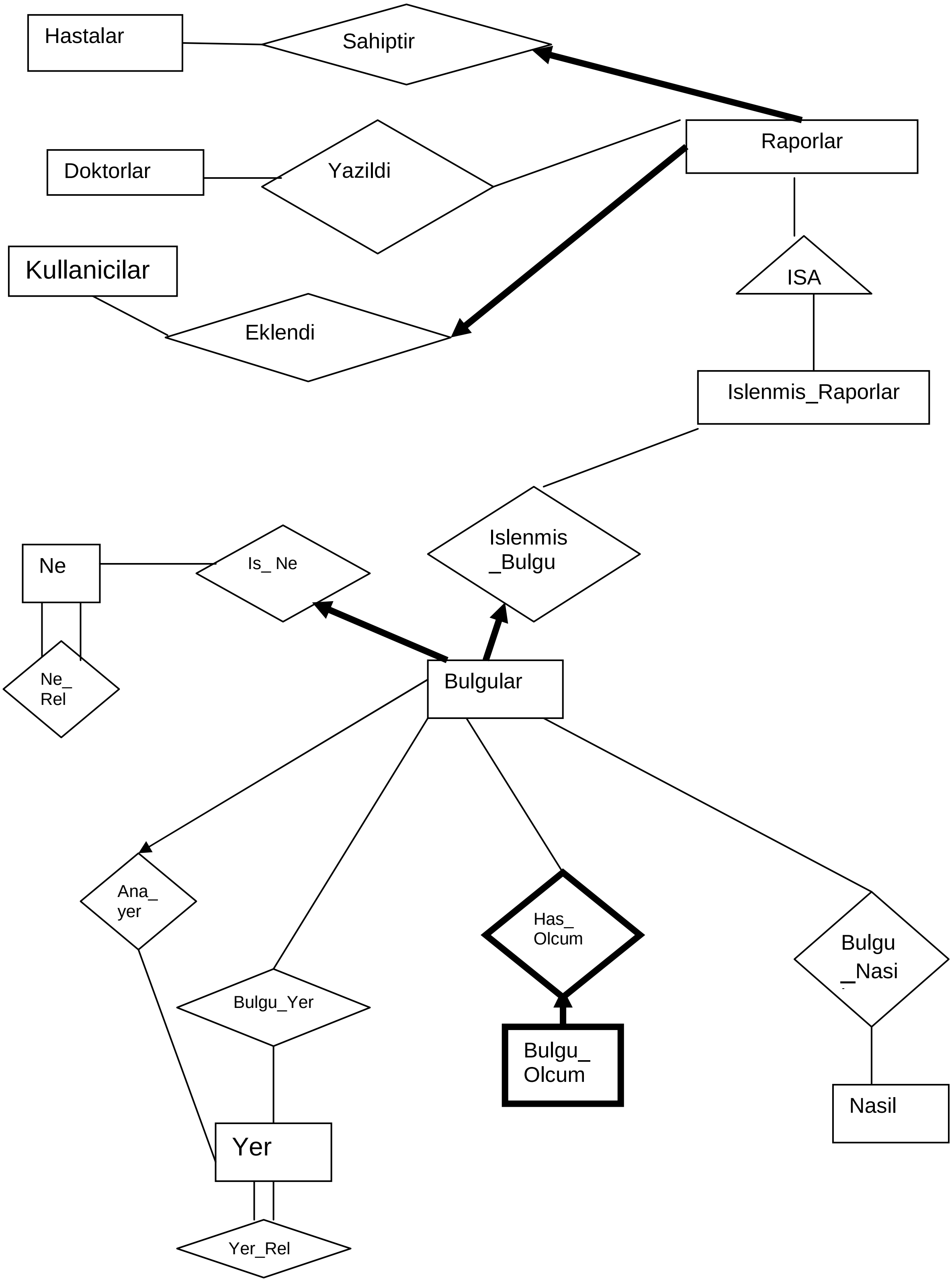
Name:	Information query
Where used?	Output of Query Analyzer (Level 2:Report Information Engine) Input to Database Interface (Level 2:Report Information Engine)
Description	Simpler queries which are obtained from complex queries.

Name:	hasta_grubu Node
Where used?	Output of Complex Query Handler(Level 3:QueryAnalyzer) Input to Patient Group Query Parser (Level 3:Query Analyzer)
Description	The root node of a hasta_grubu syntax tree. See statistical query grammar for details.

Name:	bulgu Node
Where used?	Output of Complex Query Handler(Level 3:QueryAnalyzer) Input to Finding Query Parser (Level 3:Query Analyzer)
Description	The root node of a bulgu syntax tree. See statistical query grammar for details.

5.2. Data Modelling

5.2.1. Entity-Relationship Diagrams



5.2.2. Data Descriptions

The data description part gives information about the structure of the database. We have demonstrated entities and relations without their attributes. Instead of this, attributes of entities are listed below. The underlined data are the primary-keys, and the data with stars are foreign keys.

We have 5 global tables that will be populated after doing some text mining on reports. They are “Ne”, “Yer”, “Nasil”, “Yer_Rel” (demonstrating the relation between two “Yer” records) and “Ne_Rel” (demonstrating the relation between two “Ne” records) (See Section 5.2.3).

Ne:

This entity contains information about kinds of all possible findings.

Yer:

This entity contains information about locations of all possible findings.

Nasil:

This entity contains information about qualities of all possible findings.

Hastalar:

This entity contains all necessary information about the patients. This information will be inserted to database through GUI, and they will not be text-mined. This information will be used for diagnostic purposes by doctors.

Doktorlar:

This entity contains all necessary information about the doctors that write the reports. This information is gathered from reports. This information will be used for statistical purposes.

Kullanicilar:

This entity contains all necessary information about the users. The “Kullanicilar” entity contains information about login information and access-rights. This information will then be used to categorize users into five groups: Admin, Staff-1, Staff-2, Doctor, and Statistician.

Raporlar:

This entity contains all necessary information about reports. Each report is owned by a patient, can have multiple doctor information which is written in reports and can be added by only one user. This entity contains only non-mined meta information about reports, such as title, text, date.

Islenmis_Raporlar:

This entity is a Raporlar. This entity holds the mined information about reports and separates meta information and mined information.

Bulgular:

This entity contains any finding mentioned in the findings (“Bulgular”) section of a report text. All information (normal, abnormal, existent, and non-existent) that can be extracted from the report text about a single finding is stored here.

Bulgu_Olcum:

This entity contains information about quantities of a “Bulgular” record.

Bulgu_Nasil:

This relation makes an n-to-n correspondence between “Bulgular” and “Nasil” entitites. This holds qualities of a “Bulgular” record.

Bulgu_Yer:

This relation makes an n-to-n correspondence between “Bulgular” and “Yer” entitites. This holds locations of a “Bulgular” record.

Database Tables:

Kullanicilar (user_id, access_rights, username, password, active, name)

Hastalar (patient_id, name, surname, cinsiyet, year_of_birth)

Doktorlar (doctor_id, title, name, surname)

Raporlar (report_id, patient_id*, user_id*, title, date, clinical_info, technical_info, findings, result)

Yazildi (doctor_id*, report_id*)

Islenmis_Raporlar (report_id*, sure, sure_birimi, normallik)

Bulgular (bulgu_id, report_id*, ne_id*, yer_id*, normal, var, sonucta_geciyor)

Bulgu_Yer (bulgu_id*, yer_id*, uzaklik_olcum, uzaklik_birim)

Bulgu_Olcum (bulgu_olcum_id, bulgu_id*, olcum, olcum_birimi, tur)

Bulgu_Nasil (bulgu_id*, nasil_id*, sonuctan)

Yer (yer_id, isim)

Yer_Rel (birincil_yer_id*, ikincil_yer_id*)

Nasil (nasil_id, isim)

Ne (ne_id, isim)

Ne_Rel (birincil_ne_id*, ikincil_ne_id*)

Raporlar

report_id*	
patient_id*	
User_id*	
Title	The title text of the report
Date	Date of the report
clinical_info	The text in the Clinical Information section
technical_info	The text in the Technical Information section
Findings	The text in the Findings section
Result	The text in the Results section

Islenmis_Raporlar

report_id*	
Sure	Quantity of the advised time for next consultation. Can be NULL if not specified in the Results section of the report
Sure_birimi	Unit of the time
Normallik	True / False / NULL – Holds whether normality / abnormality is specified in the Results section of the report

Bulgular

bulgu_id	
report_id*	
ne_id*	
yer_id*	Holds the primary location of this finding. Bulgu_Yer table holds secondary locations
Normal	True / False / NULL – holds whether this finding is specified as normal or abnormal or not specified in normality
Var	True / False / NULL – holds whether this finding is specified as existent or non-existent or not specified in existence
sonucta_geciyor	True / False – Holds whether this finding is also referenced in the results section of the report

Bulgu_Yer

bulgu_id*	
yer_id*	
uzaklik_olcum	The quantity of the distance
uzaklik_birim	The unit of the measurement

Bulgu_Olcum

bulgu_olcum_id	
bulgu_id*	
Olcum	The quantity of the measurement
olcum_birimi	The unit of the measurement
Tur	The kind of the measurement (i.e. “çap”, “hız”, “uzunluk”, “boyut”)

Bulgu_Nasil

bulgu_id*	
Nasil_id*	
Sonuctan	True/False. Holds whether this quality is gained only from the “Results” section of a report

Yer

yer_id	
Isim	Name of the quality (i.e “meme”, “sol meme”, “areola”)

Yer_Rel

birincil_yer_id*	
ikincil_yer_id*	

Nasil

Nasil_id	
Isim	Name of the quality (i.e “dens”, “heterojen”, “solid (lezyon)”)

Ne

Ne_id	
Isim	Name of the finding (i.e “duktal ektazi”, “patern”, “lezyon”)

Ne_Rel

birincil_ne_id*	
ikincil_ne_id*	

5.2.3. Database and Ontology

In medical terminology all terms are related to each other in some way. These terms can be relations between locations such as the relations between “aksiler kuyruk”, “sol meme”, “meme” and “üst gövde”. These relations can be seen in every part of the terminology, among location names, diagnosis, treatments, qualities, drugs. This kind of a structure is called ontology and can be represented successfully by a graph structure in software.

SNOMED database provides such an ontology for English medical terms. It contains all the relations between diagnosis, locations and treatments. SNOMED is popular in medical software, both in text mining and hospital automation systems in English speaking countries.

As we study on text mining on Turkish medical reports, we can not directly use an English medical ontology like SNOMED. But it is still possible to create a Turkish medical ontology based on sample reports.

We have designed our database to support an ontology structure, such that every location can be bound to a "parent" location (for example parent of "sol meme" is "meme", and parent of "meme" is "üst gövde"), or every finding "what" can be bound to a related parent "what". Locations and other names in mined findings will not be stored as free-text in the database, but rather a link to a global instance of that name.

Whenever a new location/finding name, that has been never encountered before, is found in a report, the new name will be added to the global tables ("Ne" or "Yer") but no link will be created for its parent in the relation tables ("Ne_Rel" and "Yer_Rel"), because we cannot deduce it only using rules.

Although we cannot deduce ontology relations for new terms at runtime, we have the ability to fill the global name and relation tables manually before RadioRead is installed to a hospital. This way, a sufficient ontology might be produced either before the installation or after some time in production, thanks to our database design.

In the future, these ontology relations may be constructed using SNOMED information too, without disturbing patient records in database.

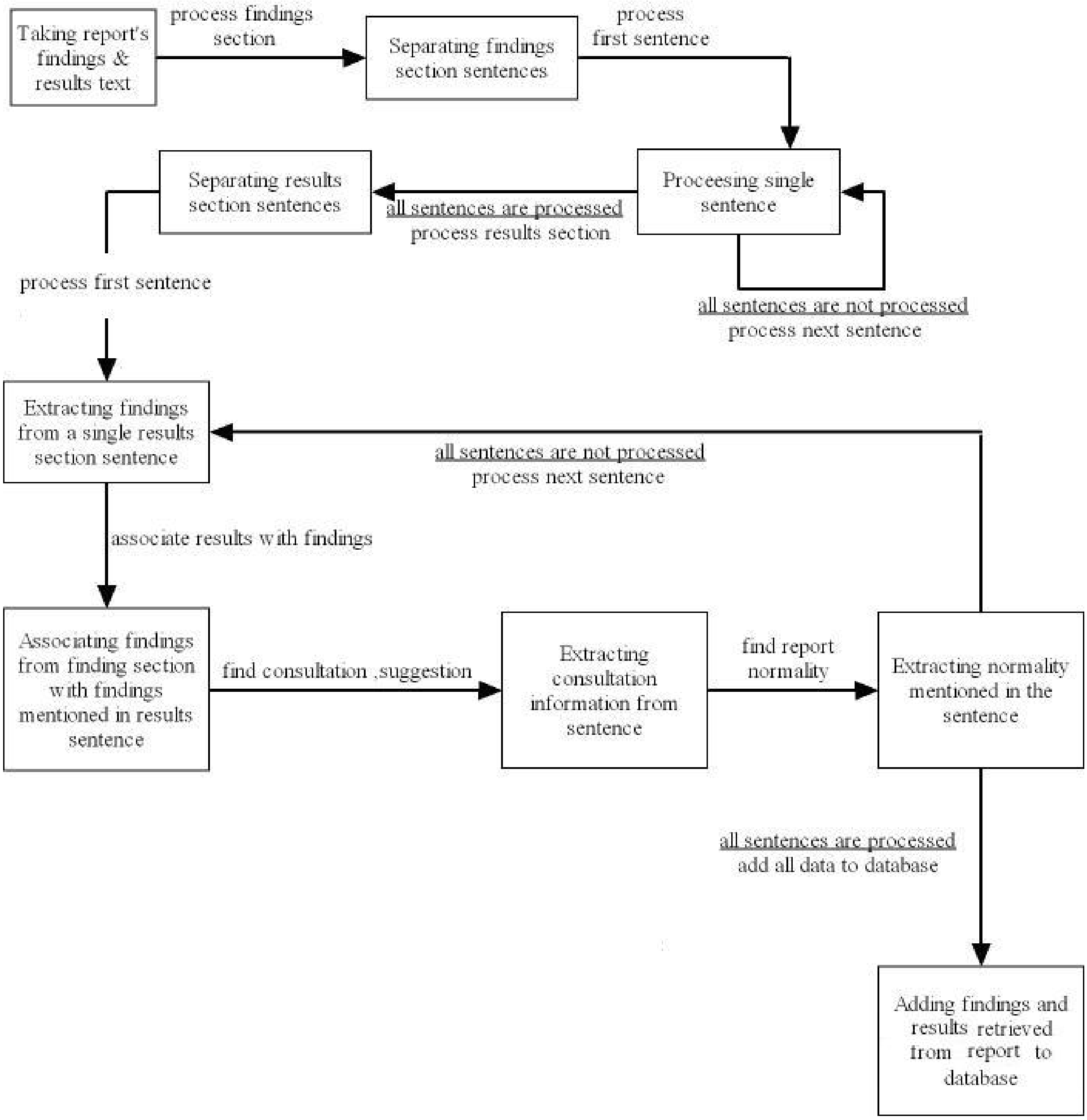
As we have mentioned above, we have 5 global tables ("Ne", "Yer", "Nasil", "Yer_Rel" "Ne_Rel"). We will relate "Ne" and "Yer".

5.2.4. Create Tables

CREATE TABLE statements are in Appendix C.

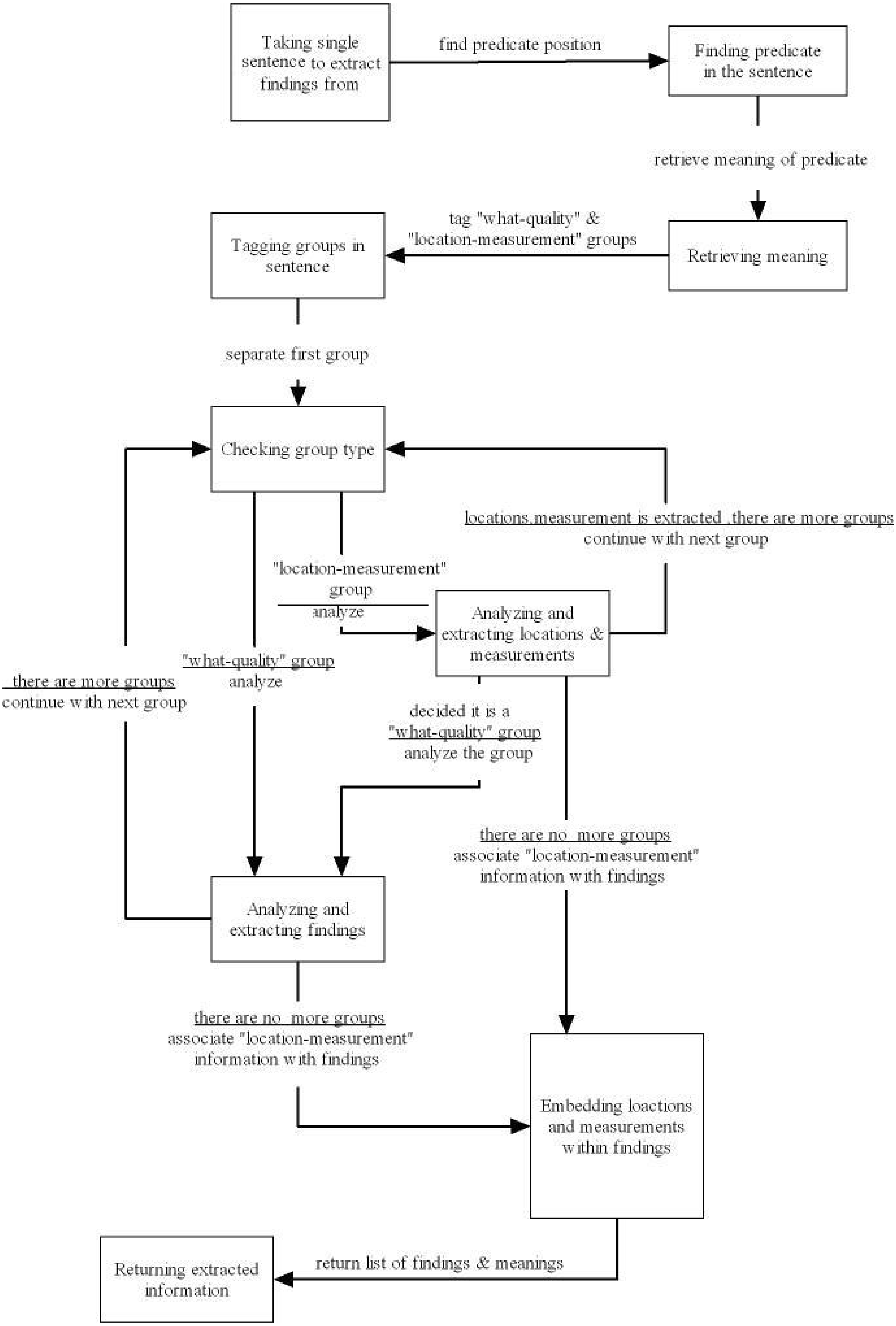
5.3. Behavioral Modelling

5.3.1. State Transition Diagram for Analyzing Reports

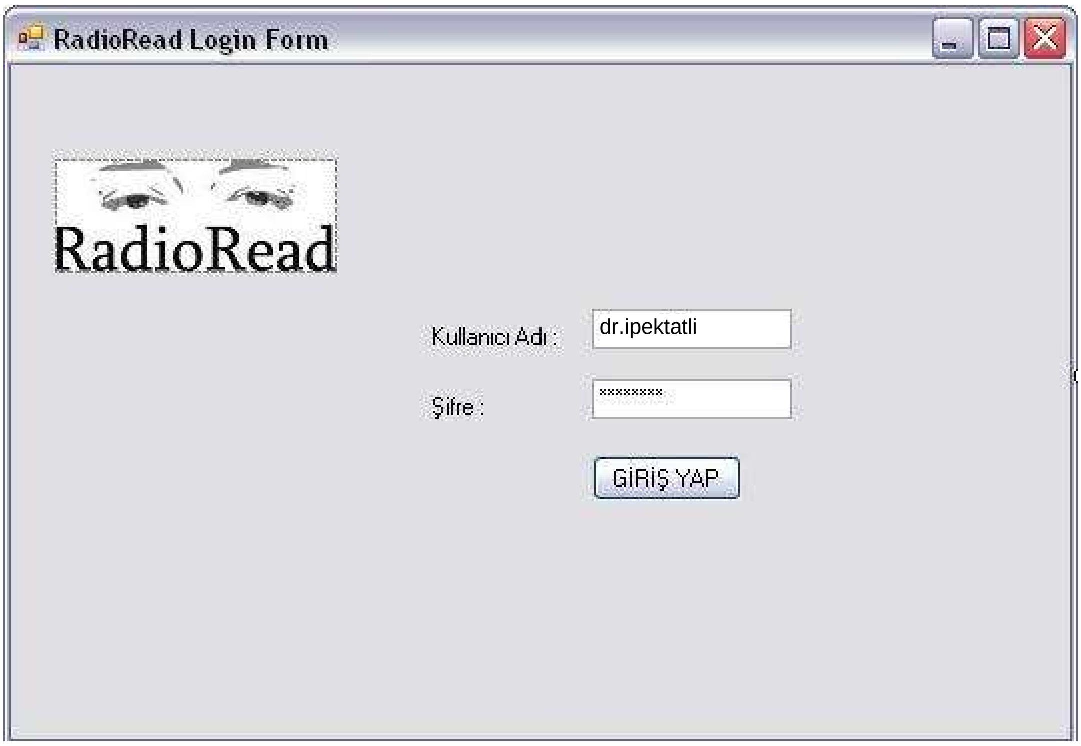


Taking report's findings & results text
process findings section
Separating findings section sentences
process first sentence
Proceesing single sentence
all sentences are processed
process results section
Separating results section sentences
all sentences are not processed
process next sentence
process first sentence
Extracting findings from a single results section sentence
all sentences are not processed
process next sentence
associate results with findings
Associating findings from finding section with findings mentioned in results sentence
find consultation ,suggestion
Extracting consultation information from sentence
find report normality
Extracting normality mentioned in the sentence
all sentences are processed
add all data to database
Adding findings and results retrieved from report to database

5.3.2. State Transition Diagram for Analyzing Single Sentences


Taking single sentence to extract findings from
find predicate position
Finding predicate in the sentence
retrieve meaning of predicate
Retrieving meaning
tag "what-quality" & "location-measurement" groups
Tagging groups in sentence
separate first group
Checking group type
locations,measurement is extracted ,there are more groups
continue with next group
"location-measurement" group
analyze
Analyzing and extracting locations & measurements
"what-quality" group
analyze
there are more groups
continue with next group
decided it is a
"what-quality" group
analyze the group
there are no more groups
associate "location-measurement" information with findings
Analyzing and extracting findings
there are no more groups
associate "location-measurement" information with findings
Embedding loactions and measurements within findings
return list of findings & meanings
Returning extracted information

6. GUI Design

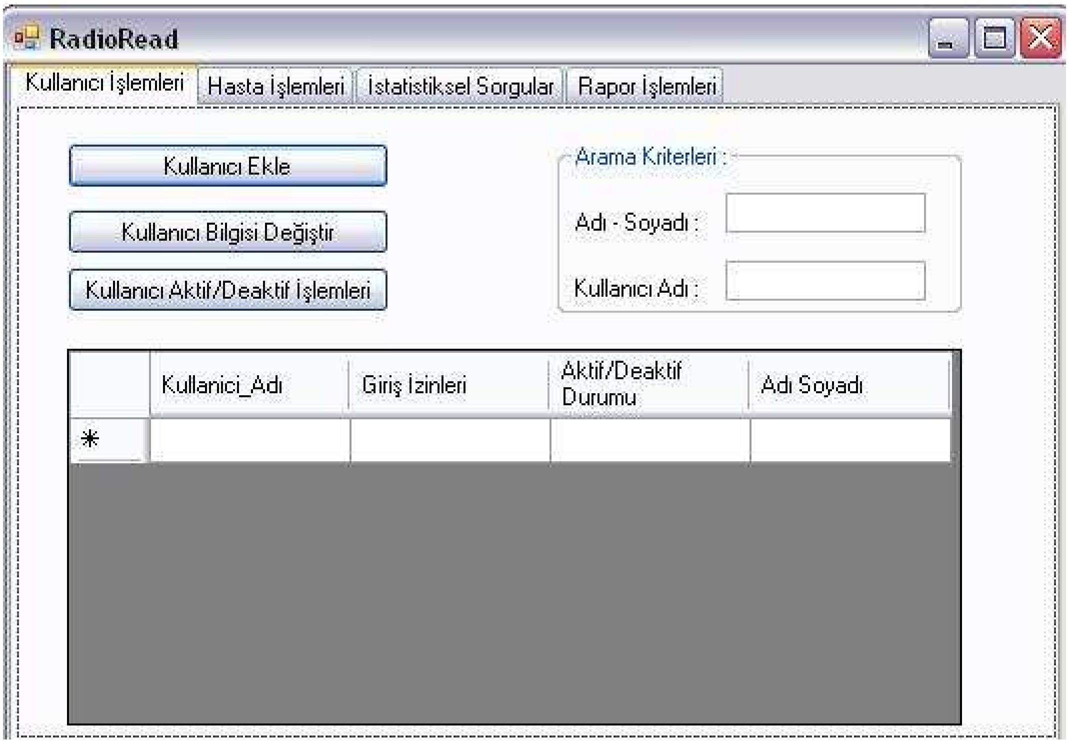


Kullanıcı Adı : dr.ipektatli

Şifre : ********

Figure 6.1- Login to RadioRead

Login screen is above. The user must enter his/her username and password to be able to login to the system (Figure 6.1).



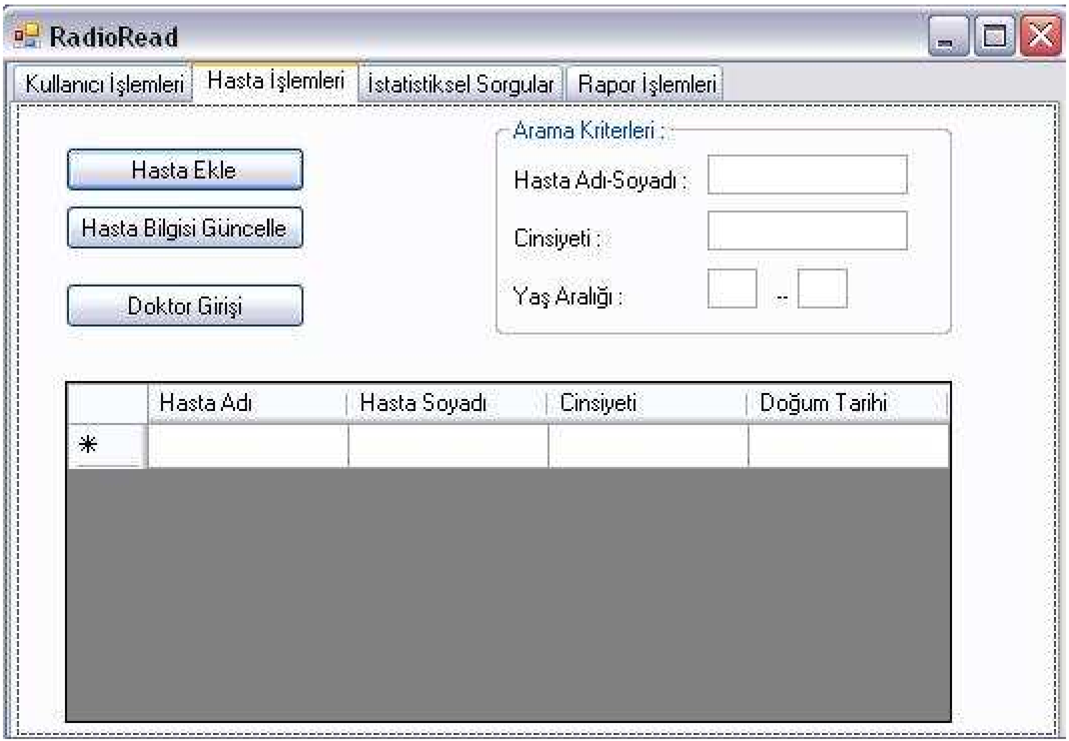
Adı - Soyadı :

Kullanıcı Adı :

	Kullanici_Adı	Giriş İzinleri	Aktif/Deaktif Durumu	Adı Soyadı
*				

Figure 6.2- "Kullanıcı İşlemleri" of RadioRead

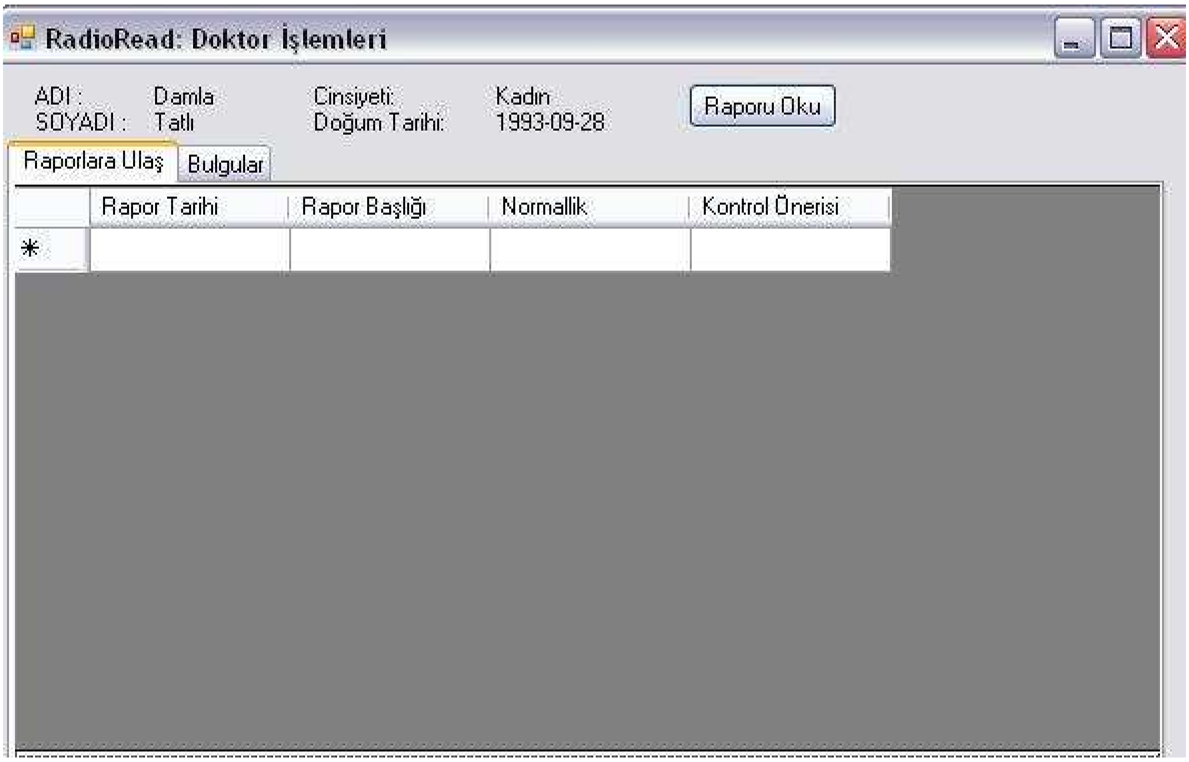
We have 4 tabs in the main window but every user is not able to use every task. They can access these tasks only if they have the necessary permissions. In RadioRead only administrators (modifyUsers privilege) can access “Kullanıcı İşlemleri” (Figure 6.2). Administrators can add and modify users, they can change user’s access rights and can activate/deactivate their accounts. They can search users according to name, surname and username. They can list users found in the grid view.



	Hasta Adı	Hasta Soyadı	Cinsiyeti	Doğum Tarihi
*				

Figure 6.3- “Hasta İşlemleri” of RadioRead

In RadioRead only the users who have AccessPatients or ManagePatients privileges can access “Hasta İşlemleri” (Figure 6.3). Users having ManagePatients privilege can list patients, can add and modify patient information. Users having AccessPatients privilege can list patients, can access the “Doktor Girişi” button which will show the patient details of the selected patient (explained later). Any user listing patients can search patients according to their name, surname, gender and age range.

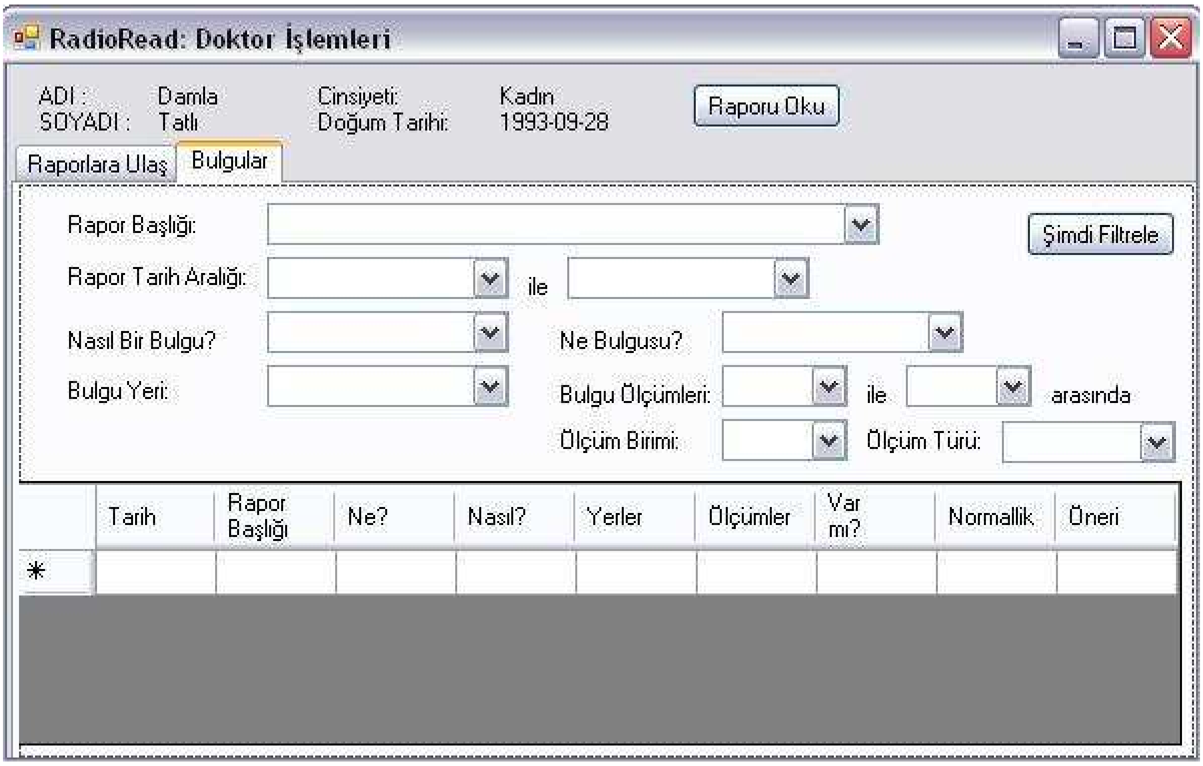


ADI : Damla
SOYADI : Tatlı
Cinsiyeti: Kadın
Doğum Tarihi: 1993-09-28

	Rapor Tarihi	Rapor Başlığı	Normallik	Kontrol Önerisi
*				

Figure 6.4- "Doktor İşlemleri" of RadioRead

In RadioRead only the users who have AccessPatients privileges can access "Doktor İşlemleri" (Figure 6.4 and 6.5). They can list reports of a patient or search within the extracted findings from the reports. By clicking on the "Raporu Oku" button, user can read the selected report or selected finding's report.

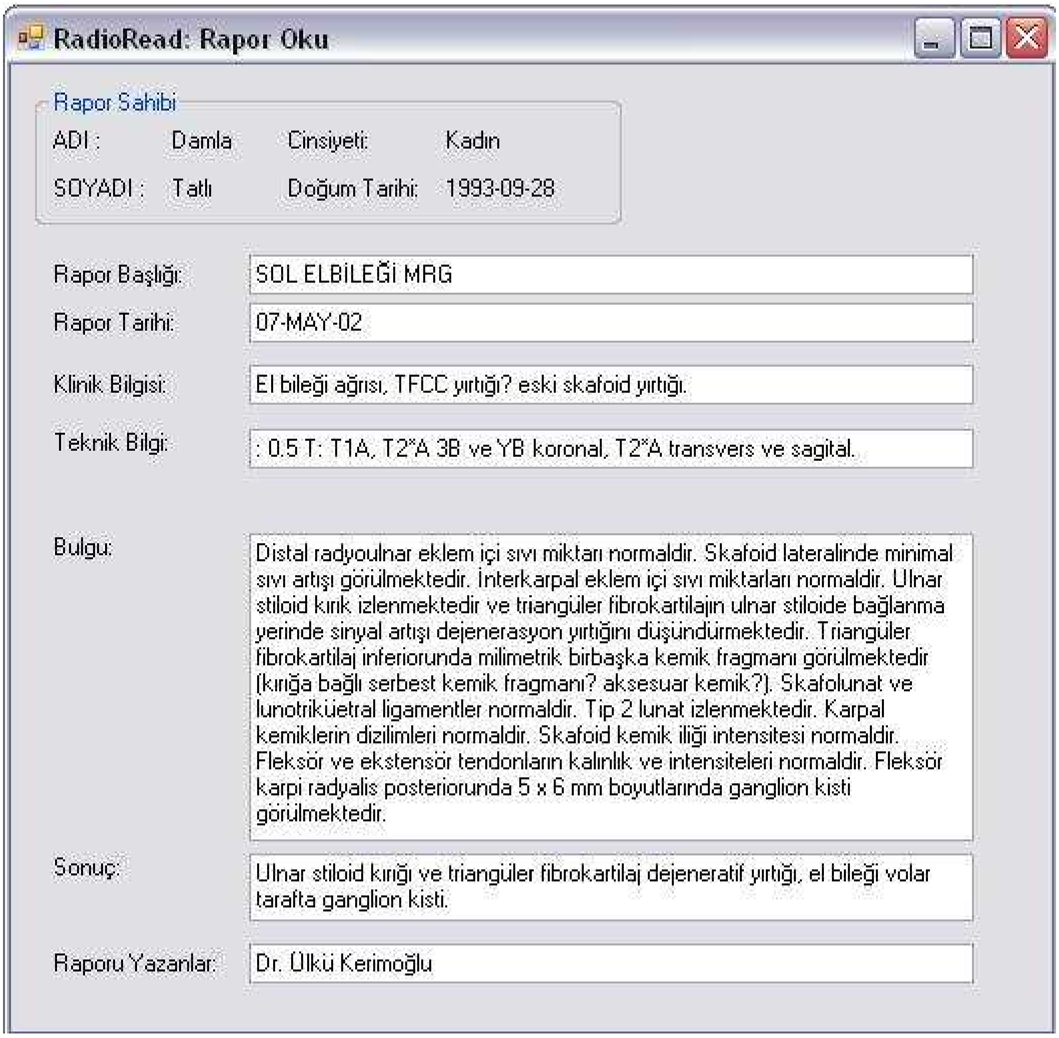


ADI : Damla
SOYADI : Tatlı
Cinsiyeti: Kadın
Doğum Tarihi: 1993-09-28

Rapor Başlığı:
Rapor Tarih Aralığı: ile
Nasıl Bir Bulgu? Ne Bulgusu?
Bulgu Yeri: Bulgu Ölçümleri: ile arasında
Ölçüm Birimi: Ölçüm Türü:

	Tarih	Rapor Başlığı	Ne?	Nasıl?	Yerler	Ölçümler	Var mı?	Normallik	Öneri
*									

Figure 6.5- "Doktor İşlemleri" of RadioRead



RadioRead: Rapor Oku

Rapor Sahibi

ADI : Damla

Cinsiyeti: Kadın

SOYADI : Tatlı

Doğum Tarihi: 1993-09-28

Rapor Başlığı: SOL ELBİLEĞİ MRG

Rapor Tarihi: 07-MAY-02

Klinik Bilgisi: El bileği ağrısı, TFCC yırtığı? eski skafoid yırtığı.

Teknik Bilgi: : 0.5 T: T1A, T2*A 3B ve YB koronal, T2*A transvers ve sagital.

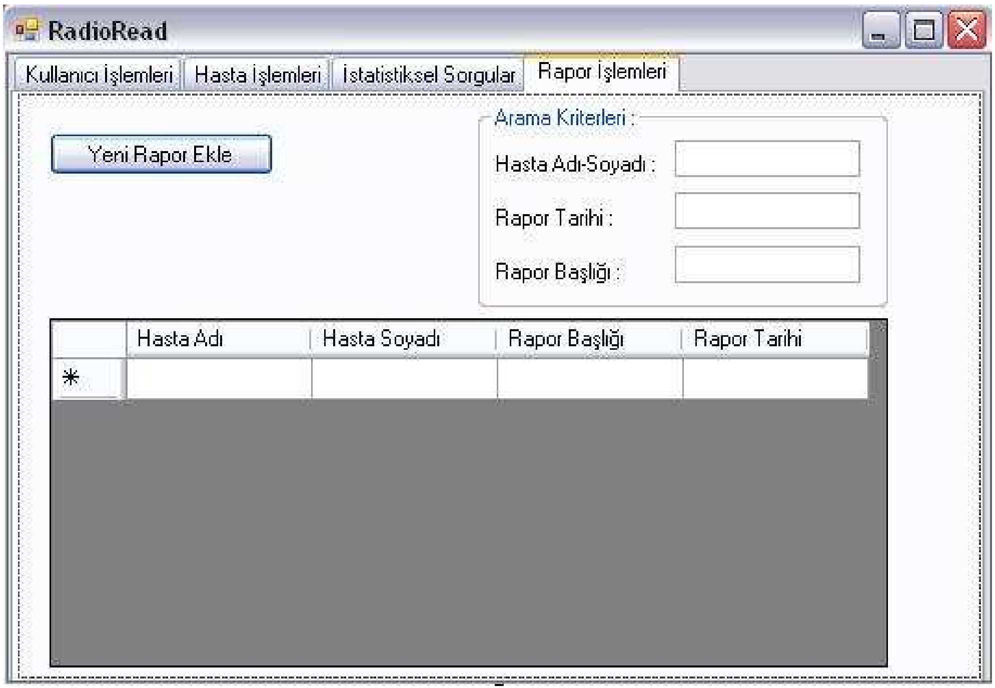
Bulgu: Distal radyoulnar eklem içi sıvı miktarı normaldir. Skafoid lateralinde minimal sıvı artışı görülmektedir. İnterkarpal eklem içi sıvı miktarları normaldir. Ulnar stiloid kırık izlenmektedir ve triangüler fibrokartilajın ulnar stiloide bağlanma yerinde sinyal artışı dejenerasyon yırtığını düşündürmektedir. Triangüler fibrokartilaj inferiorunda milimetrik birbaşka kemik fragmanı görülmektedir (kırığa bağlı serbest kemik fragmanı? aksesuar kemik?). Skafolunat ve lunotriküetral ligamentler normaldir. Tip 2 lunat izlenmektedir. Karpal kemiklerin dizilimleri normaldir. Skafoid kemik iliği intensitesi normaldir. Fleksör ve ekstensör tendonların kalınlık ve intensiteleri normaldir. Fleksör karpi radyalis posteriorunda 5 x 6 mm boyutlarında ganglion kisti görülmektedir.

Sonuç: Ulnar stiloid kırığı ve triangüler fibrokartilaj dejeneratif yırtığı, el bileği volar tarafta ganglion kisti.

Raporu Yazanlar: Dr. Ülkü Kerimoğlu

Figure 6.6- "Rapor Oku" in RadioRead

This is the view report window. Any user that has AccessPatients privilege can access this window (Figure 6.6).



	Hasta Adı	Hasta Soyadı	Rapor Başlığı	Rapor Tarihi
*				

Figure 6.7 "Rapor İşlemleri" of RadioRead

In RadioRead only the users who have "AddReports" permissions can access "Rapor İşlemleri" (Figure 6.7). They can add a new report. They can search reports according to name and surname of patients, report date and report title. They can list reports found in the grid view. They cannot read reports.

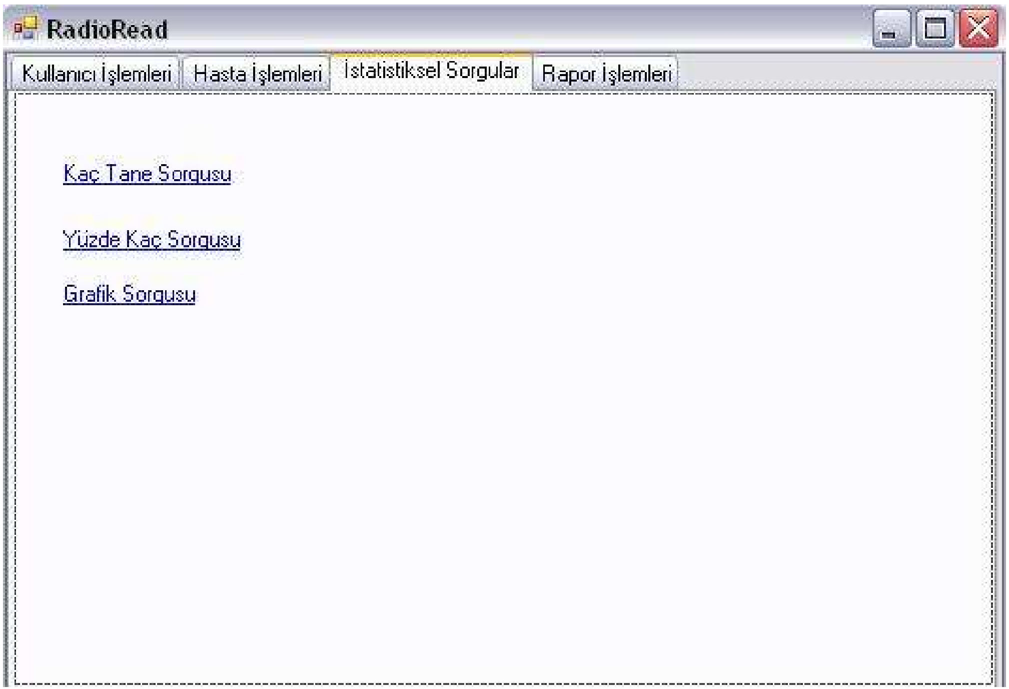


Figure 6.8 "İştatistiksel Sorgular" of RadioRead

In RadioRead only the users who have “QueryReports” privilege can access “İstatistiksel Sorgular” (Figure 6.8). There are three types of queries: “how many”, “what percentage” and “graphical”. Users can select a query type by using the links in the window.

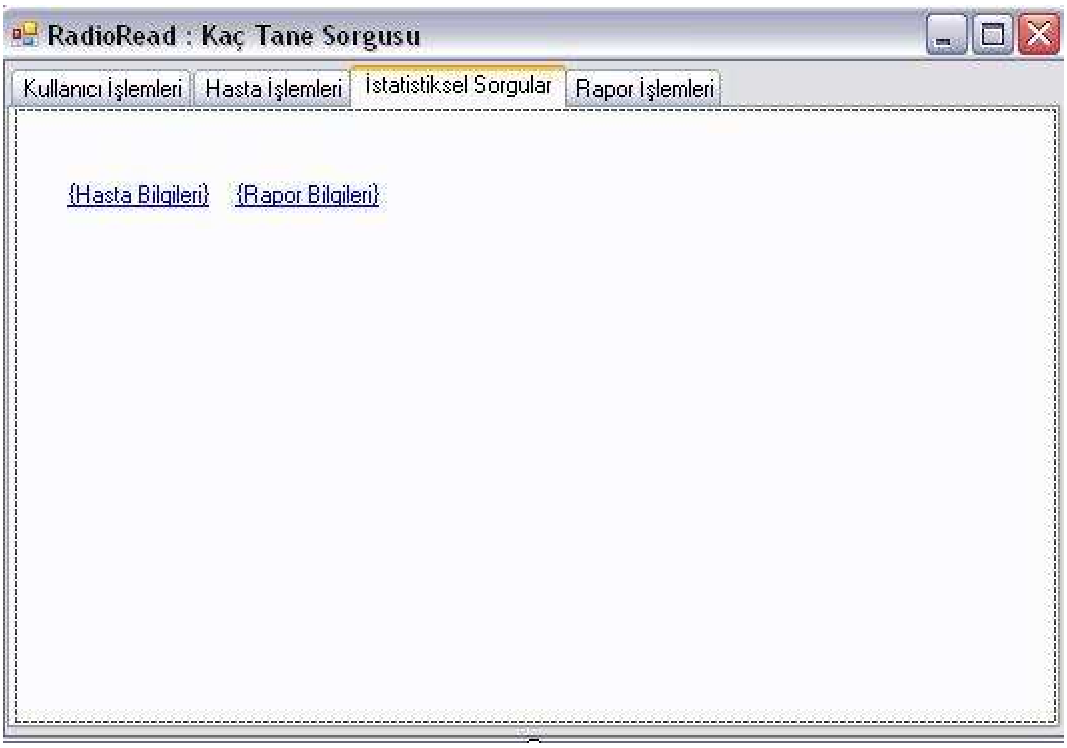


Figure 6.9 “Kaç Tane Sorgusu” part of RadioRead

Above is the “how many” query (Figure 6.9). The user will specify details of the components of the query by clicking on the links. Clicking on “Hasta Bilgileri” brings the following window:

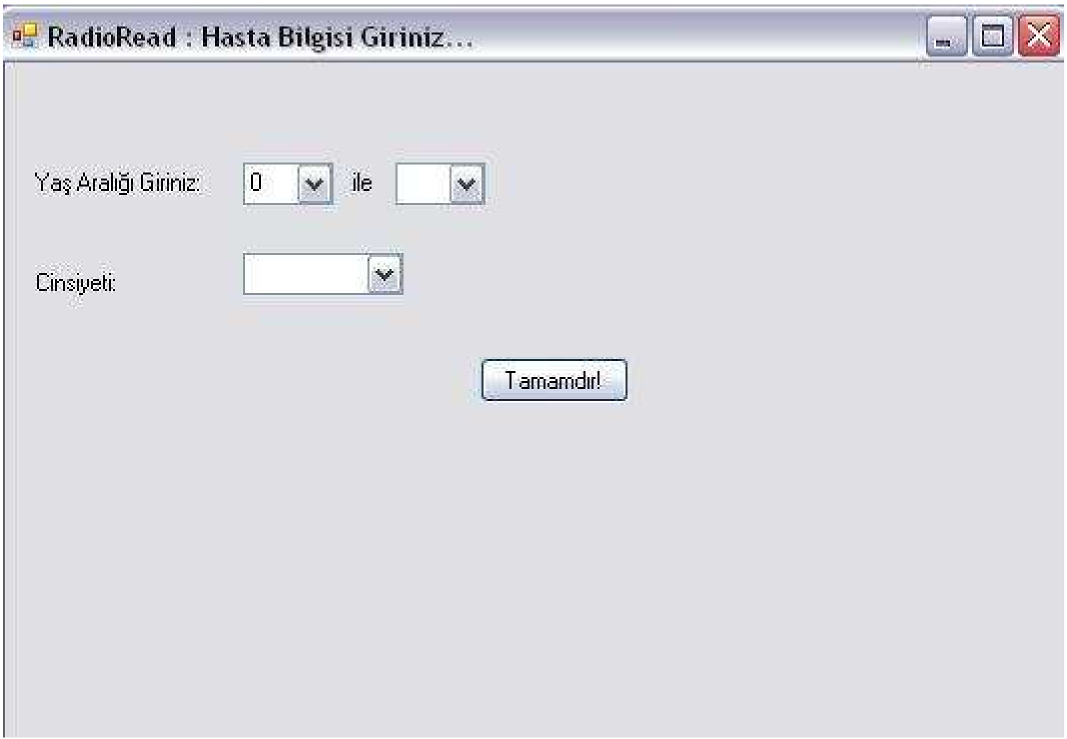
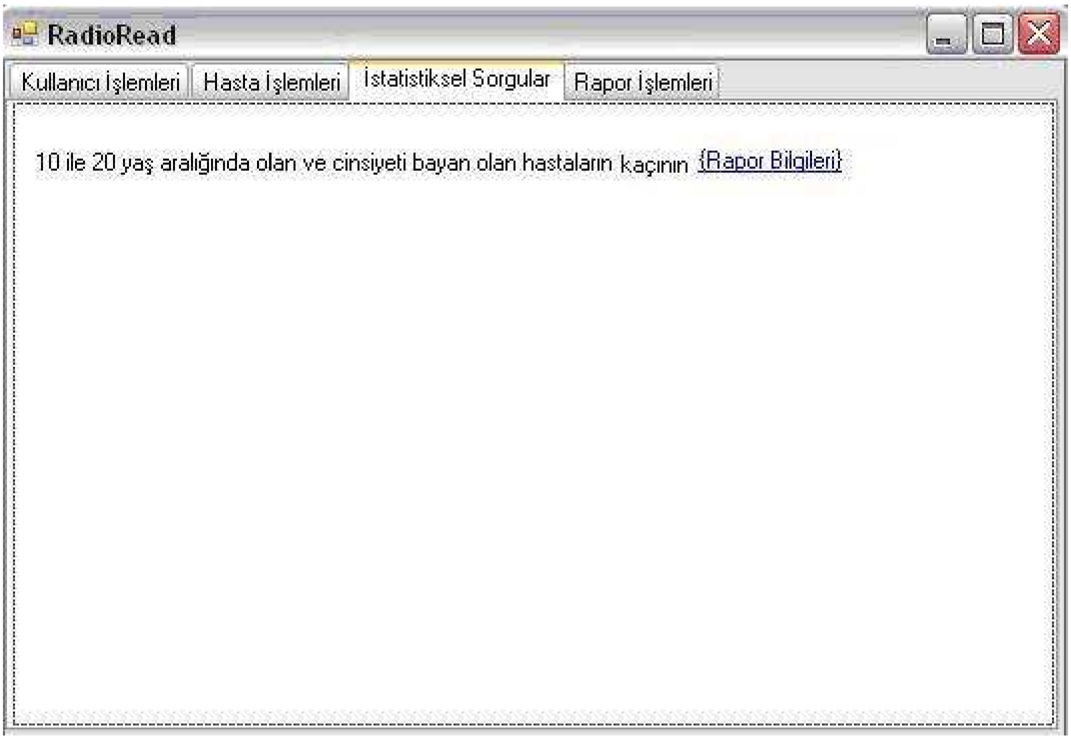


Figure 6.10 "Hasta Bilgisi" part "İstatistiksel Sorgular"

After the user specifies the constraints in the above window (Figure 6.10), the query text will be automatically updated as below:



10 ile 20 yaş aralığında olan ve cinsiyeti bayan olan hastaların kaçının {Rapor Bilgileri}

Figure 6.11 View of the program after patient constraints

Then the user can click the "Rapor Bilgileri" link to specify report constraints (Figure 6.11 and 6.12).

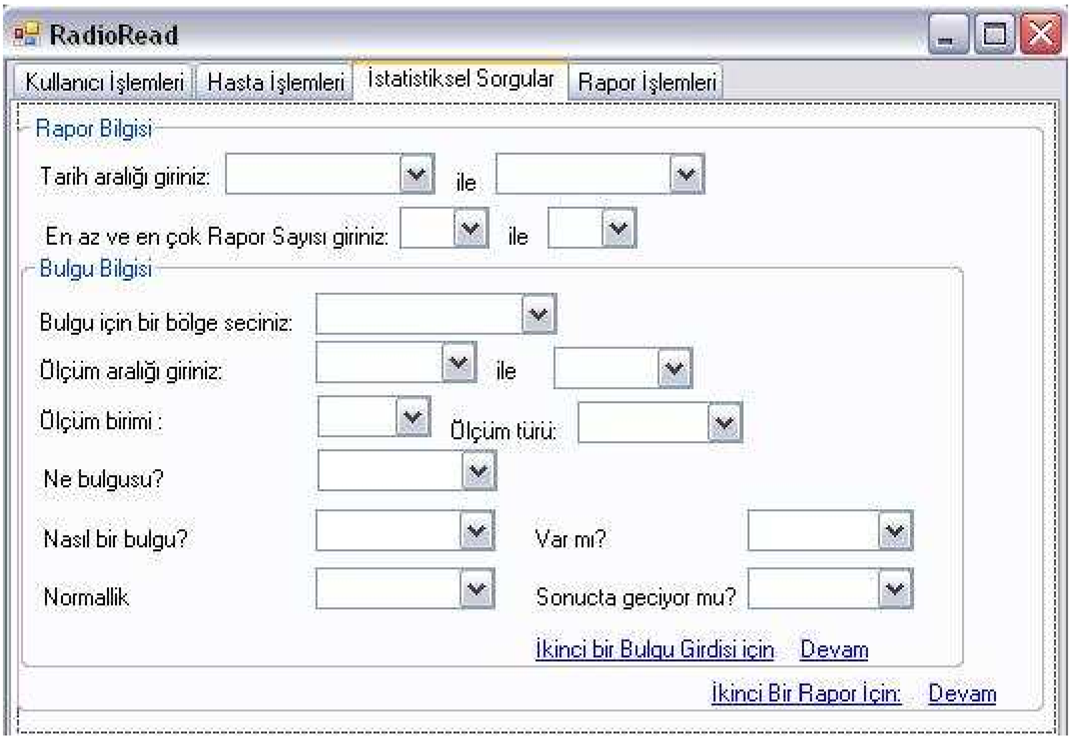
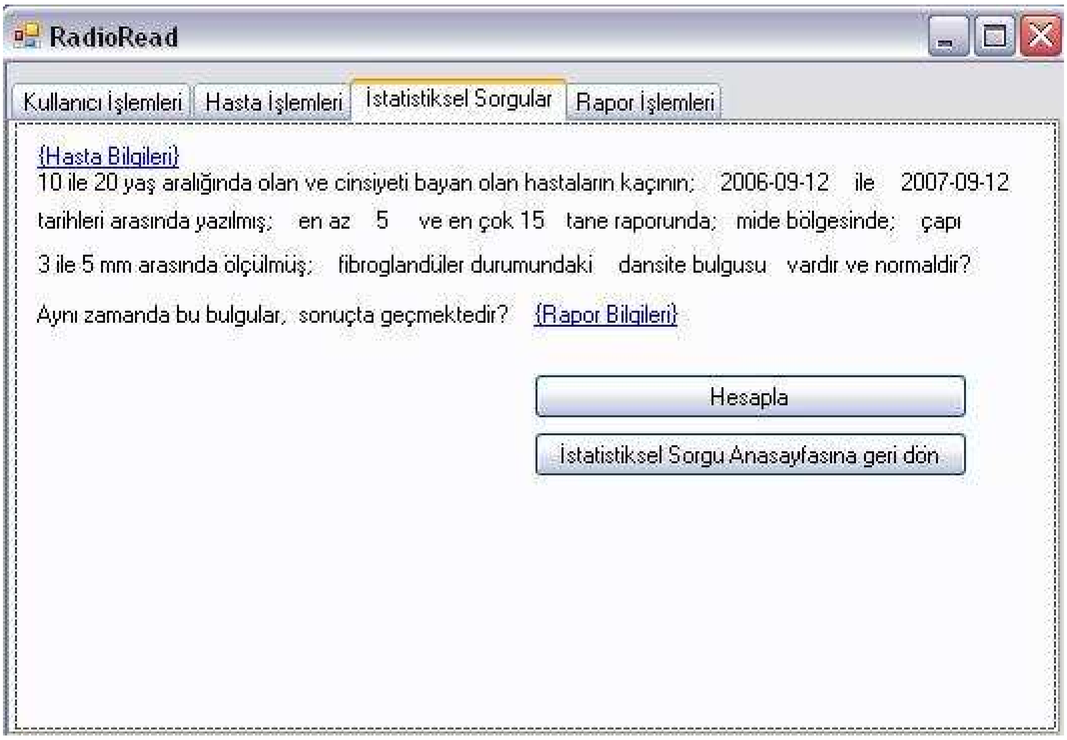


Figure 6.12 "Rapor Bilgileri" part of "İstatistiksel Sorgular"

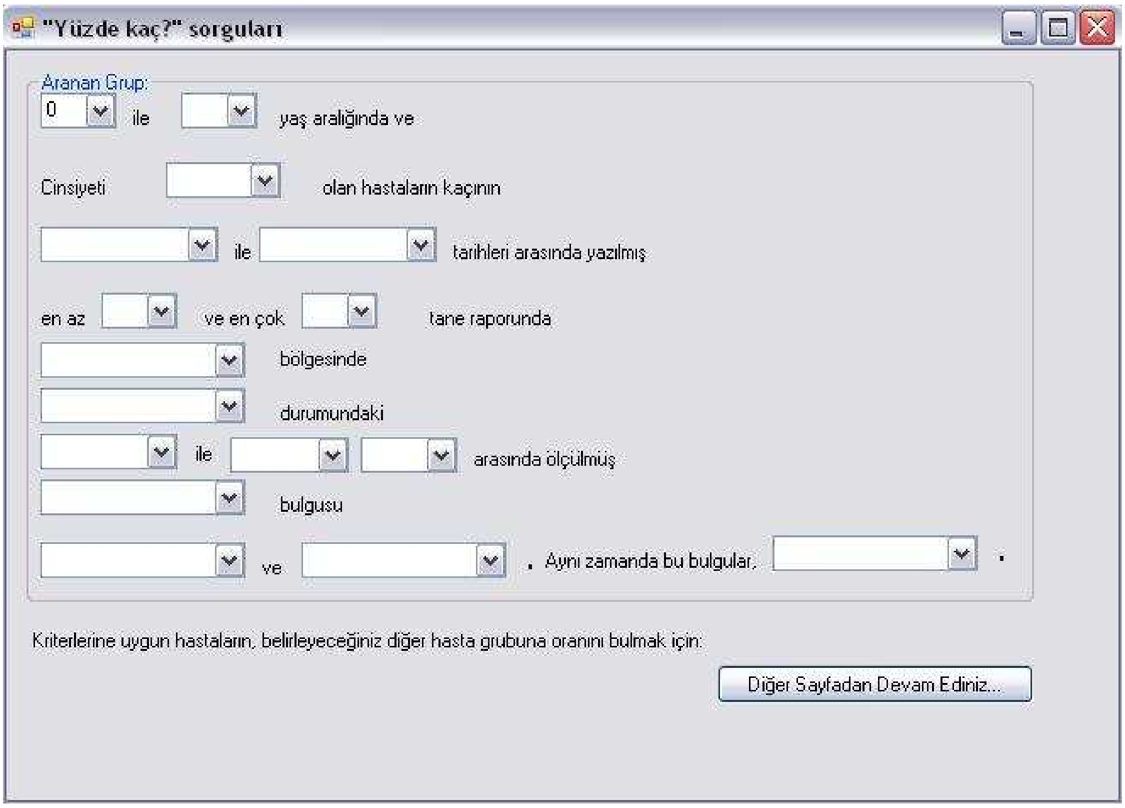


10 ile 20 yaş aralığında olan ve cinsiyeti bayan olan hastaların kaçının; 2006-09-12 ile 2007-09-12 tarihleri arasında yazılmış; en az 5 ve en çok 15 tane raporunda; mide bölgesinde; çapı 3 ile 5 mm arasında ölçülmüş; fibroglandüler durumundaki dansite bulgusu vardır ve normaldir?

Aynı zamanda bu bulgular, sonuçta geçmektedir?

Figure 6.13 View of the program after report and findings constraints

A finished "how many" query will look like as in the above window (Figure 6.13). The user can click on the "Hesapla" button to execute the query.



"Yüzde kaç?" sorguları

Aranan Grup:

0 ile yaş aralığında ve

Cinsiyeti olan hastaların kaçının

ile tarihleri arasında yazılmış

en az ve en çok tane raporunda

bölgesinde

durumundaki

ile arasında ölçülmüş

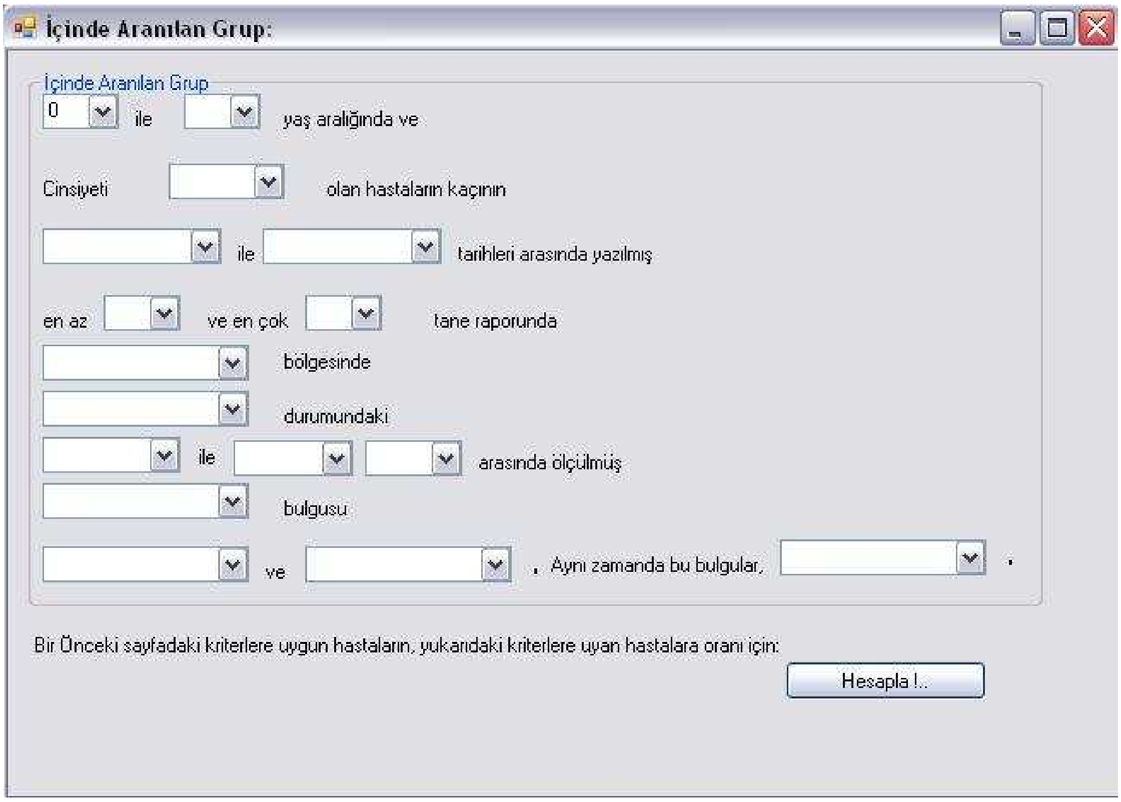
bulgusu

ve , Aynı zamanda bu bulgular, .

Kriterlerine uygun hastaların, belirleyeceğiniz diğer hasta grubuna oranını bulmak için:

Diğer Sayfadan Devam Ediniz...

Figure 6.14. "What Percentage Queries" of Statistical Query



İçinde Aranılan Grup:

İçinde Aranılan Grup

0 ile yaş aralığında ve

Cinsiyeti olan hastaların kaçının

ile tarihleri arasında yazılmış

en az ve en çok tane raporunda

bölgesinde

durumundaki

ile arasında ölçülmüş

bulgusu

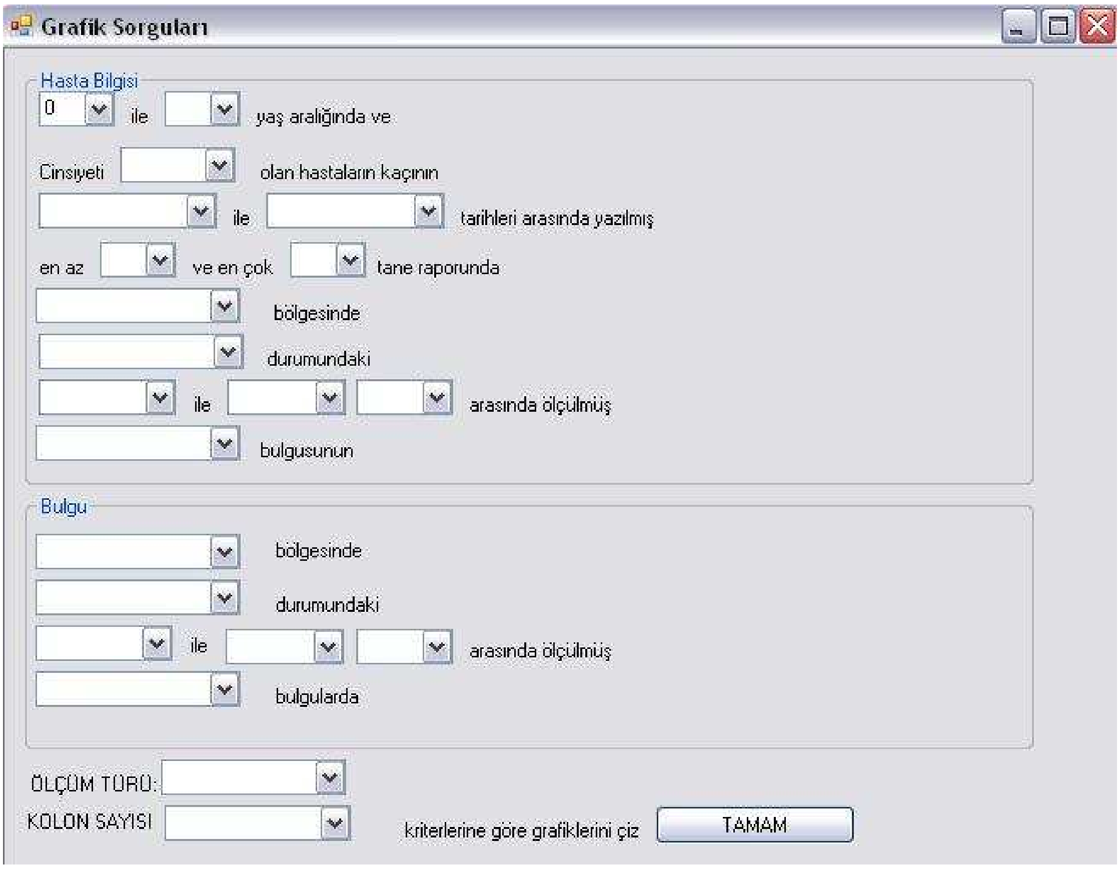
ve , Aynı zamanda bu bulgular, .

Bir Önceki sayfadaki kriterlere uygun hastaların, yukarıdaki kriterlere uyan hastalara oranı için:

Hesapla !..

Figure 6.15. "What Percentage Queries" of Statistical Query-Part2

After clicking the "Hesapla" button, the user will first see the query sentence in natural language, then he can change the query or decide to execute the query (Figure 6.14 and 6.15).



Grafik Sorguları

Hasta Bilgisi

0 ile yaş aralığında ve

Cinsiyeti olan hastaların kaçının

ile tarihleri arasında yazılmış

en az ve en çok tane raporunda

bölgesinde

durumundaki

ile arasında ölçülmüş

bulgusunun

Bulgu

bölgesinde

durumundaki

ile arasında ölçülmüş

bulgularda

ÖLÇÜM TÜRÜ:

KOLON SAYISI kriterlerine göre grafiklerini çiz

Figure 6.16. "Measurement Graphic Queries" of Statistical Query

After clicking the "Tamam" button, the user will first see the query sentence in natural language, and then he can change the query or decide to execute the query (Figure 6.16). When the query is executed, the user will see a bar graph with "KOLON_SAYISI" bars; each bar showing the number of finding measurements of type "ÖLÇÜM_TÜRÜ" laying in some range.

7. Testing Methodology

7.1. Integrity Checks

We plan to use Unit Tests to maintain the integrity of components and classes over time. Due to time restrictions in the first semester, we did not write unit tests during the first semester, but maintained the design and components separate from each other. For example, our current design allows instantiating and testing a SentencePartsGrouper apart from the rest of RadioRead, which makes our components ideal for unit testing.

The design restrictions chosen this semester will allow us to easily integrate Unit Test frameworks such as JUnit within RadioRead codebase in the second semester.

7.2. Success Analysis

We will measure success of our project in two dimensions: rate of negatives (of findings), and rate of positives (of findings). A negative is defined as a finding that does not exist in a report, but found to be existent by the mining engine, and a positive is defined as a finding that exists in a report and found. Ideally, rate of negatives should be 0% and rate of positives should be 100%.

We plan to create a list of expected results from randomly selected sample reports in a systematic way. After every major improvement, these reports will be given to RadioRead-Report Information Engine for mining, and the automated results will be compared with the results we have extracted manually. From these comparisons, we will be able to calculate rate of negatives and rate of positives.

Doing these analyses periodically, we will be able to track our success over time.

8. Development Schedule

8.1. What Has Been Done So Far

8.1.1. Overview

During the first semester, we have designed crucial parts of our project and implemented a proof-of-concept prototype to show our ideas.

We have decided on using rule based approaches in RadioRead. We believe that rule based approach is most suitable in these radiology reports as the information contained within is not sparse, but very dense.

We have designed the code and component structure of RadioRead in this semester. We have implemented all the classes under Report Information Engine's Mining part in a basic way for the prototype, as that is the main focus of our project. The implemented part covers all the children of FindingsSectionMiner in the component graph shown in section 4.2.3.

We have clearly defined our algorithms and rules for text-mining, and distributed these rules to different components in our design, as mentioned in section 4.3.2.

8.1.2. Statistical Queries (also mentioned as Complex Queries)

As we stated before, we aim to develop a useful information acquirement method from huge amount of electronic patient reports to enable secure, ethical and user friendly access to patient information. We will provide an environment for users to access these information as easy as using a natural language; an environment in which the user does not have to know anything about technical aspects of how the information is represented in the database systems involved.

RadioRead has 3 types of statistical queries. One of them is "How many?" query which provides to access how many patients there are with the given specifications. Other is "What percentage?" query. This query provides access to percentage of patients over a super class of patients. The user will give specifications about a group of patients (a) and another group of patients (b) that encloses the first group. RadioRead will return the percentage of 'a intersection b' over 'b'. The third type of the queries is "Measurement-Graph" query which provides access to some graphics about the specifications given by the user. User will give a group of patients (a), a single finding (b), and measurement type (c) and a number (d). 'b' must exist inside 'a's specified reports if there is any report constraint given. Otherwise 'b' exists inside all reports of the patient 'a'. RadioRead will return a graph plotting the measurement of ‘c’ in ‘b’. ‘d’ specifies the number of groups (columns) in the graph. Each column in the graph has the range

```
        (calculated_max_measurement - calculated_min_measurement) / d
```

as measurement value.

The language grammar that we created for the statistical queries can be found in Appendix A. A sample on how a parse tree of this grammar can be used to construct an SQL query or Turkish sentence is shown in Section 4.3.5.

8.1.3. Basic Queries

These queries do not use acquired data mined from the reports. Instead, they are used for data such as information of patients, doctors, users or reports.

8.1.4. Accessing an External Dictionary

We plan to use *Zargan* as an external dictionary because it has a medical dictionary inside. We have implemented some classes in Java and we can send words to Zargan and we can have information about whether it is a medical term or not.

When we ask a word to Zargan, Zargan may propose us some words similar to the given word if the word is not in it. This provides us to guess and analyze the most similar word and send it to Zemberek’s dictionary. Zemberek can separate it into its root and suffixes. If the word is in Zargan, Zargan gives the Turkish, English meanings and source dictionary type of the word. It also gives us phrases about its usage and synonyms of the word that we asked. This is useful for us because we can guess if it is an illness or it is a locus of a patient etc. Also we will utilize the phrase usage information to separate “what” and “quality” from an input in a better way. Source dictionary type in Zargan is also convenient to differentiate between medical terms and non-medical words.

Zargan has XHTML pages that can be easily parsed with Java’s XML parsers. This makes Zargan an excellent choice.

8.1.5. Semantic Analysis

8.1.5.1. Importance of Noun Phrases in Sample Radiology Reports

Semantic analysis is the most important step in natural language processing. In this step, sentences are translated to semantic formulas. In order to create these formulas, lexical and syntactic analysis are used.

We have analyzed the sample clinical reports given to us and seen that most of the sentences in the findings part can be considered as “simple sentences”, composed of a (probably) complex noun phrase and a verb phrase. The noun phrase is a composition of different findings, whereas the verb phrase (usually consisting of one or two words) identifies the overall semantic information about these findings, such as “exists” “do not exist” “is identified” “is normal” “is abnormal”.

8.1.5.2. Noun Phrase Parser Grammar

We have decided to start with parsing noun phrases. Noun phrases are rather complex phrases, since there can be multiple noun phrases connected to each other with conjunction operators (e.g: ',', 've', 'ile'). Even more, these operators not only specify connections, but also associativity. Common parts of phrases can be grouped together such as in arithmetic, forming a complex noun phrase, which can then take part in a bigger noun phrase. In order to handle noun phrases we have decided to write our own noun phrase parser.

We have written a grammar for handling noun phrases suitable for bottom-up parsing, and conducted tests using JS/CC. JS/CC is a LALR(1) parser and lexical analyzer generator for JavaScript, written in JavaScript. Although JavaScript interpreters are readily available for Java, JS/CC has its limitations. JS/CC does not allow backtracking, and that restricts our grammar to a certain subset. We plan to use another compiler compiler for our grammar in RadioRead.

There are mainly two types of phrases in Turkish that we are interested in: adjective phrases and noun phrases. These phrases both consist of two parts, “Tamlayan” and “Tamlanan”. There are 2 kinds of suffixes that are related; “-ı/-i” (specifying a Tamlanan) and “-ın/-in (specifying a Tamlayan).

Adjective phrases consists of one adjective (Tamlayan) and one noun (Tamlanan), without suffixes. Although adjective phrases act as nouns (as a group) in other phrases, they cannot act as a noun in another adjective phrase; so an adjective phrase only consist of two consecutive words without suffixes.

Noun phrases are in three different forms, “belirtili”, “belirtisiz” and complex. All three of them has two parts: Tamlayan and Tamlanan. “Belirtili” and “belirtisiz” noun phrases consist of two nouns, the second one (“tamlanan”) always has the suffix “-ı/-i”. In “belirtili” noun phrase, the first word (“tamlayan”) always has the suffix “-ın/-in”, and in “belirtisiz” noun phrase, the “tamlayan” does not contain any suffix.

Complex noun phrases are in fact “belirtili” noun phrases, whose “tamlayan” part is not a word but another noun phrase. The noun phrase still has the suffix “-ın/-in”.

According to our grammar, there are two noun phrase kinds: adjective phrases (SIFAT_TAMLAMASI) and noun phrases (ISIM_TAMLAMASI). Adjective phrases (SIFAT_TAMLAMASI) are composed of one noun (ISIM) or two consecutive nouns. If there is one noun in an adjective phrase, then the phrase degrades to a word. Noun phrases (ISIM_TAMLAMASI) are composed of two parts, namely TAMLAYAN and TAMLANAN_GRUBU.

TAMLAYAN can be composed of BELIRTILI_TAMLAYAN_GRUBU or BELIRTISIZ_TAMLAYAN. BELIRTILI_TAMLAYAN_GRUBU is composed of BELIRTILI_TAMLAYAN_GRUBU connected to each other with commas (','), ILE ('with'), VE ('and') or BELIRTILI_TAMLAYAN_GRUBU2. BELIRTILI_TAMLAYAN_GRUBU2 is composed of TAMLAMA -IN or multiple SIFAT_TAMLAMASI connected with commas and ends with VE/ILE SIFAT_TAMLAMASI –IN . BELIRTISIZ_TAMLAYAN is mainly a name only.

TAMLANAN_GRUBU is composed of multiple TAMLANAN_GRUBU connected to each other with commas (','), ILE, VE or TAMLANAN_GRUBU2. TAMLANAN_GRUBU2 is composed of SIFAT_TAMLAMASI -I or multiple SIFAT_TAMLAMASI connected with commas and ended with VE/ILE SIFAT_TAMLAMASI -I.

Example:

Ali’nin evinin pembe duvarı, mavi panjuru ve eflatun çatısı

ISIM -IN ISIM -I -IN ISIM ISIM -I , ISIM ISIM -I VE ISIM ISIM –I

You can see our Noun Phrase Parser Grammar in Appendix B.

Ex: "Sol memede kitle, sağ memede dansitenin artışı normal değildir."

Result:

```
--- Parse tree ---
p'
      p
            BASLANGIC >^<
            tAMLAMA
                  sIFAT_TAMLAMASI
                        bELIRTISIZ_TAMLAYAN
                              ISIM >Sol<
                        bELIRTISIZ_TAMLAYAN
```

```
                              ISIM >meme<
            BITIS >$<
            P

“TAKISIZ” NOUN PHRASE
Tamlayanlar:
      ISIM Sol
Tamlananlar:
      ISIM meme


Result1: Sol meme


--- Parse tree ---
p'
      p
            BASLANGIC >^<
            tAMLAMA
                  sIFAT_TAMLAMASI
                        bELIRTISIZ_TAMLAYAN
                              ISIM >Sağ<
                        bELIRTISIZ_TAMLAYAN
                              ISIM >meme<
            BITIS >$<
            p
“TAKISIZ” NOUN PHRASE
Tamlayanlar:
      ISIM Sağ
Tamlananlar:
      ISIM meme


Result1: Sağ meme


--- Parse tree ---
p'
      p
            BASLANGIC >^<
            tAMLAMA
                  iSIM_TAMLAMASI
                        bELIRTILI_TAMLAYAN_GRUBU
                              bELIRTILI_TAMLAYAN_GRUBU2
                                    sIFAT_TAMLAMASI
                                          bELIRTISIZ_TAMLAYAN
                                                ISIM >kitle<
                                    vtyildiz
                                    VE >VE<
                                    sIFAT_TAMLAMASI
                                          bELIRTISIZ_TAMLAYAN
                                                ISIM >dansite<
                                    IN >-IN<
                        tAMLANAN_GRUBU
                              tAMLANAN_GRUBU2
                                    tEK_TAMLANAN
                                          bELIRTISIZ_TAMLAYAN
                                                ISIM >artış<
                                          I >-I<
            BITIS >$<
            p

“BELIRTILI” NOUN PHRASE:
```

```
Tamlayanlar:
      ISIM kitle
      ISIM dansite
Tamlananlar:
      ISIM artış
Result1: kitle artışı
Result2: dansite artışı
```

8.2. Future Work

In second semester, we will be mainly concerned with the improvement of our first prototype on mining, and also work on other parts of the project. We have already implemented some parts of report information engine. In the remaining time, we will optimize these parts and implement remaining parts of report information engine (statistical query engine), external query engine, GUI, GUI manager classes and database components.

As we have mentioned, we have implemented the mining module which extracts what the finding is and the location, measurement, quality of the finding. According to our plan, we will optimize this module to reach high percentage of success. We have designed external query, GUI and database; we will start to implement these components.

We have planned our schedule and divided it into three main phases, which of each has a milestone. In the second semester, we will start with an overview of our prototype.

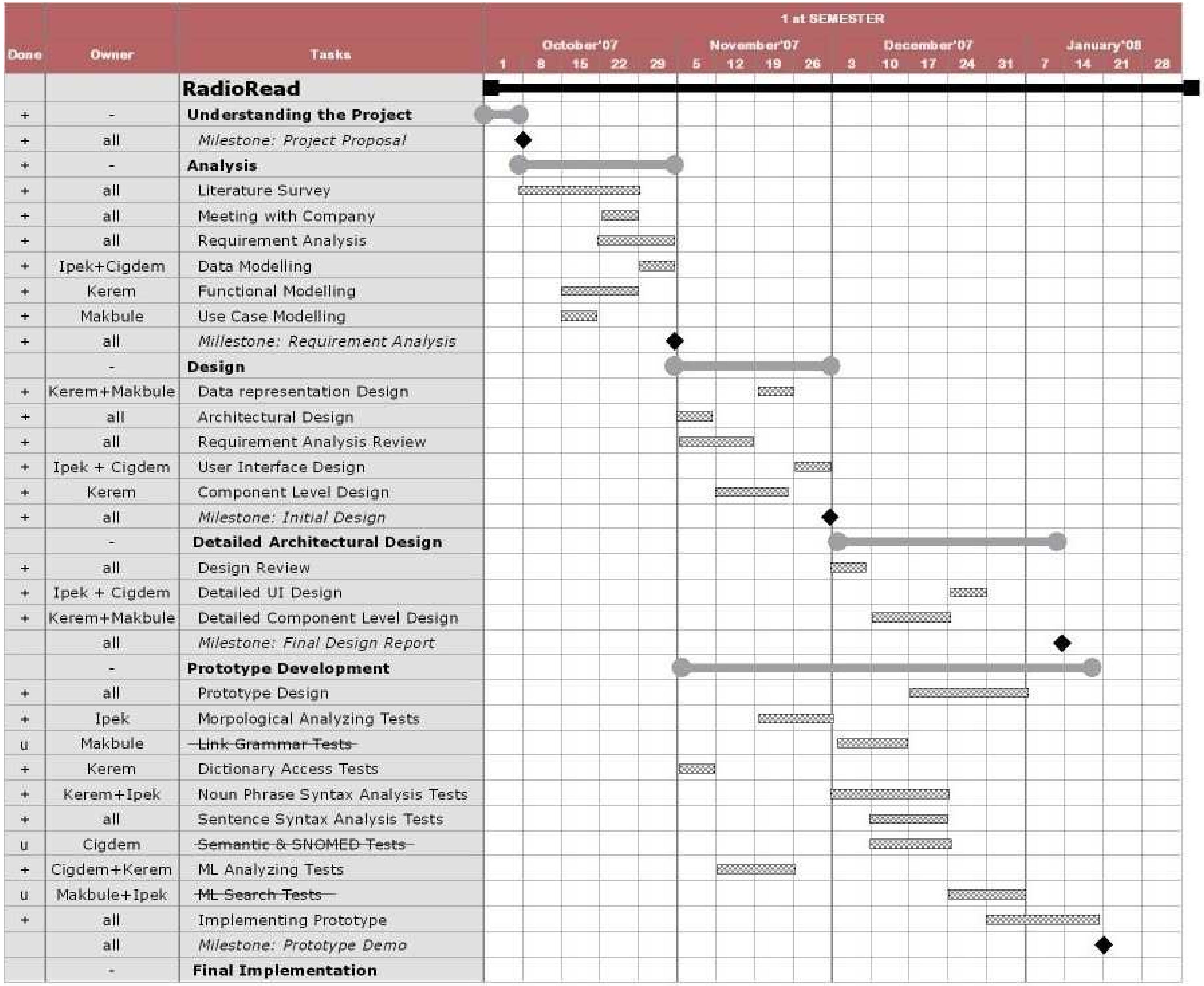
For the first milestone, we plan to re-implement Noun Phrase Parser (due to some bugs in the prototype version), to implement External Query Manager and to optimize Report Information Engine’s Mining Module.

Until the second milestone, we will start to implement Database, GUI, Report Information Engine’s Query Module and Data Engine Module. Also, we will continue optimizing Report Information Engine’s Mining Module.

Until the last milestone, we will continue to implement Database, GUI and Report Information Engine’s Query Module. In the rest of the semester, we will test our project to get the final version.

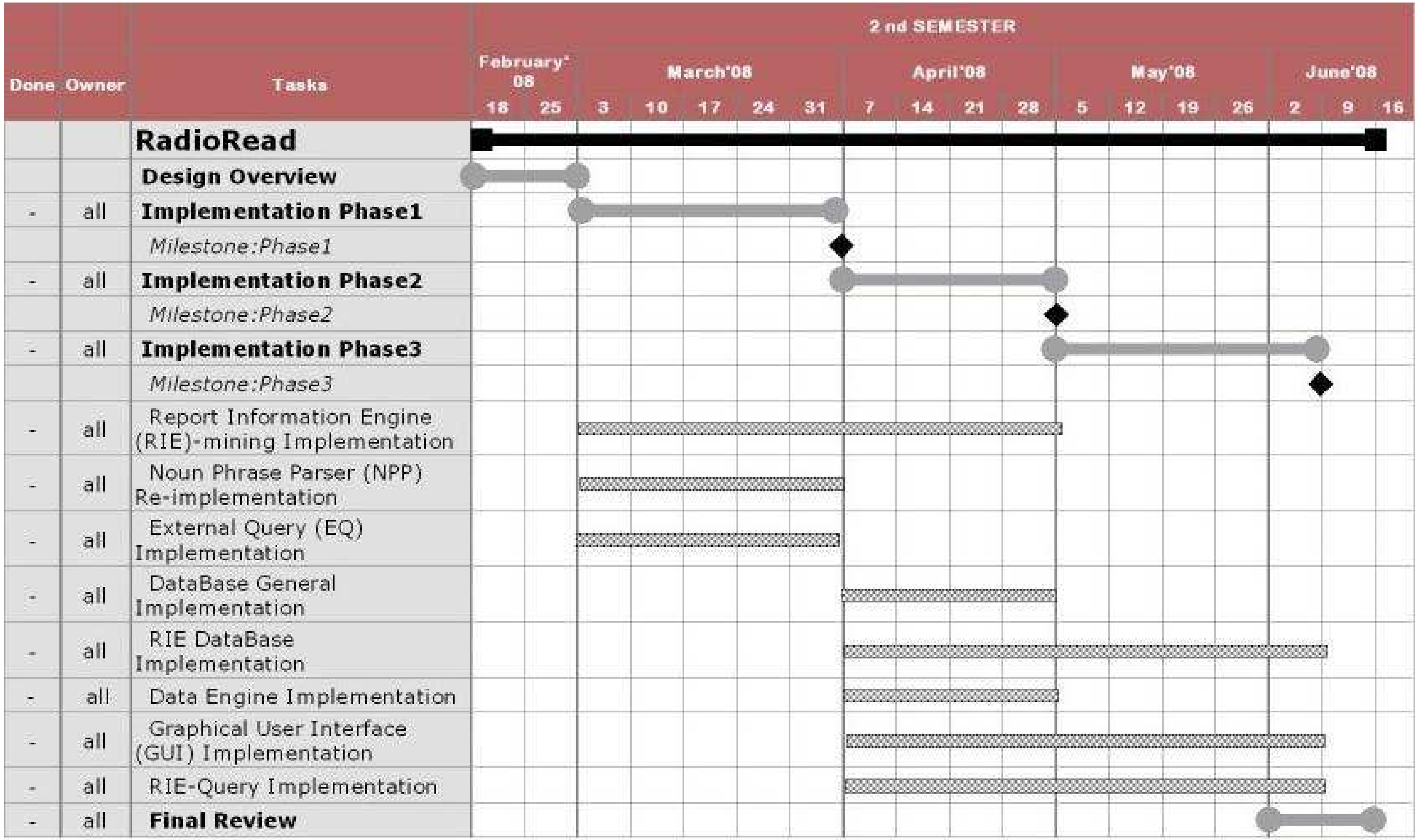
After every phase is completed, we will conduct tests and analysis to see our progress and success in mining the reports.

8.3. Gantt chart



Done	Owner	Tasks
		RadioRead
+	-	**Understanding the Project**
+	all	*Milestone: Project Proposal*
+	-	**Analysis**
+	all	Literature Survey
+	all	Meeting with Company
+	all	Requirement Analysis
+	Ipek+Cigdem	Data Modelling
+	Kerem	Functional Modelling
+	Makbule	Use Case Modelling
+	all	*Millestone: Requirement Analysis*
	-	**Design**
+	Kerem+Makbule	Data representation Design
+	all	Architectural Design
+	all	Requirement Analysis Review
+	Ipek + Cigdem	User Interface Design
+	Kerem	Component Level Design
+	all	*Milestone: Initial Design*
	-	**Detailed Architectural Design**
+	all	Design Review
+	Ipek + Cigdem	Detailed UI Design
+	Kerem+Makbule	Detailed Component Level Design
	all	*Milestone: Final Design Report*
	-	**Prototype Development**
+	all	Prototype Design
+	Ipek	Morpological Analyzing Tests
u	Makbule	~~Link Grammar Tests~~
+	Kerem	Dictionary Access Tests
+	Kerem+Ipek	Noun Phrase Syntax Analysis Tests
+	all	Sentence Syntax Analysis Tests
u	Cigdem	~~Semantic & SNOMED Tests~~
+	Cigdem+Kerem	ML Analyzing Tests
u	Makbule+Ipek	~~ML Search Tests~~
+	all	Implementing Prototype
	all	*Milestone: Prototype Demo*
	-	**Final Implementation**

*u =unsuccessful



Done	Owner	Tasks
		RadioRead
		Design Overview
-	all	**Implementation Phase1**
		Milestone:Phase1
-	all	**Implementation Phase2**
		Milestone:Phase2
-	all	**Implementation Phase3**
		Milestone:Phase3
-	all	Report Information Engine (RIE)-mining Implementation
-	all	Noun Phrase Parser (NPP) Re-implementation
-	all	External Query (EQ) Implementation
-	all	DataBase General Implementation
-	all	RIE DataBase Implementation
-	all	Data Engine Implementation
-	all	Graphical User Interface (GUI) Implementation
-	all	RIE-Query Implementation
-	all	**Final Review**

9. Coding Convention

1. Tab will be used as indentation unit.
2. In editors, tab length will be specified as 4 space characters for viewing. This is for cases when spaces will support tabs (in `if`'s and so on).
3. Class names will be `CamelCase`, starting with uppercase.
4. Method and member names will be `camelCase`, starting with lowercase.
5. Constants will be defined in `UPPER_CASE_AND_SEPARATED_WITH_UNDERLINES`. They should be able to explain themselves, but not too long (they can be much longer than a member / method name).
6. Method names will be imperative (`doSomething()`).
7. All members will be declared before methods.
8. `this` keyword will be used explicitly for accessing class members and methods (`this.blaBlaBla`).
9. No indentation before `import` statements.
10. First `private`s, then `public`s will be defined.
11. No public members, just methods.
12. Class declaration starts with no indentation, inside the class, there is at least one level of indentation.
13. If block sample:

```
if ( something )
{
  this.lalalala();
  other statements;
}

if ( something && another thing //line limit reached
     && another thing )
{
  this.lalalala();
  other statements;
}
else if ( anything )
{
     some statements;
}
else
     single statement;
```

14. Inside method body, there will be 2 levels of indentation (one for class one for method).

15. We will be using packages, and nearly every component will have multiple classes. Multiple classes in a single package will be preferred over a single class with many subclasses inside. Subclasses may only be used if it is very specific to the parent class.
16. Every class does a single job and does it best.
17. JavaDoc comments will be utilized.

10. Conclusion

This report includes the general aspects of our project and is also a guide for the reader to get the general idea of the project. During the preparation of this report, we have gained insight for our project. Some points that still seem ambiguous after Requirements Analysis Report are now clearer for the team with this report. Our project is scheduled to spend our effort more efficiently during all semester. Also we listed our general requirements to determine our basic functionalities and drew diagrams to make the implementation easier.

The process, from the beginning to the end, will be heavily-loaded and challenging, but we believe in the success of our team and our project. Our users will easily realize the difference of RadioRead when it takes its place in the market.

11. References

[1] Zemberek Library, *http://zemberek.googlecode.com*

[2] Zargan English Turkish Online Dictionary, with Roche Medical Dictionary, *http://www.zargan.com*

[3] TDK (Türk Dil Kurumu) Online Dictionary, *http://www.tdk.gov.tr*

Appendix A. Statistical Query Grammar

```
statistical_queries := how_many_query | percentage_query | measurement_graph_query

how_many_query := hasta_grubu

percentage_query := hasta_grubu hasta_grubu

measurement_graph_query := hasta_grubu bulgu ÖLÇÜM_TÜRÜ SAYI

hasta_grubu := yaş cinsiyet rapor_and

yaş := YAŞ “ile” YAŞ “aralığında olanlar” | Empty

cinsiyet := CİNSİYET | Empty

rapor_and := rapor_or rapor_and | Empty

rapor_or := rapor rapor_or | rapor

rapor := tarih sayı bulgu_and

tarih := TARİH “ile” TARİH “arasında yazılmış” | Empty

sayı := “koşullarına uyan en az” SAYI “en fazla” SAYI “tane” | Empty

bulgu_and := bulgu_or bulgu_and | Empty

bulgu_or := bulgu bulgu_or | bulgu

bulgu := yer_and bilgi

yer_and := yer_or yer_and | Empty

yer_or := yer yer_or | yer

yer := YER “bölgesinde” | Empty

bilgi := nasıl ölçüm ne varlık normallik önemlilik | Empty

nasıl := NASIL “durumundaki” | Empty

ölçüm := SAYI “ile” SAYI ÖLÇÜM_BİRİMİ “arasında ölçülmüş” | Empty

ne := NE “bulgusu”

varlık := TESPİT_EDİLMİŞTİR | TESPİT_EDİLMEMİŞTİR | Empty

normallik := NORMALDİR | ANORMALDİR | Empty

önemlilik := SONUÇTA_BAHSEDİLMİŞTİR | SONUÇTA_BAHSEDİLMEMİŞTİR | Empty
```

Appendix B. Noun Phrase Parser Grammar

```
Lexer'dan gelen Lexeme'ler:

ISIM
-IN
-I
,
VE
ILE

Ali’nin evinin pembe duvarı, mavi panjuru ve eflatun çatısı

ISIM -IN ISIM -I -IN ISIM ISIM -I , ISIM ISIM -I VE ISIM ISIM -I



Grammar:

TAMLAMA ->     ISIM_TAMLAMASI
               | SIFAT_TAMLAMASI

ISIM_TAMLAMASI -> TAMLAYAN TAMLANAN_GRUBU

SIFAT_TAMLAMASI -> ISIM ISIM | ISIM


BELIRTILI_TAMLAYAN_GRUBU ->
               BELIRTILI_TAMLAYAN_GRUBU , BELIRTILI_TAMLAYAN_GRUBU
               | BELIRTILI_TAMLAYAN_GRUBU ILE BELIRTILI_TAMLAYAN_GRUBU
               | BELIRTILI_TAMLAYAN_GRUBU VE BELIRTILI_TAMLAYAN_GRUBU
               | BELIRTILI_TAMLAYAN_GRUBU2

BELIRTILI_TAMLAYAN_GRUBU2 ->
               TAMLAMA -IN
               | SIFAT_TAMLAMASI ( , SIFAT_TAMLAMASI )* (VE|ILE) SIFAT_TAMLAMASI -IN


BELIRTISIZ_TAMLAYAN -> ISIM


TAMLAYAN -> BELIRTILI_TAMLAYAN_GRUBU
               | BELIRTISIZ_TAMLAYAN

TAMLANAN_GRUBU ->
               TAMLANAN_GRUBU , TAMLANAN_GRUBU
               | TAMLANAN_GRUBU ILE TAMLANAN_GRUBU
               | TAMLANAN_GRUBU VE TAMLANAN_GRUBU
               | TAMLANAN_GRUBU2


TAMLANAN_GRUBU2 ->
               TEK_TAMLANAN
               | SIFAT_TAMLAMASI ( , SIFAT_TAMLAMASI )* (VE|ILE) TEK_TAMLANAN

TEK_TAMLANAN -> SIFAT_TAMLAMASI –I
```

Appendix C. Create Table SQL Queries

```
CREATE TABLE Kullanicilar
                              (user_id INTEGER NOT NULL,
                              access_rights INTEGER NOT NULL,
                              username VARCHAR(16) NOT NULL,
                              password VARCHAR(8) NOT NULL,
                              active BOOL,
                              name VARCHAR(20) NOT NULL,
                              PRIMARY KEY(user_id));

CREATE TABLE Hastalar
                              (patient_id INTEGER NOT NULL,
                              name VARCHAR(32) NOT NULL,
                              surname VARCHAR(32) NOT NULL,
                              cinsiyet CHAR(1) NOT NULL,
                              year_of_birth DATE NOT NULL,
                              PRIMARY KEY(patient_id));

CREATE TABLE Doktorlar
                              (doctor_id INTEGER NOT NULL,
                              title VARCHAR(10) NOT NULL,
                              name VARCHAR(32) NOT NULL,
                              surname VARCHAR(32) NOT NULL,
                              PRIMARY KEY ( doctor_id));

CREATE TABLE Raporlar
                              (report_id INTEGER NOT NULL,
                              patient_id INTEGER NOT NULL,
                              user_id INTEGER NOT NULL,
                              title VARCHAR(255) NOT NULL,
                              rdate DATE NOT NULL,
                              clinical_info TEXT NOT NULL,
                              technical_info TEXT NOT NULL,
                              diagnosis TEXT NOT NULL,
                              findings TEXT NOT NULL,
                              results TEXT NOT NULL,
                              PRIMARY KEY ( report_id),
                              FOREIGN KEY ( patient_id) REFERENCES Hastalar,
                              FOREIGN KEY ( user_id) REFERENCES Kullanicilar);

CREATE TABLE Yazildi
                              (doctor_id INTEGER NOT NULL,
                              report_id INTEGER NOT NULL,
                              PRIMARY KEY ( doctor_id, report_id),
                              FOREIGN KEY ( doctor_id) REFERENCES Doktorlar,
                              FOREIGN KEY ( report_id) REFERENCES Raporlar);

CREATE TABLE Islenmis_Raporlar
                              (report_id INTEGER NOT NULL,
                              sure INTEGER,
                              sure_birimi VARCHAR(10) NOT NULL,
                              normallik BOOL,
                              PRIMARY KEY ( report_id),
                              FOREIGN KEY ( report_id) REFERENCES Raporlar);

CREATE TABLE Bulgular
                              ( bulgu_id INTEGER NOT NULL,
                              report_id INTEGER NOT NULL,
                              ne_id INTEGER NOT NULL,
                              yer_id INTEGER,
                              normal BOOL,
                              var BOOL,
                              sonucta_geciyor BOOL NOT NULL,
                              PRIMARY KEY ( bulgu_id),
                              FOREIGN KEY ( report_id) REFERENCES Raporlar,
                              FOREIGN KEY ( ne_id) REFERENCES Ne,
                              FOREIGN KEY (yer_id) REFERENCES Yer);

CREATE TABLE Bulgu_Yer
                              (bulgu_id INTEGER NOT NULL,
                              yer_id INTEGER NOT NULL,
                              uzaklik_olcum REAL,
                              uzaklik_birim VARCHAR(20) NOT NULL,
                              PRIMARY KEY ( bulgu_id, yer_id),
                              FOREIGN KEY ( bulgu_id) REFERENCES Bulgular,
                              FOREIGN KEY ( yer_id) REFERENCES Yer);


CREATE TABLE Bulgu_Olcum
                              (bulgu_olcum_id INTEGER NOT NULL,
                              bulgu_id INTEGER NOT NULL,
                              olcum REAL NOT NULL,
                              olcum_birim VARCHAR(20) NOT NULL,
                              tur INTEGER NOT NULL,     -- 0=uzaklik, 1=cap, 2=hiz …
                              PRIMARY KEY ( bulgu_olcum_id),
                              FOREIGN KEY ( bulgu_id) REFERENCES Bulgular);

CREATE TABLE Bulgu_Nasil
                              (bulgu_id INTEGER NOT NULL,
                              nasil_id INTEGER NOT NULL,
                              sonuctan BOOL NOT NULL,
                              PRIMARY KEY ( bulgu_id, nasil_id),
                              FOREIGN KEY ( bulgu_id) REFERENCES Bulgular,
                              FOREIGN KEY ( nasil_id) REFERENCES Nasil);

CREATE TABLE Yer
                              (yer_id INTEGER NOT NULL,
                              isim VARCHAR(50) NOT NULL,
                              PRIMARY KEY ( yer_id));

CREATE TABLE Yer_Rel
                              (birincil_yer_id INTEGER NOT NULL,
                              ikincil_yer_id INTEGER NOT NULL,
                              PRIMARY KEY ( birincil_yer_id, ikincil_yer_id),
                              FOREIGN KEY ( birincil_yer_id) REFERENCES Yer(yer_id),
                              FOREIGN KEY ( ikincil_yer_id) REFERENCES Yer(yer_id));

CREATE TABLE Nasil
                              (nasil_id INTEGER NOT NULL,
                              isim VARCHAR(50) NOT NULL,
                              PRIMARY KEY ( nasil_id));


CREATE TABLE Ne
                              (ne_id INTEGER NOT NULL,
                              isim VARCHAR(50) NOT NULL,
                              PRIMARY KEY ( ne_id));


CREATE TABLE Ne_Rel
                              (birincil_ne_id INTEGER NOT NULL,
                              ikincil_ne_id INTEGER NOT NULL,
                              PRIMARY KEY ( birincil_ne_id, ikincil_ne_id),
                              FOREIGN KEY ( birincil_ne_id) REFERENCES Ne (ne_id),
                              FOREIGN KEY ( ikincil_ne_id) REFERENCES Ne (ne_id));
```